Proceedings of Extreme Markup Languages®

Master Bibliography   Author Index   Topic Index   Date Index   Proceedings Home

On the Lossless Transformation of Single-File, Multi-Layer Annotations into Multi-Rooted Trees

Andreas Witt
Oliver Schonefeld
Georg Rehm
Jonathan Khoo
Kilian Evang

Abstract

The Generalised Architecture for Sustainability (GENAU) provides a framework for the transformation of single-file, multi-layer annotations into multi-rooted trees. By employing constraints expressed in XCONCUR-CL, this procedure can be performed lossless, i.e., without losing information, especially with regard to the nesting of elements that belong to multiple annotation layers. This article describes how different types of linguistic corpora can be transformed using specialised tools, and how constraint rules can be applied to the resulting multi-rooted trees to add an additional level of validation.

Keywords: Validating; Trees/Graphs; Concurrent Markup/Overlap

Table of Contents

Andreas Witt

Andreas Witt received his PhD in Computational Linguistics and Text Technology from the University of Bielefeld in 2002. After graduating in 1996, he started as a researcher and instructor in Computational Linguistics and Text Technology at Bielefeld University. He was heavily involved in the establishment of the minor subject Text Technology in Bielefeld University's Magister and B.A. program. In 2006 he moved to University of Tübingen, where he is engaged in a project on Sustainability of Linguistic Resources. Witt's main research interests deal with questions on the use and limitations of markup languages for the linguistic description of language data. He is a member of several research organizations, amongst them the TEI Special Interest Group on overlapping markup, for which he wrote parts of the latest version of the chapter "Multiple Hierarchies", which is included in TEI-Guidelines P5.

Oliver Schonefeld

Oliver Schonefeld has studied computer science at Bielefeld University, Germany until 2005. Since then he is working at the department of for computational linguistics and "text technology" in at Bielefeld University. Parts of this contribution deal with aspects of his forthcoming PhD thesis.

Georg Rehm

Georg Rehm works in Tübingen University's collaborative research centre Linguistic Data Structures in a project that develops the foundations for sustainable linguistic resources. He holds a PhD in Applied and Computational Linguistics and has been working with SGML and related technologies in the context of Natural Language Processing (especially with regard to text and corpus analysis as well as ontologies) since 1995.

Jonathan Khoo

Jonathan Khoo is a Masters student in the International Studies in Computational Linguistics program at University of Tübingen. He received his B.A. in Linguistics from Northwestern University in 1998. Between his studies, he worked as a web developer focusing on browser-based replacements for traditional rich-client applications. He is currently writing his M.A. thesis on one aspect of this paper.

Kilian Evang

Kilian Evang is a B.A. student of Computational Linguistics at the University of Tübingen. Furthermore, he works on sustainable linguistic resources.

XML Source PDF (for print) Author Package Typeset PDF

On the Lossless Transformation of Single-File, Multi-Layer Annotations into Multi-Rooted Trees

Andreas Witt [University of Tübingen]
Oliver Schonefeld [University of Bielefeld]
Georg Rehm [University of Tübingen]
Jonathan Khoo [University of Tübingen]
Kilian Evang [University of Tübingen]

Extreme Markup Languages 2007® (Montréal, Québec)

Note: This paper contains W3C MathML, which is not equally well supported in all browsers. If you have reason to think that mathematical expressions are not displaying properly, consult the PDF version (or try a different browser).

Introduction

Due to the complexity of this type of textual data, the annotation of linguistic corpora can be seen as a benchmark test for markup languages. In recent years [1], linguistic analyses became more and more detailed and linguists incorporated the results of these analyses into corpora. Furthermore, linguistic descriptions applied to texts are extremely heterogeneous. This is especially evident if the research belongs to different fields of linguistics, e.g., syntax, semantics, and phonology.

This contribution touches upon several different topics with regard to using XML-based markup languages for linguistic research. Our overall goal is the sustainable archiving of linguistic data. Corpora usually contain multiple annotation layers (morphology, part-of-speech, syntax, semantics, information structure, etc.). We devised a generalised architecture that, among other aspects, requires individual conceptual annotation layers contained in linguistic corpora to be separated from one another. To meet this prerequisite, we have to transform a linguistic corpus (normally represented by a single XML file, i.e., a single-rooted tree) into several XML files (i.e., a multi-rooted tree) so that each file contains one specific annotation layer. After a short introduction to the Generalised Architecture for Sustainability of linguistic data (GENAU; section 2), sections 3 and 4 describe two tools for the purpose of transforming different types of linguistic corpora into multi-rooted trees. While the separation of individual annotation layers can be considered an important and necessary step for the sustainable archiving of linguistic corpora, this process does remove potentially important element nesting information. For a transformation from single- to multi-rooted trees that is 100% lossless, we need to incorporate a mechanism that enables us to store information with regard to element nesting. The approach we introduce in this article is based on XCONCUR, which we briefly describe in section 5.

This contribution reports on work in progress within the project "Sustainability of Linguistic Data", funded by the German Research Foundation (DFG). It is an update to our Extreme Markup Languages 2006 Late Breaking paper **[Wörner et al. (2006)]**

Generalised Architecture for Sustainability (GENAU)

Since the late 1990s, practically all annotation formats for linguistic corpora have been realised as XML-based markup languages (see **[Sperberg-McQueen & Burnard (1994)]** , **[Lehmberg & Wörner (to appear)]** , **[Wagner (2005)]** for examples). They usually come in two different flavours – traditionally and in accordance with the built-in XML data model, most corpus markup languages form hierarchies that are expressed by nested element trees (for example, for the representation of syntactic constituents or document structures that are in practically all cases modelled based on the OHCO paradigm, i.e., text is supposed to be an ordered hierarchy of content objects, see **[Renear et al. (1993)]**).

In stark contrast to hierarchical data formats are markup languages that anchor a data set to a timeline (primarily used for the transcription of spoken language) – this approach borrows heavily from Bird and Liberman's Annotation Graphs **[Bird & Liberman (2001)]** . In timeline-based formats such as EXMARaLDA **[Schmidt (2001)]** , the annotator can draw an arc from one anchor to another point on the timeline. However, these upper-level structures are not represented by nested XML element-trees, but with the help of corresponding attribute-value pairs. At the same time and regardless of the hierarchical or timeline-based model, both approaches usually encode several annotation layers concurrently: a certain set of XML elements or attributes represents information on morphology, another set encapsulates syntactic information, while other elements encode linguistic data related to semantic or pragmatic structures. In our sustainability project we have to deal with both hierarchical and timeline-based corpora (examples can be found in **[Wörner et al. (2006)]**) and we have to provide the means for enabling users to query both types of resources in a uniform way **[Rehm et al. (2007)]** . In fact, the original annotation format will be irrelevant to the user, as the user interface and the underlying technology will abstract from any idiosyncrasies and peculiarities of the original corpus data formats.

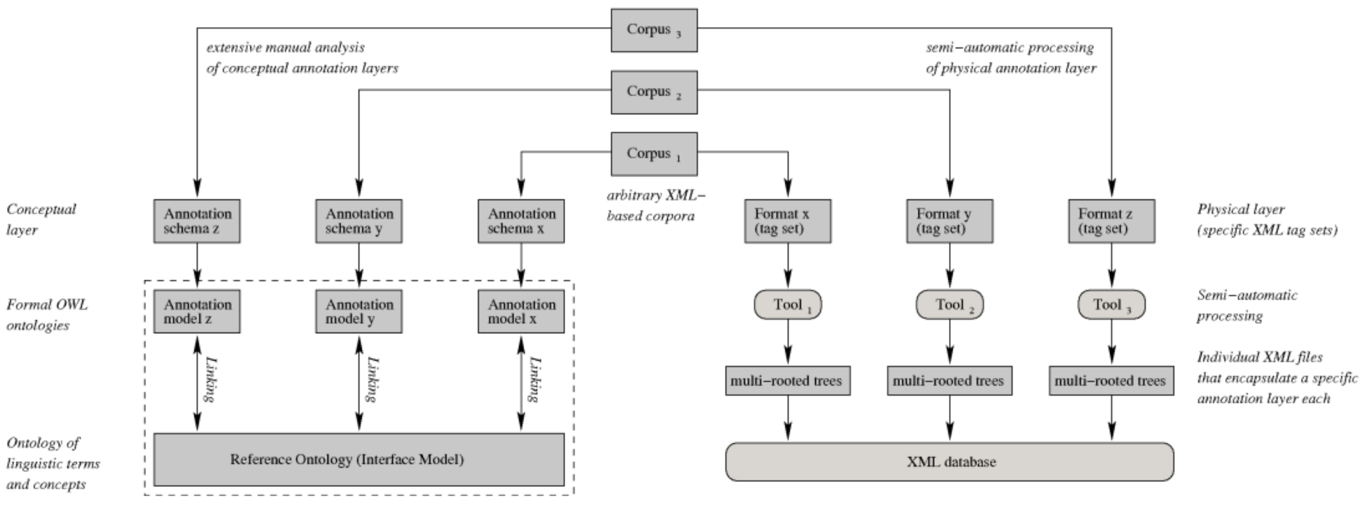
We devised an approach called GENAU (Generalised Architecture for Sustainability) that is able to cope with the abovementioned difficulties (**[Dipper et al. (2006)]** , **[Schmidt et al. (2006)]** , **[Wörner et al. (2006)]**) and that can be compared to the NITE Object Model **[Carletta et al. (2003)]** . The system architecture and our corpus processing workflow are depicted in figure 1. First, a corpus to be imported into our web-based corpus platform has to be analysed manually. Depending on the XML markup used in annotating the respective corpus, the XML document instance is transformed into multi-rooted trees. Some corpora can be transformed using simple XSLT stylesheets, while other corpora have to be processed using a custom set of tools (Tool$_1$, Tool$_2$ etc. in figure 1) with regard to this initial processing stage.

Corpora annotated based on the hierarchical model are analysed by a tool that enables us to map XML elements, attributes and textual content onto one or more annotation as well as primary or secondary data layers (see section 3). As soon as this mapping exists, the n annotation layers identified in the analysis can be exported as n XML document instances. In other words, this tool semi-automatically splits hierarchically annotated corpora that typically consist of a single XML document instance, into individual XML files, so that each file represents all the information related to a single linguistic annotation layer. Furthermore, this approach guarantees that overlapping structures – a notorious problem using single XML documents and, hence, single element trees – can be represented in a straightforward way **[Witt (2004)]** .

Timeline-based corpora (for example, EXMARaLDA corpora) are split using another tool in order to separate the graph annotations that are also stored in individual XML files (see section 4). Our approach enables us to represent arbitrary types of XML-annotated corpora as individual files that can be conceptualised as individual XML element trees. In a way, these multi-rooted trees are represented as regular XML document instances, but, since a single corpus is comprised of *multiple* files, there is a need to go beyond the functionality offered by typical XML tools in order to enable us to process multiple files, as regular tools work with single files only.

Finally, these single XML files are imported into a native XML database; currently we use the open source database eXist (see the right hand side in figure 1). A third tool anchors all files to a set of primary data in order to allow query-time coordination between the individual files that represent a single-rooted tree each (**[Eckart & Teich (2007)]** , **[Rehm et al. (2007)]**).

Figure 1: The two main phases of our corpus processing workflow


Corpus 3
Corpus 2
Corpus 1
extensive manual analysis of conceptual annotation layers
semi−automatic processing of physical annotation layer
arbitrary XML−based corpora
Conceptual layer
Annotation schema z
Annotation schema y
Annotation schema x
Format x (tag set)
Format y (tag set)
Format z (tag set)
Physical layer (specific XML tag sets)
Formal OWL ontologies
Annotation model z
Annotation model y
Annotation model x
Tool 1
Tool 2
Tool 3
Semi−automatic processing
Linking
multi−rooted trees
Individual XML files that encapsulate a specific annotation layer each
Ontology of linguistic terms and concepts
Reference Ontology (Interface Model)
XML database

[Link to open this graphic in a separate page**]**

At the same time, the elements and attributes used in the markup languages are analysed and incorporated into an ontology, represented in OWL (Web Ontology Language), that encapsulates knowledge about linguistic terms and concepts. The ontology is used to generalise over the specific and, at times, idiosyncratic names and labels used in the corpus annotation markup languages and to provide a coherent, unified, and homogeneous perspective on the large set of heterogeneous corpora (see the left hand side in figure 1 as well as **[Chiarcos (2007)]** , **[Rehm et al. (2007)]**). The OWL ontology is the main resource within our query interface: users can, for example, search for different combinations of part-of-speech tags. Usually, different corpora use different element and attribute names for encoding part-of-speech information, but the homogenising ontology of linguistic terms and concepts enables us to automatically expand a given query into all matching and appropriate element and attribute names. **[Rehm et al. (2007)]** describe this approach in detail.

Transforming Single Rooted Trees

The multiple annotation layers contained in a regular XML-annotated corpus may need to be separated in order to be transformed into other multi-hierarchical annotation formats such as XCONCUR (and to comply with the GENAU approach). We developed a pipeline called Leveler for such XML-document transformations. Its purpose is two-fold:

1. Moving certain PCDATA content to attributes (in order to separate PCDATA content that represents annotations from the actual primary data of the corpus which are, in practically all cases, also PCDATA).
2. Splitting the corpus into different files according to the different layers of annotation (e.g., syntactic, morphological, etc.).

The actual transformations are carried out using XSLT and are directed by configuration files created in a web application (also) called **Leveler**.

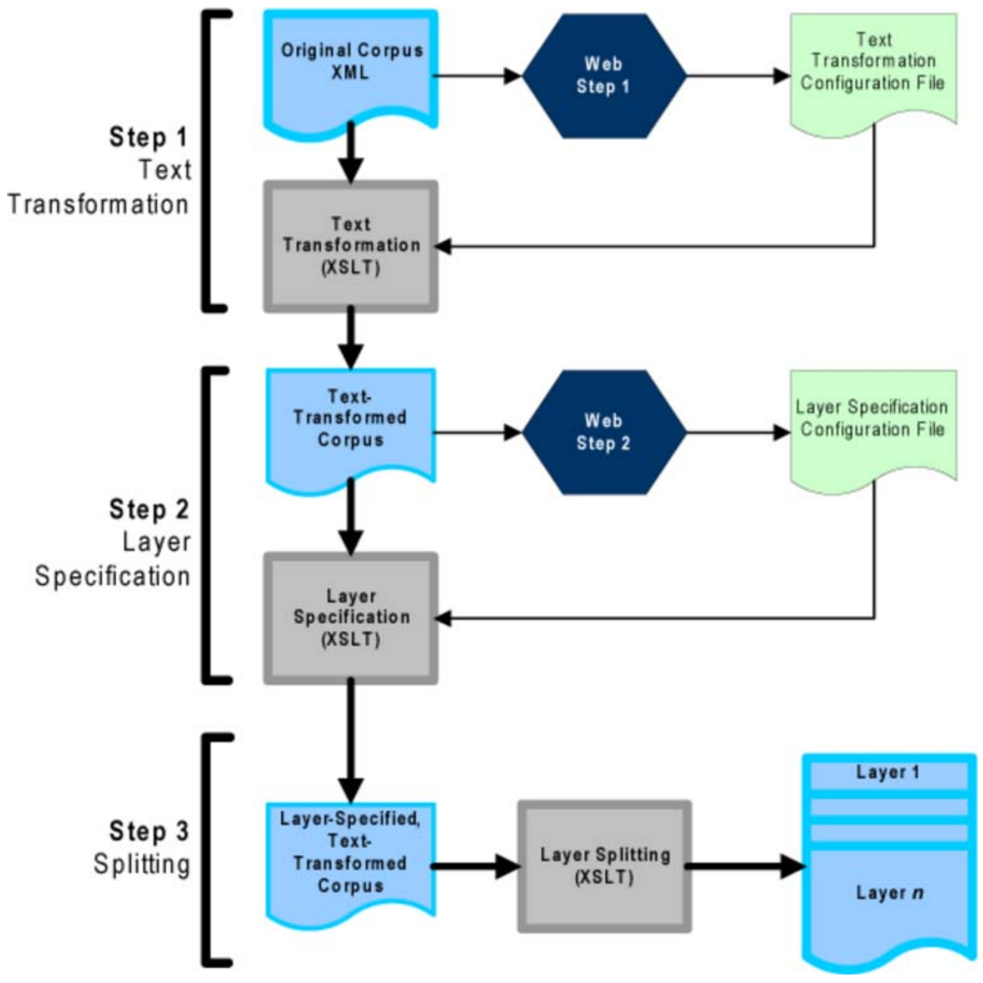
Leveler Pipeline

As shown in figure 2 there are two main steps to the Leveler process, each corresponding to one of the goals stated above. The original corpus (C_0) is used as input into Leveler to generate the Step 1 configuration file which contains directives on converting annotation information stored as PCDATA to attribute values. An XSL transformation then takes C_0 and the configuration file as input to generate a text-transformed corpus file, C_1. At this point, if one extracts the PCDATA from C_1, one should just get the straight text of the corpus without annotation information.

C_1 is itself used as input to another part of the Leveler tool to generate the Step 2 configuration file which specifies the different layers of annotation found in the corpus as well as which XML elements belong to them. Another XML transformation takes this configuration file and C_1 to generate C_2, a layer-annotated, text-transformed corpus. Layer information is simply added to the annotation elements as an attribute, and as such, the straight text of the corpus is preserved.

C_2 can then be fed into a final XSL transformation which creates an XML document for each layer, each containing only those nodes which belong to that layer. Each of these files will contain the same PCDATA information, although the branching of the element tree structure may be different. Leveler uses its own dedicated namespace for automatically-inserted elements.

Figure 2: The Leveler pipeline

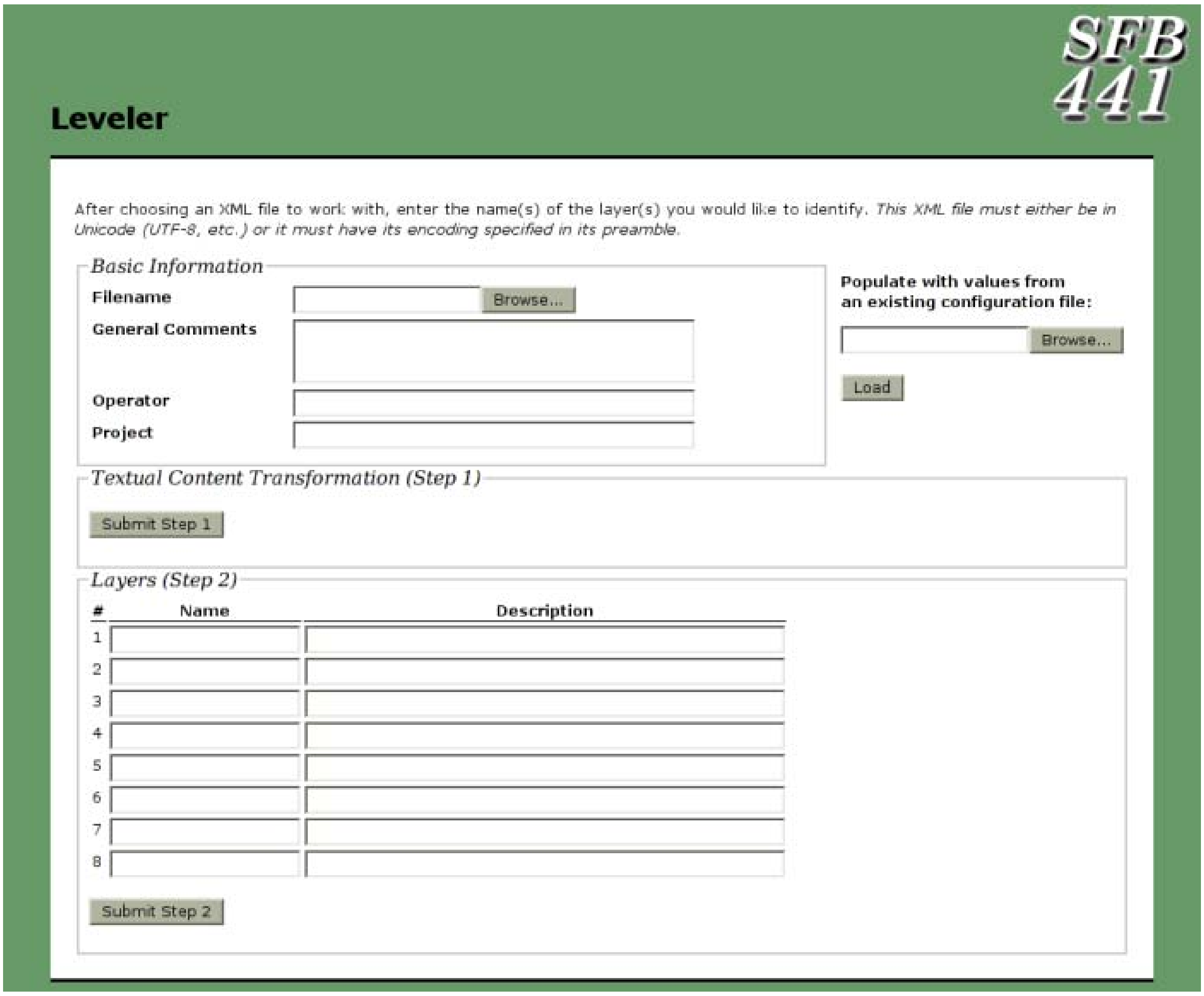

Step 1
Text
Transformation
Original Corpus XML
Web Step 1
Text Transformation Configuration File
Text Transformation (XSLT)
Step 2
Layer
Specification
Text-Transformed Corpus
Web Step 2
Layer Specification Configuration File
Layer Specification (XSLT)
Step 3
Splitting
Layer-Specified, Text-Transformed Corpus
Layer Splitting (XSLT)
Layer 1
Layer n

[Link to open this graphic in a separate page]

The Leveler Web Application

The bulk of the work is done with a web-based tool, Leveler, which takes user input in combination with an XML document to generate configuration files for use in the XSL transformations. Leveler is written in PHP with the SimpleXML extension and presents the user with a series of web forms.

Figure 3: Leveler main screen


Leveler

After choosing an XML file to work with, enter the name(s) of the layer(s) you would like to identify. *This XML file must either be in Unicode (UTF-8, etc.) or it must have its encoding specified in its preamble.*

Basic Information

Filename	Browse...
General Comments	
Operator	
Project	

Populate with values from an existing configuration file:

#	Name	Description
1		
2		
3		
4		
5		
6		
7		
8		

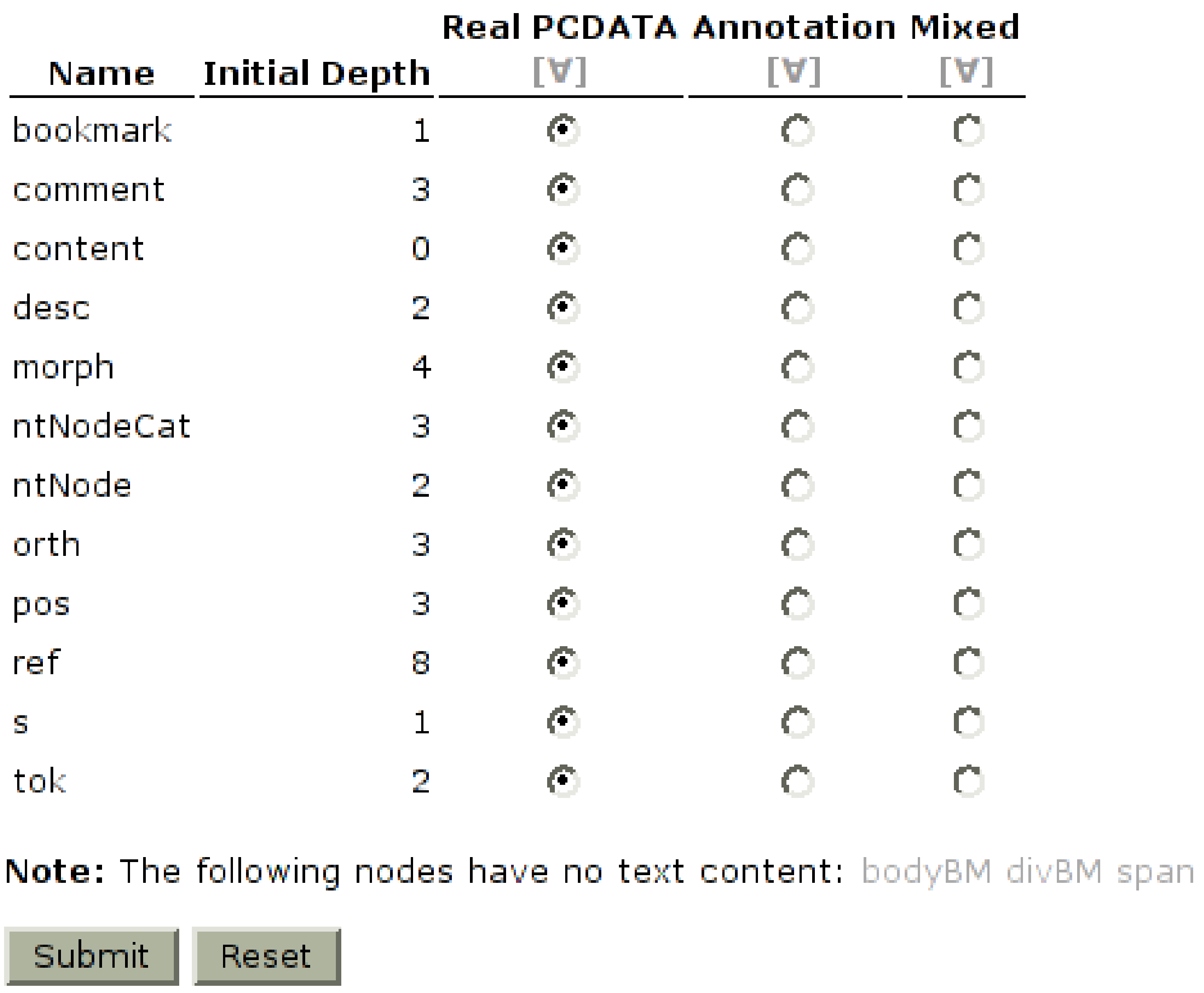
[Link to open this graphic in a separate page]

Figure 3 shows the main Leveler web form. The first fieldset, "Basic Information", is used for both Step 1 and Step 2. Here the user specifies the XML document filename and can optionally add Comment, Operator, and Project metadata to the resulting configuration file. To advance to Step 1, the user clicks the "Submit Step 1" button, uploading the XML file for server-side processing.

Leveler: Step 1

In the first step (figure 4), Leveler parses C_0 and returns a table that contains a row for each element in the XML document that has textual information (i.e., contains PCDATA); a list of elements that do not contain text nodes is listed at the bottom of the page for reference. For each element, the user selects one of three transformations by clicking on a radio button (the ∀ link at the top of each column is a shortcut that sets all elements to that transformation type). Upon submission of this form, Leveler generates an XML configuration file which contains a list of elements and how PCDATA within them should be handled.

Figure 4: Step 1 Leveler web form



Name	Initial Depth	Real PCDATA [∀]	Annotation [∀]	Mixed [∀]
bookmark	1	●	○	○
comment	3	●	○	○
content	0	●	○	○
desc	2	●	○	○
morph	4	●	○	○
ntNodeCat	3	●	○	○
ntNode	2	●	○	○
orth	3	●	○	○
pos	3	●	○	○
ref	8	●	○	○
s	1	●	○	○
tok	2	●	○	○

Note: The following nodes have no text content: bodyBM divBM span

[Link to open this graphic in a separate page]

The three text transformations are detailed in the following table with examples that show the transformation as applied to the following corpus XML fragment which contains words in `orth` elements and their parts of speech in `pos` elements:

Figure 5: Sample XML fragment

```
<tok id="s119n0">
    <orth>Wir</orth>
    <pos func="HD">PPER</pos>
</tok>
<tok id="s119n1">
    <orth>müssen</orth>
    <pos func="HD">VMFIN</pos>
</tok>
<tok id="s119n2">
    <orth>uns</orth>
    <pos func="HD">PRF</pos>
</tok>
<tok id="s119n3">
    <orth>selbst</orth>
    <pos func="HD">ADV</pos>
</tok>
<tok id="s119n4">
    <orth>helfen</orth>
    <pos func="HD">VVINF</pos>
</tok>
<punc>.</punc>
```

Types of Text Transformations (Step 1)

Real PCDATA

Identity transformation - all PCDATA nodes remain PCDATA. Note that in the example output the part of speech tags remain PCDATA nodes; extraction of text nodes results in the intermingling of POS labels: "Wir PPER müssen VMFIN uns PRF selbst ADV helfen VVINF ."

```
<tok id="s119n0">
    <orth>Wir</orth>
    <pos func="HD">PPER</pos>
</tok>
<tok id="s119n1">
    <orth>müssen</orth>
    <pos func="HD">VMFIN</pos>
</tok>
<tok id="s119n2">
    <orth>uns</orth>
    <pos func="HD">PRF</pos>
</tok>
<tok id="s119n3">
    <orth>selbst</orth>
    <pos func="HD">ADV</pos>
</tok>
<tok id="s119n4">
    <orth>helfen</orth>
    <pos func="HD">VVINF</pos>
</tok>
<punc>.</punc>
```

Annotation

Text data is converted to an introduced attribute `text` (in the Leveler namespace) of the containing element.

Example: `pos` set to "Annotation", the rest set to "Real PCDATA". This results in a fragment where the PCDATA is the pure corpus text: "Wir müssen uns selbst helfen .".

```
<tok id="s119n0">
    <orth>Wir</orth>
    <pos func="HD" leveler:text="PPER" />
</tok>
<tok id="s119n1">
    <orth>müssen</orth>
    <pos func="HD" leveler:text="VMFIN" />
</tok>
<tok id="s119n2">
    <orth>uns</orth>
    <pos func="HD" leveler:text="PRF" />
</tok>
<tok id="s119n3">
    <orth>selbst</orth>
    <pos func="HD" leveler:text="ADV" />
</tok>
<tok id="s119n4">
    <orth>helfen</orth>
    <pos func="HD" leveler:text="VVINF" />
</tok>
<punc>.</punc>
```

Mixed

Text data remains as PCDATA of the element but is also duplicated as an introduced attribute `text` (in the Leveler namespace).

Example: `pos` set to "Annotation", `punc` set to "Mixed", the rest set to "Real PCDATA". This also results in a fragment where the PCDATA is the pure corpus text: "Wir müssen uns selbst helfen .".

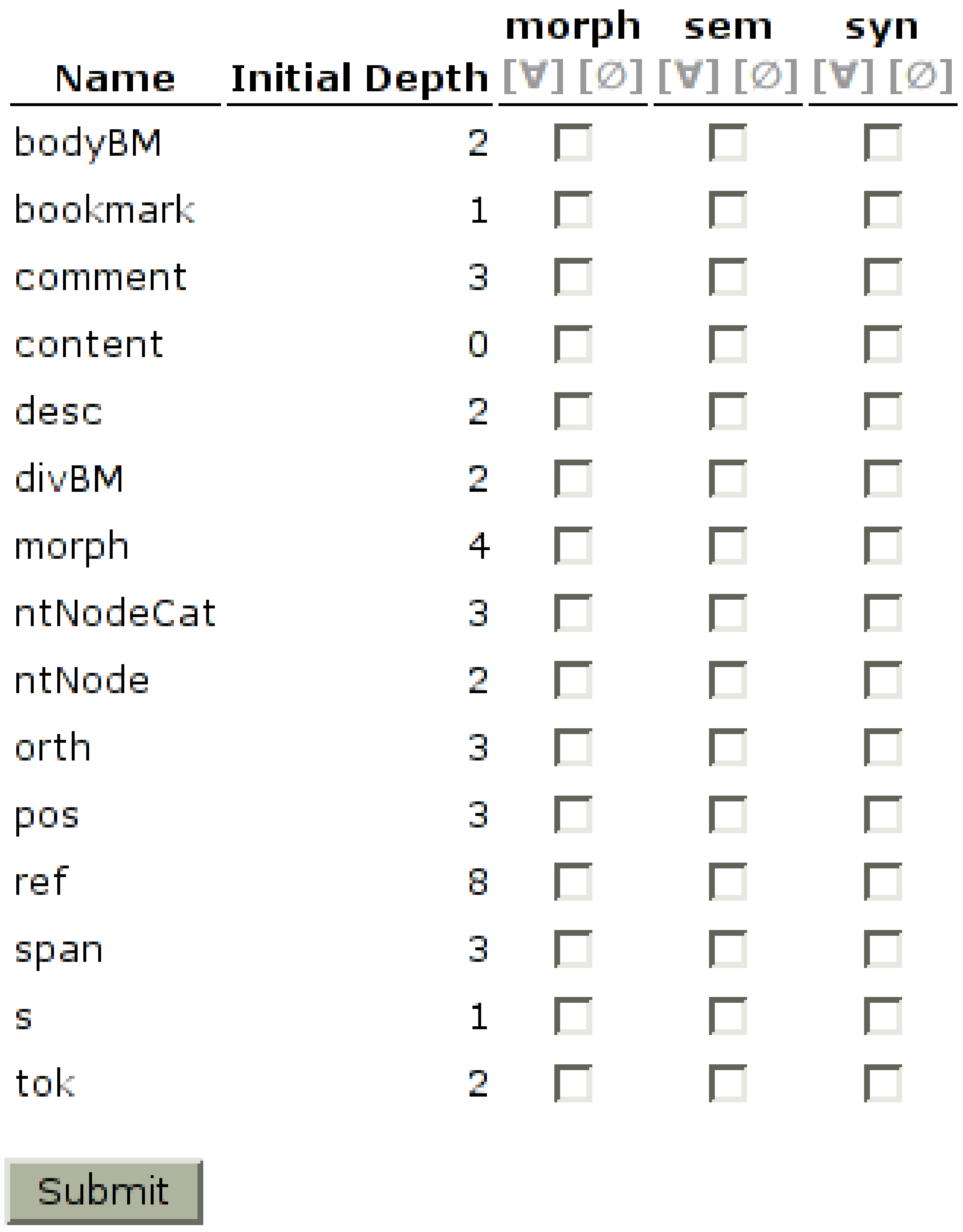
```
<tok id="s119n0">
    <orth>Wir</orth>
    <pos func="HD" leveler:text="PPER" />
</tok>
<tok id="s119n1">
    <orth>müssen</orth>
    <pos func="HD" leveler:text="VMFIN" />
</tok>
<tok id="s119n2">
    <orth>uns</orth>
```

```
        <pos func="HD" leveler:text="PRF" />
    </tok>
    <tok id="s119n3">
        <orth>selbst</orth>
        <pos func="HD" leveler:text="ADV" />
    </tok>
    <tok id="s119n4">
        <orth>helfen</orth>
        <pos func="HD" leveler:text="VVINF" />
    </tok>
    <punc leveler:text=".">.</punc>
```

Leveler: Step 2

In the second step, users enter a list of up to 8 annotation layers (e.g., "morph", "sem", "syn") that are found in the corpus before uploading the XML document (C_1) for analysis (figure 3). After the analysis of the document the user is again presented with a table where each row is a (unique) element as in figure 6. For each element the user can then specify which annotation layer(s) it belongs to (with shortcuts to assign all elements to or remove all elements from a particular layer). Like Step 1, submission of this form results in an XML configuration file which contains a list of layer names and a list of element names with layer membership information (IDREF to a layer ID).

Figure 6: Step 2 Leveler web form



		morph	sem	syn
Name	Initial Depth	[∀] [∅]	[∀] [∅]	[∀] [∅]
bodyBM	2	☐	☐	☐
bookmark	1	☐	☐	☐
comment	3	☐	☐	☐
content	0	☐	☐	☐
desc	2	☐	☐	☐
divBM	2	☐	☐	☐
morph	4	☐	☐	☐
ntNodeCat	3	☐	☐	☐
ntNode	2	☐	☐	☐
orth	3	☐	☐	☐
pos	3	☐	☐	☐
ref	8	☐	☐	☐
span	3	☐	☐	☐
s	1	☐	☐	☐
tok	2	☐	☐	☐

[Link to open this graphic in a separate page]

Because of the number of files and steps involved, a Java utility called **LevelRunner** provides a simple graphical user interface to running the XSL transformations. Users select which step they are on, then use file selectors to choose a source XML file, a configuration file (i.e., output from the web application), and an output directory. Clicking on a "Run" button shells out to Saxon to run the appropriate transformation. (Note that the actual command used is configurable so users are not locked into a particular processor.)

Considerations and Error Detection

The web application should be run on a local network (or on the local machine) due to the large amount of data that is usually transferred while uploading a corpus (which may be several tens, if not hundreds, of megabytes) to the web server for parsing by SimpleXML/PHP (which itself must be configured to handle very large HTTP Post requests). This design decision was taken because users could not be guaranteed to have a specific browser (e.g., one that supports XML loading and parsing), although a future switch to client-side XML document analysis is not out of the question. Performance so far has not been an issue, even with corpora several hundred megabytes large.

It could be the case that in Step 2, users do not assign the element(s) containing the text of the corpus (as PCDATA) to one or possibly all layers, meaning the text of the corpus is not the same from one layer to the next. This would cause standoff annotation methods such as XCONCUR to fail since segments will be missing and location information would differ for each layer. This could be solved by using the `normalize` program [2] that tries to normalize whitespace in one or more XML files and throws an error if normalisation is impossible (e.g., missing PCDATA nodes).

Transforming Annotation Graphs

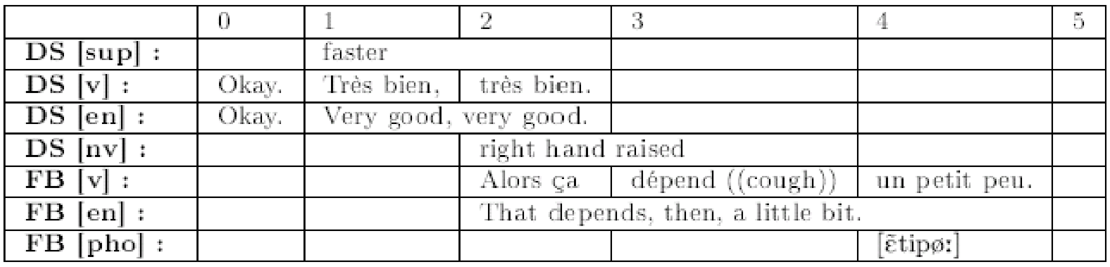
Annotation Graphs (AGs, [Bird & Liberman (2001)]) are used for the representation of multi-layered linguistic annotations. They allow for the declaration of an unlimited number of annotation layers and for an unrestricted linking of points in the text and in the annotation. The formal framework of Annotation Graphs is instantiated in different annotation tools and annotation models. A prominent application of AGs is EXMARaLDA. [Schmidt (2001)] . This section describes the transformation of EXMARaLDA into GENAU.

We developed Splitter, an XSLT-based tool that operates on documents in the time-based EXMARaLDA basic-transcription format and that converts them into a hierarchy-based format called *split-transcription*. The XML-based EXMARaLDA *basic-transcription* format uses a time-based data model, the *"single timeline, multiple tiers"* (STMT) data model ([Schmidt (2005)]). Information about speakers' verbal and nonverbal actions, other events, and various annotations referring to these actions and events is contained in several *tiers*. Each tier is identified by two attributes: The *category* of data it contains – orthographic transcription of verbal actions vs. phonetic transcription of verbal actions vs. description of nonverbal actions vs. annotation of manner of articulation of actions, and so on – and the *speaker* whose actions this particular tier contains; tiers of *speakerless* categories – e.g. for description of noises heard in the background – lack the speaker attribute, of course. Thus, actions of any two different speakers participating in the discourse are physically removed from each other in an EXMARaLDA file by being stored on different tiers, even if their category is the same. Also, actions of the same speaker but of different categories – e.g. Mary saying "I do not know" and shrugging at the same time – don't appear proximate in an EXMARaLDA file. Not even, e.g., a phonetic transcription appears in the vicinity of its orthographic counterpart. Rather, the temporal relation between events as well as the relations between annotations and physical events are expressed using a *common timeline*, to which all events from all tiers refer. Consider the following example EXMARaLDA *basic-transcription* (contents taken from [Schmidt (2005)]):

```
<?xml version="1.0" encoding="UTF-8"?>
<basic-transcription>
  <head>
    <meta-information>
      <!-- ... -->
    </meta-information>
    <speakertable>
      <speaker id="DS"/>
      <speaker id="FB"/>
    </speakertable>
  </head>
  <basic-body>
    <common-timeline>
      <tli id="T0" time="0.0"/>
      <tli id="T1"/>
      <tli id="T2"/>
      <tli id="T3"/>
      <tli id="T4"/>
      <tli id="T5"/>
    </common-timeline>
    <tier id="TIE0" speaker="DS" category="sup" type="a">
      <event start="T1" end="T3">faster</event>
    </tier>
    <tier id="TIE1" speaker="DS" category="v" type="t">
      <event start="T0" end="T1">Okay.</event>
      <event start="T1" end="T2">Très bien,</event>
      <event start="T2" end="T3">très bien.</event>
    </tier>
    <tier id="TIE2" speaker="DS" category="en" type="a">
      <event start="T0" end="T1">Okay.</event>
      <event start="T1" end="T3">Very good, very good.</event>
    </tier>
    <tier id="TIE3" speaker="DS" category="nv" type="d">
      <event start="T2" end="T4">right hand raised</event>
    </tier>
    <tier id="TIE4" speaker="FB" category="v" type="t">
      <event start="T2" end="T3">Alors ça</event>
      <event start="T3" end="T4">dépend ((cough))</event>
      <event start="T4" end="T5">un petit peu.</event>
    </tier>
    <tier id="TIE5" speaker="FB" category="en" type="a">
      <event start="T3" end="T5">That depends, then, a little bit.</event>
    </tier>
    <tier id="TIE6" speaker="FB" category="pho" type="a">
      <event start="T4" end="T5">[ɛ̃tipø:]</event>
    </tier>
  </basic-body>
</basic-transcription>
```

As can be seen in this example, events are connected indirectly via *time line items* (TLIs) on the common timeline. That, for example, the first event, "faster", refers to the events "Très bien" and "très bien" can be inferred from the common start and end attributes, together with all three events being on tiers with speaker DS. Likewise, that FB starts talking ("Alors ça") between DS's "très bien" utterances, is represented by the TLI T2, to which utterances of both speakers are anchored. These temporal relations allow for a two-dimensional visualisation of the transcription, such as the "musical score" notation that stacks tiers vertically and aligns events with respect to their start and end points in time:

Figure 7: Transcript in the "musical score" notation



	0	1	2	3	4	5
DS [sup] :		faster				
DS [v] :	Okay.	Très bien,	très bien.			
DS [en] :	Okay.	Very good, very good.				
DS [nv] :			right hand raised			
FB [v] :			Alors ça	dépend ((cough))	un petit peu.	
FB [en] :			That depends, then, a little bit.			
FB [pho] :					[ɛ̃tipø:]	

[Link to open this graphic in a separate page]

The purpose of Splitter is to transform this time-based format into a hierarchy-based format, where the relation between an event and the annotation describing it is not principally one of identical position on a timeline but a dominance relation in an ordered hierarchy. Since in the basic transcription, everything from "tape crackles" to the phonetic transcription of "un petit peu" ([ɛ̃tipø:]) is expressed as `event` elements, a distinction between "events" in the narrow sense and "annotations" describing them has to be introduced first. The user specifies a parameter called `skeletonCategory`. The events on tiers of this category are to become the leaves of a *multi-rooted tree* Splitter produces. Here one would probably choose `v`, the orthographic category. They will be treated as "events" in the narrow sense, everything else being "annotation".

In the initial step, Splitter takes all the events from all the tiers of the skeleton category and puts them into a single, chronologically-ordered sequence. The physical separation of utterances of different speakers is thereby removed. To keep speakers identifiable, the respective speaker ID is added as an attribute to each event. Temporally adjacent events belonging to the same speaker are treated as a coherent utterance and kept adjacent in the transcription, even if there are events belonging to other speakers during the utterance. As opposed to the *score*-like visualisation, the XML representation now resembles a *script* and can intuitively be read line by line:

```
<event id="1" speaker="DS" start="T0" end="T1">Okay.</event>
<event id="2" speaker="DS" start="T1" end="T2">Très bien,</event>
<event id="3" speaker="DS" start="T2" end="T3">très bien.</event>
<event id="4" trans="sync" ref="3" speaker="FB" start="T2" end="T3">Alors ça</event>
<event id="5" speaker="FB" start="T3" end="T4">dépend ((cough))</event>
<event id="6" speaker="FB" start="T4" end="T5">un petit peu.</event>
```

A script is less apt than a score to represent verbal actions taking place simultaneously. To facilitate the spotting of overlaps, Splitter annotates them explicitly. For example, event 4 comes after event 3 in the sequence of XML elements but occupies exactly the same position on the timeline, so there is an overlap. This is indicated by giving the "overlapping" event (4) two additional attributes, `ref` containing the ID of the "overlapped" element (3) and `trans` indicating the type of overlap. The following values are possible for the `trans` attribute of an event Y overlapping an event X:

- **overlap:** Classic overlap: Y starts during X and continues beyond the end of X.
- **sync:** X and Y share their start points and their end points (see example).

- **sync-shorter:** X and Y start at the same time, but Y ends before X.
- **sync-longer:** X and Y start at the same time, but Y ends after X.
- **within:** All Y takes place during X. X starts earlier and ends later or at the same time.

The "skeleton" sequence of events is the common ground for all that is to come – it will never be filtered, reordered, or changed in content any more, only marked up with elements annotating the events. In our case, we have annotation for suprasegmental features (`sup`), an English translation (`en`), nonverbal actions (`nv`), and (sporadic) phonetic transcription. The annotation is to be expressed as additional XML elements *containing* the events they describe. For example, to add markup for supersegmental features, Splitter adds `sup` elements as follows:

```
<sup> <event id="1" speaker="DS" start="T0"
        end="T1">Okay.</event>
</sup>
<sup value="faster" start="T1" end="T3">
  <event id="2" speaker="DS" start="T1" end="T2">Très bien,</event>
  <event id="3" speaker="DS" start="T2" end="T3">très bien.</event>
</sup>
<sup>
  <event id="4" trans="sync" ref="3" speaker="FB" start="T2" end="T3">Alors ça</event>
  <event id="5" speaker="FB" start="T3" end="T4">dépend ((cough))</event>
  <event id="6" speaker="FB" start="T4" end="T5">un petit peu.</event>
</sup>
```

The name of the tier category, `sup`, now serves as a name for the annotating elements. Each annotating element corresponds to either an event on a tier of the respective category in the source document (in this case, "faster" annotating events 2–3) or a chunk of unannotated "skeleton" events (in this case, 1 and 4–6).

Simply continuing like this and adding elements of other categories would go awry in the general case, because annotating events in the source document can overlap freely, whereas XML elements cannot overlap – a strict tree-structure is required. Splitter's solution is to create multiple result documents. Each result document contains annotating elements of just one category – hence the name "Splitter" – but all are completely identical with respect to the sequence of contained events. The set of the result documents can thus be interpreted as a multi-rooted tree, with different annotating nodes descending from different roots, but ultimately dominating a common sequence of leaves.

The result documents (in this case en.xml, nv.xml, pho.xml, sup.xml, and v.xml) are completed by one *split-transcription* meta-document, essentially the same as the source document, except the tiers are replaced with references to the respective documents:

```
<?xml version="1.0" encoding="UTF-8"?>
<split-transcription xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:f="http://www.sfb441.uni-tuebingen.de/c2/Kilian/XSLT/Funktionen">
  <head>
    <meta-information><!-- ... --></meta-information>
    <speakertable>
      <speaker id="DS">
        <abbreviation>DS</abbreviation>
      </speaker>
      <speaker id="FB">
        <abbreviation>FB</abbreviation>
      </speaker>
    </speakertable>
  </head>
  <split-body>
    <common-timeline>
      <tli id="T0" time="0.0"/>
      <tli id="T1"/>
      <tli id="T2"/>
      <tli id="T3"/>
      <tli id="T4"/>
      <tli id="T5"/>
    </common-timeline>
    <tier id="sup" category="sup" type="a" href="./split/example/sup.xml"/>
    <tier id="nv" category="nv" type="d" href="./split/example/nv.xml"/>
    <tier id="v" category="v" type="t" href="./split/example/v.xml"/>
    <tier id="en" category="en" type="a" href="./split/example/en.xml"/>
    <tier id="pho" category="pho" type="a" href="./split/example/pho.xml"/>
  </split-body>
</split-transcription>
```

XCONCUR

SGML (see [SGML (1986)] , [Goldfarb (1990)]) gave authors a facility to create documents with multiple, possibly overlapping, hierarchies in a rather easy way by its `CONCUR` option. The SGML specification has never been implemented completely, and, as a consequence, there is not a single SGML system we are aware of with full support for `CONCUR`. XCONCUR (introduced by [Hilbert et al. (2005)]) provides authors who are familiar with XML a standardized and intuitive way to write documents with overlapping markup. This is achieved by reviving SGML's `CONCUR` option and by introducing this option into XML.

A set of XML documents with identical primary data [3] can be transformed into XCONCUR, but, unfortunately, there is currently no tool to achieve this in a straightforward way. If, however, the set of XML documents is converted to a Prolog clause database, as defined within the architecture of the Sekimo project (see [Witt et al. (2005)]), this clause database can be converted to XCONCUR using `prolog2xconcur`.

Document Syntax

XCONCUR's document syntax is very similar to SGML with the `CONCUR` option set to `YES`. Each element is prefixed with an annotation layer id. This annotation layer id is used to assign the specific element to a distinct annotation layer. A *generic identifier* in terms of XCONCUR is the combination of annotation layer id and element name. It has the form `(<layer-id>)<name>`, where `<layer-id>` is the annotation layer id and `<name>` the element name. The annotation layer id must conform to the XML Namespaces (see [Bray at el. (2006b)]) rules for NCName while the element name must conform to the rules for QName.

Generally, XCONCUR is subject to the same restrictions XML (see [Bray et al. (2006a)]) imposes on SGML (e.g. elements must be closed, no tag minimization, etc). In contrast to SGML, no elements without an annotation layer id are allowed. Likewise, even if elements with the same element name occur on different annotation layers, each element has to be specified explicitly on its annotation layer by means of the annotation layer id.

Similar to XML, each XCONCUR document is required to be well-formed. This well-formedness is defined in terms of XML well-formedness. Each well-formed XCONCUR document can be projected (or decomposed) to a set of well-formed XML documents as follows: select an annotation layer and remove all annotations (tags) which do not belong to the selected layer. Then, remove all annotation layer ids, including the parentheses, and delete any XCONCUR processing instruction(s); repeat this procedure for the remaining annotation layers. An XCONCUR document is well-formed if all layers are well-formed in terms of XML.

XCONCUR currently defines two processing instructions. The schema processing instruction is a replacement for the deprecated DOCTYPE declaration [4] , which allows an author to assign an annotation schema to a specific annotation layer. The annotation schema may be written in any common schema language like DTD, XML Schema or RelaxNG. Furthermore, one or more constraint processing instructions can be used to associate constraint sets (see section 5-2) to the XCONCUR document. If an annotation schema is assigned to a layer, that layer can be checked for validity. However, an XCONUR document can only be considered valid if all layers are valid with regard to their annotation schema and if there are no violations to the constraint set.

The following example shows an excerpt of the Uppsala corpus of Russian documents (see [Lönngren et al. (1993)]) in a Latin transliteration as an XCONCUR-annotated document. The corpus, which is annotated using the Tusnelda standard, was preprocessed using the Leveler application and split into multiple XML files. These files were combined into a single XCONCUR document. For the sake of readability only two of the four layers are displayed in figure 8. Furthermore only one headline is shown and all of the coprus metadata has been omitted. The first layer (with the annotation layer id `l1`) encodes morphology while the second layer (with the annotation layer id `l2`) captures the document and sentence structure (headlines, paragraphs, and sentences). The smallest unit in the annotation is a word which is annotated in `tok` elements. On both layers the `tok` elements contain `orth` elements, which encode orthography. [5]

Figure 8: Excerpt of the Uppsala corpus as an XCONCUR document (reformatted for readability)

```
<?xconcur version="1.1" encoding="utf-8"?>
<?xconcur-schema layer-id="l1" root="tusneldaCorpus" system="tusnelda.dtd"?>
<?xconcur-schema layer-id="l2" root="tusneldaCorpus" system="tusnelda.dtd"?>
<(l1)tusneldaCorpus version="1.0"><(l2)tusneldaCorpus version="1.0">
  <!-- metadata sections omitted, here just an excerpt of the body -->
  <(l1)body id="SGID0201"><(l2)body id="SGID0201">
    <(l1)div type="unspecified"><(l2)div type="unspecified">
      <(l2)head type="main">
        <(l2)s id="SGID0201.1" n="1">
          <(l1)tok id="SGID0201.1.1" n="1">
            <(l2)tok id="SGID0201.1.1" n="1">
              <(l1)orth>
                <(l2)orth>Kakoj</(l2)orth>
              </(l1)orth>
              <(l1)pos leveler:text="pronoun"/>
              <(l1)desc>
                <(l1)feature type="subpos" leveler:text="interrogative"/>
                <(l1)feature type="tag" leveler:text="pronomen_int_nom_sg_masc_adj"/>
                <(l1)feature type="syntactic type" leveler:text="adjectival"/>
                <(l1)lemma leveler:text="kakoj"/>
                <(l1)case leveler:text="nominative"/>
                <(l1)gender leveler:text="masculine"/>
                <(l1)number leveler:text="singular"/>
              </(l1)desc>
            </(l2)tok>
          </(l1)tok>
          <(l1)tok id="SGID0201.1.2" n="2">
            <(l2)tok id="SGID0201.1.2" n="2">
              <(l1)orth>
                <(l2)orth>socializm</(l2)orth>
              </(l1)orth>
              <(l1)pos leveler:text="noun"/>
              <(l1)desc>
                <(l1)feature type="subpos" leveler:text="common"/>
                <(l1)feature type="tag" leveler:text="substantiv_masc_sg_nom_unb"/>
                <(l1)feature type="animacy" leveler:text="inanimate"/>
                <(l1)lemma leveler:text="socializm"/>
                <(l1)case leveler:text="nominative"/>
                <(l1)gender leveler:text="masculine"/>
                <(l1)number leveler:text="singular"/>
              </(l1)desc>
            </(l2)tok>
          </(l1)tok>
          <(l1)tok id="SGID0201.1.3" n="3">
            <(l2)tok id="SGID0201.1.3" n="3">
              <(l1)orth>
                <(l2)orth>narodu</(l2)orth>
              </(l1)orth>
              <(l1)pos leveler:text="noun"/>
              <(l1)desc>
                <(l1)feature type="subpos" leveler:text="common"/>
                <(l1)feature type="tag" leveler:text="substantiv_masc_sg_gen_gen2_unb"/>
                <(l1)feature type="subcase" leveler:text="gen_2"/>
                <(l1)feature type="animacy" leveler:text="inanimate"/>
                <(l1)lemma leveler:text="narod"/>
                <(l1)case leveler:text="genitive"/>
                <(l1)gender leveler:text="masculine"/>
                <(l1)number leveler:text="singular"/>
              </(l1)desc>
            </(l2)tok>
          </(l1)tok>
          <(l1)tok id="SGID0201.1.4" n="4">
            <(l2)tok id="SGID0201.1.4" n="4">
              <(l1)orth>
                <(l2)orth>nuzhen</(l2)orth>
              </(l1)orth>
              <(l1)pos leveler:text="adjective"/>
              <(l1)desc>
                <(l1)feature type="tag" leveler:text="adjektiv_masc_sg_kurzform"/>
                <(l1)feature type="Adjform" leveler:text="short form"/>
                <(l1)lemma leveler:text="nuzhnyj"/>
                <(l1)gender leveler:text="masculine"/>
                <(l1)number leveler:text="singular"/>
                <(l1)degree leveler:text="positive"/>
              </(l1)desc>
            </(l2)tok>
          </(l1)tok>
          <(l1)tok id="SGID0201.1.5" n="5">
            <(l2)tok id="SGID0201.1.5" n="5">
              <(l1)orth>
                <(l2)orth>?</(l2)orth>
               </(l1)orth>
              <(l1)pos leveler:text="interpunction"/>
              <(l1)desc>
                <(l1)feature type="tag" leveler:text="satzzeichen_fragezeichen"/>
                <(l1)feature type="type" leveler:text="question"/>
                <(l1)lemma leveler:text="?"/>
              </(l1)desc>
            </(l2)tok>
          </(l1)tok>
        </(l2)s>
      </(l2)head>
    <!-- ... -->
    </(l2)div></(l1)div>
  </(l2)body></(l1)body>
```

```
  <!-- ... -->
</(l2)tusneldaCorpus></(l1)tusneldaCorpus>
```

Validation

As mentioned in section 5-1, XCONCUR allows the use of annotation schemas for the validation of each annotation layer. However, to examine each tree by itself (a "tree-by-tree" approach) is not sufficient to validate multiple hierarchies in a proper manner. [6] XCONCUR-CL, the validation component in XCONCUR, is a constraint-based, cross-layer validation approach for the validation of multiple hierarchies. A constraint schema consists of a set of rules which define relations between an arbitrary number of layers in an XCONCUR document. Initial and related work was presented by **[Schonefeld & Witt (2006)]** and **[Schonefeld (2007)]** . Different approaches to the validation of conncurrent markup are described by **[Sperberg-McQueen (2006)]** and **[Tennison (2007)]** .

Basic constraint expressions

A basic constraint expression serves as the most fundamental building block of a rule in an XCONCUR-CL schema. The general form is `operand operator operand`. The operand allows the user to refer to the elements in an XCONCUR document while the operator defines the relation between these elements. The operand takes a parametrized generic identifier (pGI) as an argument that does not contain a literal annotation layer id, but an annotation layer id variable. This variable will be bound to a literal annotation layer id using the constraint processing instruction in an XCONCUR instance. All annotation layer id variables that occur in a constraint set have to be bound. This approach allows an author to write more generic and reusable constraint sets.

The following basic operands and operators are defined:

Operands

start[*pGI*]
: The *start* operand denotes the *start tag* of an element identified by pGI.

end[*pGI*]
: The *end* operand denotes the *end tag* of an element identified by pGI.

element[*pGI*]
: The *element* operand denotes the *element tag* pGI.

Operators

$op_a << op_b$
: The *precedes* operator: the position of entity op_a precedes the position of entity op_b. Entities op_a and op_b may be the *start* or *end* operand. The precedes operator realizes a strict precede relation and *not* a precedes-or-equals relation.

$op_a == op_b$
: The *equals* operator: the position of entity op_a equals the position of entity op_b. Entities op_a and op_b may be the *start* or *end* operand.

Each basic constraint expression will either evaluate to *true* or *false*. While evaluating the expression, the start and end operands will evaluate to a position of the corresponding tag. This position is an ordinal value defined as the offset into the primary data where this element occurs in the document. For non-empty elements the start position is always less than the end position, and for empty elements both positions are always equal.

Basic constraint expressions can be connected using logical operators and explicit grouping to form more powerful expressions.

exp_a && exp_b
: *Logical conjunction*: the whole expression evaluates to true, if each of the subexpressions exp_a and

exp_b evaluate to true.

exp_a || exp_b
: *Logical disjunction*: the whole expression evaluates to true, if any of the subexpressions exp_a and exp_b evaluates to true.

!exp
: *Negation*: the whole expression evaluates to true, if the expression exp evaluates to false.

The following table defines the precedences of the different components of basic constraint expressions and the logical operators. An XCONCUR-CL-aware parser should use them to build a correct representation of the constraint rules that are used to validate an XCONCUR document.

Table 1

Precedence	Operator	Comment
1	== <<	applies to derived operators as well
2	(expression)	explicit grouping
3	!	negation
4	&&	conjunction
5	||	disjunction

Common derived operators

Using the basic constraint expression and combining them using the logical operators can quickly lead to complex and practically unreadable constraint expressions. To countervail this fact, we defined a set of the most commonly used operators. The *common derived operators* are solely defined in terms of the basic operators combined by utilizing the logical operators.

The set contains the following operators:

op_a >> op_b
: The *succeeds* operator: the position of entity op_a succeeds the position of entity op_b. Entities op_a and op_b may be the *start* or *end* operand.

  !(op_a << op_b || op_a == op_b)

op_a <= op_b
: The *precedes-or-equals* operator: the position of entity op_a precedes or equals the position of entity op_b. Entities op_a and op_b may be the *start* or *end* operand.

  op_a << op_b || op_a == op_b

op_a => op_b
: The *succeeds-or-equals* operator: the position of entity op_a succeeds or equals the position of entity op_b. Entities op_a and op_b may be the *start* or *end* operand.

  !op_a << op_b || op_a == op_b

op_a [] op_b
: The *inside* or *inclusion* operator: entity op_a is contained inside entity op_b. Entity op_a may be the *start*, *end* or *element* operand and entity op_b must be an *element* operand.

  start(op_b) << op_a && op_a << end(op_b) | op_a in {start, end}

$start(op_b) << start(op_a)$ && $end(op_a) << end(op_b) \mid op_a$ in {element}

op_a][op_b

The *outside* or *independence* operator: entity op_a is not enclosed within the range between the start and end tag of entity op_b. Entity op_a may be the *start*, *end* or *element* operand and entity op_b must be an *element* operand.

!($start(op_b) << op_a$ && $op_a << end(op_b)$) | op_a in {start, end}

!($start(op_b) << start(op_a)$ && $end(op_a) << end(op_b)$) | op_a in {element}

op_a // op_b

The *overlap* operator: entity op_a overlaps with the range between the start and end tag of entity op_b. Entities op_a and entity op_b must be an *element* operand.

($start(op_a) << start(op_b)$ && $start(op_b) << end(op_a)$ && $end(op_a) << end(op_b)$) ||
($start(op_b) << start(op_a)$ && $start(op_a) << end(op_b)$ && $end(op_b) << end(op_a)$)

Rule Evaluation

Each constraint expression is evaluated in a specific context. To define a context, an addressing language such as XPath could be used. However, XPath would need to be extended to deal with multiple hierarchies. **[Alink et al. (2006)]** and **[Eckart & Teich (2007)]** propose new axes to XPath for this purpose, but for the purpose of simplicity XCONCUR-CL currently uses an approach adopted from the selectors in Cascading Style Sheets (see **[Bos et al. (2006)]**).

An element in an XCONCUR document can be seen as a range over the primary data, which is similiar to the approach used in LMNL (**[Tennison & Piez (2002)]**). The context of a constraint expression selects an element and spans a range over the primary data. The constraint expression is evaluated for all elements inside this range. If no specific element is chosen the *universal context*, denoted as `*`, can be used.

Figure 9 shows an excerpt of an XCONCUR document that is annotated on three different levels: a syntactic layer (sentences, words, using the annotation layer id `s`), a morphological layer (morphemes, annotation layer id `m`) and a phonological layer (syllables, annotation layer id `p`. The excerpt only shows the annotation of a single word ("tables"). Morphemes and words are annotated on distinct layers, since they may or may not overlap. However, both are always included inside of words. A set of constraints expressions is given in figure 10

Figure 9: Excerpt of a verbose XCONCUR annotation of words, morphemes and syllables.

```
<!-- ... -->
<(s)sentence>
  <!-- ... -->
  <(s)word>
    <(m)morph><(p)syll>ta</(p)syll><(p)syll>ble</(m)morph><(m)morph>s</(p)syll></(m)morph>
  </(s)word>
  <!-- ... -->
</(s)sentence>
<!-- ... -->
```

Figure 10: A set of constraint expressions for the XCONCUR document shown in figure 9.

```
# rule 1
($S)word {
  (element[($P)syll] [] element[self]) ||
  (start[self] <= start[($P)syll] && end[($P)syll] <= end[self])
} assert

# rule 2
($S)word {
  (element[($M)morph] [] element[self]) ||
  (start[($M)morph] == start[self] && end[($M)morph] == end[self]) ||
  (start[($M)morph] == start[self] && end[($M)morph] [] element[self]) ||
  (start[($M)morph] [] element[self] && end[($M)morph] == end[self])
} assert
```

```
# rule 3
($)word [
  element[($P)syll] // element[($M)morph]
} optional

# rule 4
* {
  element[($P)syll ][ element[($S)word]
} reject

# rule 5
* {
  element[($M)morph ][ element[($S)word]
} reject
```

Rule 1 asserts that a `syll` element must be contained inside a `word` element or both elements must be equal. The use of `self` as a pGI substitutes the selected context element here. This is used to make rules less ambiguous. [7] The rule has to be interpreted as follows: *each* element `syll`, which falls into the range which is spanned by the context element `s`, must either be completely included (first clause of the disjunction) or be of equal range, or share the start or end point with the context element (second clause). For each element matched by the context element, the rule will be evaluated; each time the rule is evaluated, a different set of elements is considered. Rule 2 is analogous to rule 1 but works on `s` and `morpheme` elements.

Rule 3 declares overlap between `syll` and `morph` elements as optional in the context of `s`. Using the negation operator, one could for example forbid overlaps between those elements. In that case, the elements `syll` and `morph` are not allowed to overlap, but otherwise could be in any possible relation.

Rule 4 rejects the occurrence of `syll` anywhere (by means of the universal context) in the document, but inside elements. It is to be read as: if *any* element `syll` is not contained inside the range of a `word` element, reject the document. Together, rules 1 and 4 ensure that `syll` elements may only occur inside of `word` elements. Rule 5 implements a similar behavior for `morph` elements.

Compact Syntax

A compact syntax for XCONCUR-CL is given in figure 11 as an EBNF grammar. The notation is similar to the one used in **[Bray et al. (2006a)]** . Similar to RelaxNG, this syntax serves as compact syntax for XCONCUR-CL. An additional XML representation will be provided in the future. [8]

Figure 11: A Compact syntax for XCONCUR-CL. [9]

```
  constraint-set ::= comment* rule (rule | comment)*
            rule ::= context "{" expression "}" rule-modifier?
         comment ::= "#" Any character valid for comments
         context ::= "*" | tag-identifier (tag-identifier)*
   rule-modifier ::= "assert" | "reject" | "optional"
      expression ::= basic-expression
                     | expression connector expression
                     | "!" expression
                     | "(" expression ")"
basic-expression ::= operand operator operand
       connector ::= "&&" | "||"
        operator ::= basic-operator | derived-operator
  basic-operator ::= "<<" | "=="
derived-operator ::= ">>" | "<=" | "=>" | "[]" | "][" | "//"
         operand ::= ("start" | "end" | "element")
                     "[" (tag-identifier | "self" ) "]"
  tag-identifier ::= "(" layer-variable ")" element-name
  layer-variable ::= "$" [A-Z] ([A-Z] | [0-9])*
    element-name ::= A QName as defined in Bray et al. (2006b)
```

GENAU and XCONCUR

The sections “Transforming Single Rooted Trees” and “Transforming Annotation Graphs” demonstrate how different types of linguistic corpora can be transformed to a set of separately annotated, primary-data-identical XML documents. These XML documents serve as the GENAU format. Since the XCONCUR document format can be understood as an interwoven set primary-data-identical XML documents, a transformation from GENAU to XCONCUR can be done trivially and is a lossless operation. Furthermore, an XCONCUR document can again be split up and transformed back via GENAU to its original representation – or even a different one.

The XCONCUR representation allows for the utilisation of XCONCUR-CL to formulate constraint rules, which express the relations between the elements of different trees in the multi-rooted-trees data model. On the one hand, this serves as a mechanism to validate the markup and can improve corpora quality. On the other hand, these constraint rules allow for the formulation of principles which are the result of a deeper analysis.

Figure 12 shows an excerpt of figure 8. The following constraint rules can be applied to ensure that `tok` elements span over equal ranges:

```
($L1)body {
  start[($L1)tok] == start[($L2)tok] && end[($L1)tok] == end[($L2)tok]
} assert

($L2)body {
  start[($L1)tok] == start[($L2)tok] && end[($L1)tok] == end[($L2)tok]
} assert
```

The same holds for the `body` elements, but the constraint rules have been omitted, since they are analogous to the other rules.

Figure 12: Excerpt from the Uppsala corpus (see figure 8)

```
<!-- ... -->
<(l1)tok id="SGID0201.1.1" n="1">
  <(l2)tok id="SGID0201.1.1" n="1">
    <(l1)orth>
      <(l2)orth>Kakoj</(l2)orth>
    </(l1)orth>
    <(l1)pos leveler:text="pronoun"/>
    <(l1)desc>
      <(l1)feature type="subpos" leveler:text="interrogative"/>
      <(l1)feature type="tag" leveler:text="pronomen_int_nom_sg_masc_adj"/>
      <(l1)feature type="syntactic type" leveler:text="adjectival"/>
      <(l1)lemma leveler:text="kakoj"/>
      <(l1)case leveler:text="nominative"/>
      <(l1)gender leveler:text="masculine"/>
      <(l1)number leveler:text="singular"/>
    </(l1)desc>
  </(l2)tok>
</(l1)tok>
<!-- ... -->
```

Likewise, similar constraints can be applied to an XCONCUR representation of an EXMARaLDA document. An example of an EXMARaLDA document that has been converted to XCONCUR is shown in figure 13. Similar constraints, such as those from the previous example, can be applied. For example, one might want to assert that `orth` elements should span over the same range as the `syll` elements.

Figure 13: An excerpt of the XCONCUR version of the E3 corpus (SFB 538, Hamburg University)

```
<?xconcur version="1.1" encoding="utf-8"?>
<?xconcur-schema layer-id="l1" root="body" system="exmeralda1.dtd"?>
<?xconcur-schema layer-id="l2" root="body" system="exmeralda2.dtd"?>
<(l1)body><(l2)body>
  <(l1)orth id="TIE1.a0" s="TLI0" e="TLI1" value="Sí.">
    <(l2)syll id="TIE2.a0" s="TLI0" e="TLI1" value="[CV]">
      <(l1)ts number="1" speaker="CHI" tier="TIE0" n="sc" id="TIE0.sc0" s="TLI0" e="TLI1">
        <(l2)ts number="1" speaker="CHI" tier="TIE0" n="sc" id="TIE0.sc0" s="TLI0" e="TLI1">
          <(l1)ts n="e" id="TIE0.e0" s="TLI0" e="TLI1">
            <(l2)ts n="e" id="TIE0.e0" s="TLI0" e="TLI1">[dʃɪ]</(l2)ts>
          </(l1)ts>
        </(l2)ts>
      </(l1)ts>
    </(l2)syll>
  </(l1)orth>
  <(l1)orth id="TIE1.a1" s="TLI2" e="TLI3" value="Mira.">
    <(l2)syll id="TIE2.a1" s="TLI2" e="TLI3" value="[CV.CV:]">
         <(l1)ts number="2" speaker="CHI" tier="TIE0" n="sc" id="TIE0.sc1" s="TLI2" e="TLI3">
           <(l2)ts number="2" speaker="CHI" tier="TIE0" n="sc" id="TIE0.sc1" s="TLI2" e="TLI3">
              <(l1)ts n="e" id="TIE0.e1" s="TLI2" e="TLI3">
                 <(l2)ts n="e" id="TIE0.e1" s="TLI2" e="TLI3">[mi.ɹɐː]</(l2)ts>
              </(l1)ts>
           </(l2)ts>
         </(l1)ts>
     </(l2)syll>
   </(l1)orth>
   <!-- ... -->
</(l2)body></(l1)body>
```

Conclusion

This paper presents an approach for the conversion of complex linguistic resources into a generalized data representation. This representation can be transformed into an XCONCUR document, so that XCONCUR-CL rules can be used to control not only the validity of each single annotation layer but also to enforce constraints with regard to the interaction of different layers.

The data we have analysed in the project so far is structured in such a way, that most relations between annotation layers belong to the "equals" relation. We did not observe any kind of overlap as a large part of the data (i.e., the corpora annotated in the Tusnelda format) originates from single-layered XML documents. In addition, the EXMARaLDA data does not include information from multiple linguistic levels of description. However, since we are interested in a general information modelling approach, we did not only focus on the overlap problem. The option to define different relations between annotation layers, or, to be precise, between trees, leads to an increase in quality, since the document may now not only be validated from a single perspective. As a result, a new class of erroneous data can be detected.

If a new information model for a corpus is to be designed from scratch, one could, besides using or defining the appropriate annotation schemata for each annotation layer, apply the full expressiveness of XCONCUR-CL to model the relations between the different trees in the multi-rooted data model of XCONCUR.

Notes

1. Indeed, linguistic data has received attention by the markup community for many years now (e.g., [Sperberg-McQueen & Burnard (1994)] , [Witt (1998)] , [Rehm (1999)])

2. http://coli.lili.uni-bielefeld.de/Texttechnologie/Forschergruppe/Phase1/sekimo/python/

3. Two documents are considered to have identical primary data if one can remove all annotations (markup) and obtain the same sequence of characters.

4. DOCTYPE declarations were used in earlier XCONCUR versions. As they are specific to DTDs, they are considered deprecated.

5. Since the occurance of the `orth` element in a `tok` element is optional, a further optimization to reduce the size of the annotation could be to encode the `orth` element on a distinct layer only.

6. For example consider the annotation of a document on three annotation layers: "syntactic layer" (sentence, words), "phonological layer" (syllables), "morphological layer" (morphemes). One can assert that syllables and morphemes are always contained in words, but they may or may not overlap. Therefore, a cross-layer method is necessary for validation proper.

7. For example consider that `($S)word` is used instead of `self`. It would be unclear if the `word` element is the current context element or another word element which is inside of the range spanned by the context element.

8. The XML syntax for XCONCUR-CL defined in [Schonefeld & Witt (2006)] is deprecated, since it will not handle the new features of XCONCUR-CL.

9. Here, the rules for `comment` and `element-name` are defined in prose rather than formally.

Bibliography

[Alink et al. (2006)] Wouter Alink, Valentin Jijoun, David Ahn, Maarten de Rijke, Peter Boncz, Arjen der Vries: *Representing and Querying Multi-Dimensional Markup for Question Answering*, In: Proceedings of the 5th Workshop on NLP and XML (NLPXML-2006): Multi-Dimensional Markup in Natural Language Processing, Trento, 2006.

[Bird & Liberman (2001)] Steven Bird, Marc Liberman: *A Formal Framework for Linguistic Annotation*. In: Speech Communication 33,1/2, 2001.

[Bos et al. (2006)] Bert Bos, Tantek Celik, Ian Hickson, Haakon Wium Lie: *Cascading Style Sheets, Level 2 Revision 1*, World Wide Web Consortium, 2006.

[Bray at el. (2006b)] Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin: *Namespaces in XML 1.1*, World Wide Web Consortium, 2006.

[Bray et al. (2006a)] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, Francois Yergeau, John Cowan: *Extensible Markup Language (XML) 1.1*. World Wide Web Consortium, 2006, 2nd edition.

[Carletta et al. (2003)] Jean Carletta, Jonathan Kilgour, Tim O'Donnell, Stefan Evert, and Holger Voormann: *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), 2003.

[Chiarcos (2007)] Christian Chiarcos: *An Ontology of Linguistic Annotation: Word Classes and Morphology*. In: Proceedings of DIALOG 2007, Toronto, 2007.

[Dipper et al. (2006)] Stefanie Dipper, Erhard Hinrichs, Thomas Schmidt, Andreas Wagner, Andreas Witt: *Sustainability of Linguistic Resources*. In: Erhard Hinrichs, Nancy Ide, Martha Palmer, and James Pustejovsky (eds.): Proceedings of the LREC 2006 Satellite Workshop on "Merging and Layering Linguistic Information", Genoa, 2006.

[Eckart & Teich (2007)] Richard Eckart, Elke Teich: *An XML-Based Data Model for Flexible Representation and Query of Linguistically Interpreted Corpora*, In: Georg Rehm, Andreas Witt, Lothar Lemmnitzer (eds.), Data Structures for Linguistic Resources and Applications, Gunter Narr Verlag, Tübingen, 2007. pp. 327–336.

[Goldfarb (1990)] Charles F. Goldfarb: *The SGML Handbook*. Clandon Press, Oxford, 1990.

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

[Lehmberg & Wörner (to appear)] Timm Lehmberg, Kai Wörner: *Annotation Standards*. In: A. Lüdeling and M. Kytö, Corpus Linguistics, HSK, de Gruyter, Berlin/New York, in press

[Lönngren et al. (1993)] Lönngren, Lennart (eds.): *Chastotnyj slovar' sovremennogo russkogo jazyka. (A Frequency Dictionary of Modern Russian. With a Summary in English.)*, Acta Universitatis Upsaliensis, Studia Slavica Upsaliensia 32, Uppsala, 1993

[Rehm (1999)] Georg Rehm: *Automatische Textannotation: Ein SGML- und DSSSL-basierter Ansatz zur angewandten Textlinguistik*. In: H. Lobin (ed.): Text im digitalen Medium, Wiesbaden, Westdeutscher Verlag, 1999.

[Rehm et al. (2007)] Georg Rehm, Richard Eckart, Christian Chiarcos: *An OWL- and XQuery-Based Mechanism for the Retrieval of Linguistic Patterns from XML-Corpora*. In: Proceedings of Recent Advances in Natural Language Processing (RANLP 2007), Borovets, Bulgaria

[Renear et al. (1993)] Allen Renear, Elli Mylonas, David Durand: *Refining our Notion of What Text Really Is: The Problem of Overlapping Hierarchies*. http://www.stg.brown.edu/resources/stg/monographs/ohco.html, 1993.

[Schmidt (2001)] Thomas Schmidt: *The Transcription System EXMARaLDA: An Application of the Annotation Graph Formalism as the Basis of a Database of Multilingual Spoken Discourse*. In: S. Bird, P. Buneman, M. Liberman: Proceedings of the IRCS Workshop on Linguistic Databases, 2001. pp. 219–227.

[Schmidt (2005)] Thomas Schmidt: *Time-Based Data Models and the Text Encoding Initiative's Guidelines for Transcription of Speech.* In: Arbeiten zur Mehrsprachigkeit (Working Papers in Multilingualism), Serie B (62), Hamburg, 2005.

[Schmidt et al. (2006)] Thomas Schmidt, Christian Chiarcos, Timm Lehmberg, Georg Rehm, Andreas Witt, Erhard Hinrichs: *Avoiding Data Graveyards: From Heterogeneous Data Collected in Multiple Research Projects to Sustainable Linguistic Resources.*. In: Proceedings of the E-MELD workshop 2006, June, 22 2006, Ypsilanti.

[Schonefeld & Witt (2006)] Oliver Schonefeld, Andreas Witt: *Towards Validation of Concurrent Markup*. In: Proceedings of Extreme Markup Languages, Montreal, 2006.

[Schonefeld (2007)] Oliver Schonefeld: *XCONCUR and XCONCUR-CL: A Constraint-Based Approach for the Validation* *of Concurrent Markup*. In: Georg Rehm, Andreas Witt, Lothar Lemmnitzer (eds.), Data Structures for Linguistic Resources and Applications, Gunter Narr Verlag, Tübingen, 2007. pp. 347–356.

[SGML (1986)] ISO 8879:1986: *Text and Office Systems – Standard Generalized Markup Language (SGML)*. International Organization for Standardization, Geneva, 1986.

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

[Sperberg-McQueen (2006)] C. M. Sperberg-McQueen: *Rabbit/Duck Grammars: A Validation Method for Overlapping Structures*. In: Proceedings of Extreme Markup Languages, Montreal, 2006.

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

[Tennison (2007)] Jeni Tennison: *Creole: Validating Overlapping Markup*. In: Proceedings of XTech 2007, Paris, 2007.

[Wagner (2005)] Andreas Wagner: *Unity in Diversity: Integrating Differing Linguistic Data in TUSNELDA*. In: S. Dipper, M. Götze, M. Stede: Heterogeneity in Focus: Creating and Using Linguistic Databases, ISIS, Working Papers of the SFB 632, Potsdam, 2005.

[Witt (1998)] Andreas Witt: *TEI-based XML-Applications: Transcriptions*. In: ALLC-ACH 1998, Joint Conference of the ALLC and ACH, Debrecen, 1998.

[Witt (2004)] Andreas Witt: *Multiple Hierarchies: New Aspects of an old Solution*. In: Proceedings of Extreme Markup Languages, Montreal, 2004

[Witt et al. (2005)] Andreas Witt, Daniela Goecke, Felix Sasaki, Harald Lüngen: *Unification of XML Documents with Concurrent Markup*. In: Literary and Linguistic Computing 2005 20(1), 2005. pp. 103–116.

[Wörner et al. (2006)] Kai Wörner, Andreas Witt, Georg Rehm, Stefanie Dipper: *Modelling Linguistic Data Structures*. In: Proceedings of Extreme Markup Languages, Montreal, 2006.

On the Lossless Transformation of Single-File, Multi-Layer Annotations into Multi-Rooted Trees

Andreas Witt [University of Tübingen]
Oliver Schonefeld [University of Bielefeld]
Georg Rehm [University of Tübingen]
Jonathan Khoo [University of Tübingen]
Kilian Evang [University of Tübingen]
