KajukModules

User's Guide

Martin Homuth

KajukModules: User's Guide

Martin Homuth

Version ${project.version}

Table of Contents

List of Tables

Foreword

The intention of this document is first to give a guide to the user of how to use the here mentioned pepper modules and how to utilize a mapping performed by them. Second this document shall give a closer view in the details of such a mapping in a declarative way, to give the user a chance to understand how specific data will be mapped by the presented pepper modules.

Chapter 1. Overview

This project contains an importer for the Kasseler Junktionskorpus, see http://www.uni-giessen.de/kajuk/. With this importer the corpus can be mapped to Salt and processed further.

The import of the corpus utilizes a mapping to Salt with the use of a SAX-parser. The mapping itself includes the conversion from the corpus annotations to a ANNIS-usable token-based setting, which will be explained further in the following chapter.

Chapter 2. Corpus Properties

The KAJUK corpus is segmented into phrases or between conjunctions, thus the text is tokenized using the Salt implementation of the TreeTagger tokenizer, a small example of the tokenization process can be found below.

The annotations of the KAJUK corpus are realized in nested SGML tags, which are not adoptable to Salt, hence a key-value representation of those elements is necessary.

Conversion Procedure

Originally the corpus was available in a SGML format specifically developed for the KAJUK corpus and hence not convertible with existing tools of the Pepper framework. With this basis the following tasks needed to be done for a conversion.

1. Transform the SGML format into a XML conform format for further processing
2. Tokenization of the KAJUK text.
3. Development of a Pepper module for the mapping of the KAJUK corpus to Salt, which leads to inherent exportability to relANNIS, the relational format of ANNIS) and thereby the ability to map the corpus to a sustainable format.

Format Transformation

The conversion from the SGML format to real XML is a straight forward process and requires just a few annotation changes, but without these changes no standardized processing can be performed.

Tokenization

ANNIS is token-based (whereas a token can be a word, a sentence or a syllable) and the KAJUK corpus is originally segmented into phrases. With a simple mapping to ANNIS it is not possible to search for words. Therefore the text has to be tokenized with a tokenizer and while the Pepper framework includes an implementation of the TreeTagger tokenizer it is used for this project.

Let's show a simple example for tokenizing the corpus:

```
1  <line n="1">
2     <lb>
3      <VOR>Im Jahr 1640 umb das Neue Jahr da</VOR>
4      <praed>
5       <V ID="Fin"><VV>kamen</VV></V>
6      </praed> in das darmstädische Land <subj>eine Armate schwedisch
7     Volk.</subj></lb><newline></newline>
8    </line>
```

When tokenizing the above text, the elements are ignored and tokens for the non-elements are created. Accordingly following tokens would be created:

```
TOK1 = Im
TOK2 = Jahr
TOK3 = 1640
TOK4 = umb
TOK5 = ...
```

In a later processing step, these tokens receive annotations, as will be shown in section 2.1.3.

However, there are special cases as in the following example:

```
<line n="18"/>
<lb type="subj" n="30">
    <SUB IR="zero">dass</SUB>
    in ihm
    <subj>das <!--hier line-->unersch&uml;tterte Vertrauen auf jenes principium
        und das ruhige <!--hier line 19-->Dasitzen des in ihm Befangenen</subj>
    <praed><Phras>seinen erhabensten Ausdruck</Phras></praed>
    <praed><V ID="PII"><PV><!--hier line 20-->bekommen</PV></V></praed>
    <praed><V ID="Fin"><HV>habe,</HV></V></praed></lb>
<line n="21">
```

In this example two comments are inserted into the continuous text. (<!--hier line 19--> and <!--hier line 20-->) With the tokenization those comments need to be considered as a span-annotation which mark graphical line breaks in the original text. The same appears for page breaks in some cases.

However, there is a problem with this kind of annotation. As mentioned, there are two ways of representing a graphical line break. As in the example above, there are two <line> elements and three comments, whereas one comment does not show the actual line. There are two ways of interpreting a <line> element and a line break comment. Either they mean that the actual break happens at that position and the mentioned line starts there, or the mentioned line ends there. There are examples, where it is not clear which one it is as well as the situation that it is clear, but the meaning alternates.

The processing of the text is not possible with this inconsistency, therefore some modifications to the text need to be made. For representing graphical line breaks, the original elements as well as the comments need to be replaced with a consistent model. The solution are two new elements added to the text (<newline/> and <newpage/>).

Conversion Module

In ANNIS linguistic annotations are represented in attribute-value pairs. In ANNIS it is not possible to use nested annotations in terms of multiple consecutive annotations.

```
1 <praed><V ID="Fin"><MV>must</MV></V></praed>
```

Such annotation need to be mapped to attribute-value pairs in which nested values can be represented. The solution for that is a variety of key-value pairs *attached* to the tokens:

```
1  ID = "Fin"
2  pos_KAJUK = "MV"
3  satzglied = "praed"
4  satzglied_type = "V"
```

These key-value pairs are then connected with the token *must*, hence this token can be found via search queries that look for any of the pairs, e.g. a query for all tokens with the pair *ID="Fin"* returns amongst others the token *must*. A complete list of element to key-value conversion can be found below.

Another speciality of KAJUK is that the same informations can be found in the XML-hierarchy in different places.

```
1 <J IR="adv"><AP type="FOK">aber</AP></J>
2 <FOK>aber</FOK>
```

In this example the information that the word "aber" is a focusparticle "FOK" is realised by an attribute as well as an element. This is also needed to be considered in the mapping process. Furthermore the attribute "type=" represents very different informations along the corpus.

Table 2.1. Conversiontable

Annotation	Key-Value Pair	Annotation
<!--hier line-->	line="n"	<KON>

Annotation	Key-Value Pair	Annotation
<newpage>	page="n"	<SUB>
<lb>	lb="lb"	<J>
<lb IR="kom">	IR_lb="kom"	<J norm="x">
<lb IR="kond">	IR_lb="kond"	<V ID="x">
<lb type="subjattr">	type="subattr"	<FOK>
<lb type="subj">	type="subj"	type="FOK"
<lb type="WR">	type="WR"	<FV>
<lb ADDtype="WR">	ADDtype="WR"	<HMV>
<lb IR="fin">	IR="fin"	<HV>
<lb IR="kaus">	IR="kaus"	<Inf>
<lb IR="temp">	IR="temp"	<IP>
<lb IR="meta">	IR="meta"	<KOR>
<J ADDIR="x">	ADDIR="x"	type="KOR"
<J EB="prag">	EB="prag"	<KV>
<subj type="noE">	type="noE"	<LASSEN>
<J ID="3">	ID="3"	<MV>
<J IR="x">	IR="x"	<RF>
<AD>	J_type="AD"	<TUN>
<AP>	J_type="AP"	<VP>
<praed real="afin">	real="afin"	<VV>
<subj type="abs">	satzglied_type="abs"	<J type="x">
<AcI>	satzglied_type="Acl"	<AP ADDtype="x">
<ADJGr>	satzglied_type="ADJGr"	<XX>
<akk>	satzglied_type="akk"	<praed dir="R">
<NGr>	satzglied_type="NGr"	<subj type="E">
<obl>	satzglied_type="obl"	<SUB IR="zero">
<PGr>	satzglied_type="PGr"	<J type="E">
<Phras>	satzglied_type="Phras"	<SUB ADDtype="E">
<prae>	satzglied_type="prae"	<obj change="gen-nom">
<PV>	satzglied_type="PV"	<obl>
<ADV>	satzglied="ADV"	<AD>
<obj>	satzglied="obj"	<praed real="afin">
<SUB type="subj">	satzglied="subj"	<praed norm="und">
<SUB ADDtype="Eviol">	viol="Eviol"	<TUN>
<VOR>	Vorfeld="VOR"	

Ellipsis

An example for an ellipsis can be seen in the following listing.

```
1  <lb n="1,3000,2000">
2      <subj type="E" dir="V" change="indef-def,MF-VF">der Hertzog von Weinmar<
3   <praed>
4       <V ID="Fin"><KV>War</KV></V>
5      </praed><newline></newline>
```

```
6   <J IR="adv">
7   <AP>aber</AP>
8   </J>
9   <praed>
10   <V ID="PII"><VV>gestorben</VV></V>
11  </praed>
12 </lb>
```

The example shows the ellipsis *der Hertzog von Weinmar*, which is not part of the main text but in the process of conversion treated as normal text. Simply the created key-value pairs for the ellipsis do make them special.

The ellipsis of the corpus are identified by specific attributes of elements and as a result, because of the decision of keeping the elliptic text in the main text, the token annotations receive the extension "_E" (see table above) as well as the special annotation *E="E"*. With this every ellipsis can be identified with its context in Salt as well as in ANNIS.

Junctions

Junctions are identified by housenumbers in the actual format. With this phrases are combined into <lb> elements, so-called "Sachverhaltsdarstellungen". Phrases that belong to the same junctionbond receive the same housenumber. The correspondend sentential connective, which combines both phrases to such a bond, is contained in both phrases. This relationship is the core of the corpus and has to be converted into ANNIS.

For this the tokens are combined into one unit (the spans), which itself can be annotated with the housenumbers they receive. The relation between phrases can be realised in Salt and ANNIS using the concept of pointing relations. With pointing relations phrases can be connected and then be further annotated, e.g. with the kind of the sentential connective like 'KON'.

In the end the junctions are realised with spans for each <lb> element, which are connected via pointing relations for each house number. In doing so the connections are placed consecutive with the appearance of house numbers. For example the spans with the housenumber <lb n="1003"> are connected cocnsecutively, which means the first appearance is connected to the second and the second to the third.

Chapter 3. Module Usage

Because the importer is implemented in a static way, there are no customization options. Every document of the corpus will be parsed and mapped to Salt. Then the created Salt representation can be used for further processing.

Chapter 4. Implementation Details

This chapter covers the details of the importer implementation to create the Salt mapping for exporting the corpus to the RelANNIS format.

The characters() method

In the characters() method, the parsed characters array is appended to the STextualDS of the DocumentGraph while the involved part is tokenized with the Salt implementation of the TreeTagger. The created STokens are then added to the SLayers for the different categories, e.g. *junktionen* or *lexikalische_annotationen*.

Note

These layers weren't used in the laudatio project, the actual RelANNIS files were configured by hand, so the Salt equivalent for the RelANNIS exporter is not implemented yet.

Those layers are used for grouping in the ANNIS webinterface. In the end, every SToken's reference is saved for every element opened to create a SSpan later on.

Additionally, modifications for elliptic annotations are done here.

The startElement() method

When a new element is openend, several things are done:

1. Skipping *pb*- and *line*-elements completely
2. Determine multi-level *lb*-elements
3. Creating SSpans for *newline*- and *newpage*-elements
4. The key-value pair(s) for the opened element and it's attributes is created, see chapter 2.1.3. The conversion class created for this purpose simply defines the SAnnotation accordingly
5. The information about the opened element is saved

The endElement() method

As with the *startElement()* there are several things to do here:

1. Skipping *pb*- and *line*-elements completely
2. If the element is a *newline*- or *newpage*-element, the information is saved
3. A SSpan for each saved STokens within this element is created as well as it's annotation
4. For every house number the closed element may contain, a SSpan is created and saved for later use
5. The STokens and SSpans created are connected with SSpanningRelations

The endDocument() method

When the document is parsed, the importer connects the SSpans of the same house number with each other with the use of SSpanningRelations. This connection happens in a FIFO fashion, so each SSpan

points to the next occurance of the same house number and so on. Additionally hypotaxis and parataxis are marked with the letters *a* and *b* in the house number. In this case the last occurance of a house number (main sentence) points to every occurance of the house number with the addition.

Missing features

There are some things, that are not covered by the importer module, but had to be manipulated by hand for the Laudatio integration:

1. The metadata of the documents is not part of the importer itself, instead there is a separate file with the information

2. The grouping in ANNIS is not done with the created SLayers, but by hand.