Multi-level annotation of linguistic data with MMAX2

Christoph Müller and Michael Strube

EML Research gGmbH, Heidelberg

Abstract

This paper describes how richly annotated corpora can be created with the annotation tool MMAX2. The description is from the point of view of Computational Linguistics, a discipline where annotated corpora are often used as resources for software development. The paper outlines the important steps in the life cycle of an annotation and details how the tool MMAX2 can be employed in each of them.

1 Introduction

In Corpus Linguistics, corpora are used as machine-readable collections of naturally-occurring language data, serving as the empirical basis of frequency, collocation, and other analyses. But corpora are also used in other, related disciplines. Researchers in Computational Linguistics and Natural Language Processing (NLP) are developing software for automatically processing written and spoken natural language. This software often takes the form of components, i.e. self-contained modules for solving a particular well-defined and limited task. Some common tasks are

- determination of the word class / part of speech (POS) for words in a text (*POS tagging*, cf. Jurafsky and Martin 2000: 287-321),
- determination of word senses, incl. the disambiguation of homonymous and polysemous words (*word sense disambiguation*, cf. Manning and Schütze 1999: 229-264), and
- detection of anaphoric expressions and identification of their antecedents (*anaphor / coreference resolution*, cf. Mitkov 2002).

Being able to solve these tasks automatically is not an end in itself, but rather a prerequisite for building systems for solving more complex tasks like machine translation (cf. Nirenburg et al. 2003), summarization (cf. Mani 2001), or question answering (cf. Maybury 2004). A popular approach towards the development of NLP components like automatic POS taggers, word sense

disambiguators, and coreference resolvers is the use of supervised machine learning methods. Essentially, these methods analyse a corpus of individual examples of a particular task, and automatically derive strategies for solving it. Often, these strategies take the form of rules, or tree-like structures consisting of hierarchical sequences of *if-then* statements. The important point is that it is not sufficient to have a corpus of *examples* only. In order to be able to create rules for assigning POS tags or word senses to words, or antecedents to anaphoric pronouns, machine learning methods require the examples to also contain the correct *solution*. In other words, the application of supervised machine learning depends on the availability of corpora that are enriched with additional information. The process and result of adding this information to an existing corpus is called *annotation*.

In Computational Linguistics, annotated corpora are not only used for the development of NLP components, but also for their evaluation, i.e. for measuring their performance. This is done by comparing the (potentially wrong) solutions of a given component with the correct solutions contained in the corpus, and calculating a numerical index (e.g. *precision* and *recall*, or *accuracy*) for the degree of agreement between both.

Apart from being used as a resource for the development of NLP components, an obvious advantage of annotated corpora is that the explicit marking of linguistic phenomena makes them accessible for automatic analysis. While an unannotated corpus can only be analysed on the *surface*, i.e. on the word level, an annotated corpus can also access *deeper* linguistic phenomena and incorporate them into quantitative (frequency counts, collocations) and qualitative (e.g. KWIC indices) analyses.

The creation of an annotated corpus is a laborious, time-consuming and expensive task, which requires both intellectual and manual effort on the part of the human annotators. Specialized annotation tools are normally employed in order to support the annotators. One such tool is MMAX2 (http://mmax.eml-research.de), which has been developed in the context of real-world annotation projects in Computational Linguistics.[1]

MMAX2 is a highly customizable tool for creating, browsing, visualizing and querying linguistic annotations on multiple levels. The screenshot below gives a first impression of the MMAX2 main window. It shows a small part of dialog Bed017 of the ICSI Meeting Corpus (Janin et al. 2003), a collection of transcribed multi-party dialogs. This example will be used throughout the remainder of this paper. The dialogs in the ICSI Meeting Corpus are structured as a sequence of segments, each of which is associated with the ID of a speaker. In the display, each segment is given in one line, with the speaker ID in front.

[1] The development of MMAX2 has been funded by the Klaus Tschira Foundation (http://www.ktf.villa-bosch.de).

Some of the annotated elements are shown in square brackets. The lines connecting some of these elements visualize coreference annotation. All these features will be described in more detail later.

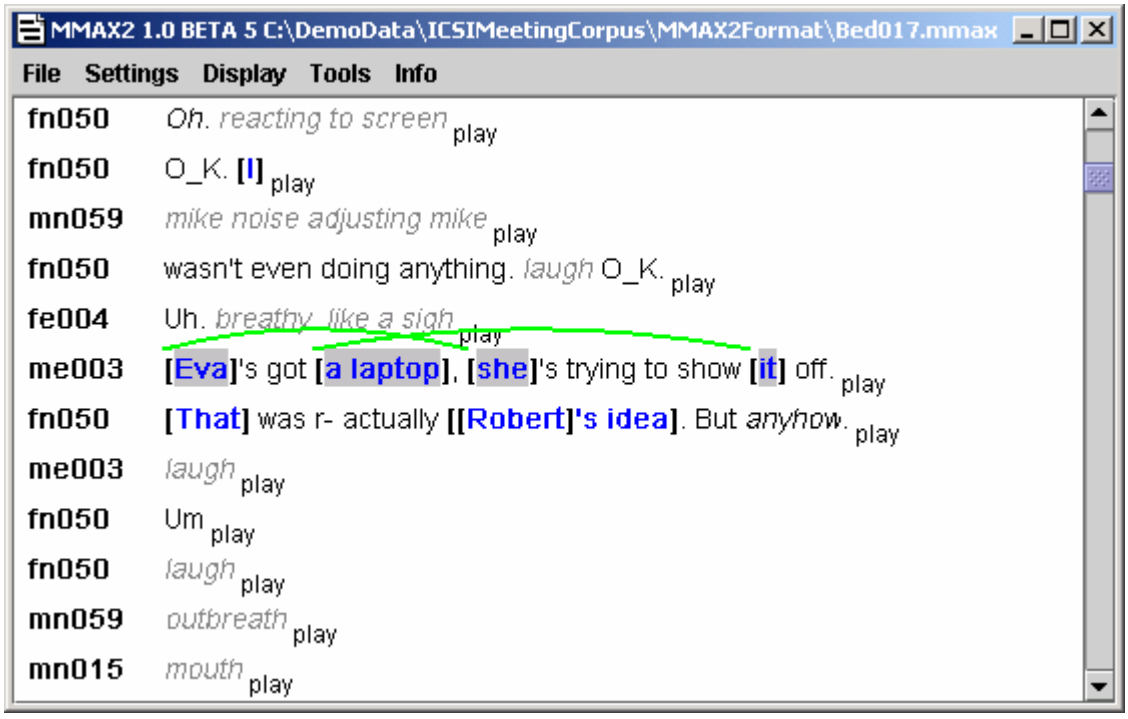

Figure 1: *The MMAX2 annotation tool main window*

The main purpose of this paper is to describe how linguistic annotation can be performed with MMAX2. The description is organized along the lines of a normal annotation life cycle in Computational Linguistics.

2 The annotation life cycle

There are several crucial steps in the life cycle of an annotation. They include the preparation of the machine-readable corpus, the definition and formalization of the annotation task, the manual annotation proper, the checking of the feasibility of the annotation, and the actual utilization of the completed annotation. In this section, we describe the capabilities of MMAX2 in each of these steps.

2.1 Preparing the base data

The first – and very important – step in the creation of an annotated corpus is the preparation of the data to be annotated. The data model that is used for

representing the annotated corpus should be designed in such a way that the usability and re-usability of the resulting resource is maximized. The data format should be as simple and theory-neutral as possible. In particular, it should not introduce arbitrary decisions or implicit assumptions which may hold for one application of the annotated corpus but which may be problematic for another. The data representation format should be implemented using a common standardized data storage format, preferably XML. This way, it is open to the whole range of XML processing tools available, and less dependent on the particular software that was used to create it. The corpus should be extensible in the sense that new annotations can be added to it without interfering with already existing ones. This implies that annotations should not alter the underlying corpus in any way. Ideally, they should be physically separated from it, and only *reference* it. This is known as the principle of *stand-off annotation* (Ide and Priest-Dorman 1996, Thompson and McKelvie 1997). Stand-off annotation, which is now a quasi-standard for representing annotated corpora, allows the corpus to contain annotations for several entirely different phenomena simultaneously. While these different levels of annotation are physically separated, it should still be possible to relate phenomena on different levels to each other. This is known as the principle of *multi-level annotation*. Apart from other advantages to be outlined below, the physical separation of annotation levels allows annotation tasks to be distributed to several research groups with different expertise. After completion of the individual annotation tasks, the annotations can be combined into one *multi-level* annotation that a single group could not have produced.

Within the MMAX2 data model, the text comprising the corpus to be annotated is called *base data*. Each element of the base data is modeled as a `<word>` XML element with an obligatory `id` attribute. The following is a base data file fragment from dialog Bed017.

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE words SYSTEM "words.dtd">
<words>
...
<word id="word_360">Eva</word>
<word id="word_361">'s</word>
<word id="word_362">got</word>
<word id="word_363">a</word>
<word id="word_364">laptop</word>
<word id="word_365">,</word>
<word id="word_366">she</word>
<word id="word_367">'s</word>
<word id="word_368">trying</word>
<word id="word_369">to</word>
<word id="word_370">show</word>
<word id="word_371">it</word>
<word id="word_372">off</word>
```

```
<word id="word_373">.</word>
...
</words>
```

The main function of these <word> elements is to serve as containers for the base data strings, and to associate a unique identifier with each of them. The granularity of the base data elements defines the smallest element that an annotation can be added to. In most cases, going down to the level of orthographical word should be sufficient, but it is possible to use units like morphemes, syllables, or even characters instead.

Elements of the base data are immutable, i.e. the base data file cannot be modified by the annotation. Annotations are not added to the base data directly, but in a stand-off manner by means of pointers to (sequences of) base data elements. In the MMAX2 data model, the actual annotations are contained in so-called *markables*. Markables are modeled as <markable> XML elements with minimally an id and a span attribute. The value of the span attribute is a list of the identifiers of the base data elements that the markable points to. All markables representing the same type of information are grouped into so-called *markable levels*, each of which is stored in a separate file.

The most simple type of annotation that a markable can represent is in the form of freetext attributes. The following is a fragment of the file containing the markable level 'segment'. The markable shown represents the information that the sequence from *word_360* to *word_373* is a connected utterance by the speaker with id *me003*.

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE markables SYSTEM "markables.dtd">
<markables xmlns="www.eml-research.de/ns/segment">
...
<markable id="markable_46" span="word_360..word_373" participant="me003"/>
...
</markables>
```

The name of the markable level is encoded in each markable file's XML name space declaration.

2.2 Defining an annotation scheme

The definition of an annotation (or coding) scheme is the next step in any non-trivial annotation project. The annotation scheme defines how the descriptive means of the data model should be used to represent the annotations. It is often complemented by a written annotation manual which serves as a guide for the annotators. The annotation manual normally contains a general description of the phenomena to be annotated, along with examples and detailed instructions.

In principle, the annotation manual should be as explicit as possible, ideally leaving no room for interpretations regarding how certain rules should be applied. If the annotation manual is too imprecise, this may lead to low inter-annotator agreement (see Section 2.4).

The MMAX2 data model is based on the assumption that any type of annotation can be represented by associating individual markables with attributes, and by linking several markables by means of relations. The attributes and relations that should be available for the annotation of a particular markable level are defined in an annotation scheme XML file.

Apart from the simple freetext attributes (see above), which can take an arbitrary string as their value, the MMAX2 data model also supports nominal attributes. Nominal attributes can only take one of a pre-defined set of possible values. The following annotation scheme fragment defines the attribute `tag` as a nominal attribute displayed as a drop-down list in the MMAX2 GUI. The set of defined values follows the Penn Treebank tag set (cf. Manning and Schütze 1999: 141f.).

```
<?xml version="1.0" encoding="US-ASCII"?>
<annotationscheme>
<attribute id="tag_level" name="tag" type="nominal_list"
 description="tag_level.html">
<value name="none" description="default_tag.html"/>
<value name="cc" description="cc_tag.html"/>
<value name="cd" description="cd_tag.html"/>
<value name="dt" description="dt_tag.html"/>
<value name="ex" description="ex_tag.html"/>
<value name="fw" description="fw_tag.html"/>
<value name="in" description="in_tag.html"/>
<value name="jj" description="jj_tag.html"/>
<value name="jjr" description="jjr_tag.html"/>
<value name="jjs" description="jjs_tag.html"/>
<value name="ls" description="ls_tag.html"/>
<value name="md" description="md_tag.html"/>
<value name="nn" description="nn_tag.html"/>
<value name="nns" description="nns_tag.html"/>
<value name="nnp" description="nnp_tag.html"/>
<value name="nnps" description="nnps_tag.html"/>
<value name="pdt" description="pdt_tag.html"/>
<value name="pos" description="pos_tag.html"/>
<value name="prp" description="prp_tag.html"/>
...
</attribute>
...
</annotationscheme>
```

The (optional) `description` attributes in the example contain references to HTML files with information about the respective attribute or value. During

annotation, the annotators can view this information within the MMAX2 GUI without having to look up definitions in a printed annotation manual.

The following is a fragment of the markable level 'POS' which utilizes the above definition to associate a POS tag with each word.

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE markables SYSTEM "markables.dtd">
<markables xmlns="www.eml-research.de/ns/pos">
...
<markable id="markable_219" span="word_360" tag="nnp"/>
<markable id="markable_220" span="word_361" tag="vbz"/>
<markable id="markable_221" span="word_362" tag="vbd"/>
<markable id="markable_222" span="word_363" tag="dt"/>
<markable id="markable_223" span="word_364" tag="nn"/>
<markable id="markable_224" span="word_365" tag="inp"/>
<markable id="markable_225" span="word_366" tag="prp"/>
<markable id="markable_226" span="word_367" tag="vbz"/>
<markable id="markable_227" span="word_368" tag="vbg"/>
<markable id="markable_228" span="word_369" tag="to"/>
<markable id="markable_229" span="word_370" tag="vb"/>
<markable id="markable_230" span="word_371" tag="prp"/>
<markable id="markable_231" span="word_372" tag="rp"/>
<markable id="markable_232" span="word_373" tag="inp"/>
...
</markables>
```

Although there is only one attribute associated with each markable in the above example, markables can in principle have many more attributes.

While attributes are capable of adding descriptive information to individual markables, relations are used to model structural or associative relations between two or more markables. Currently, the MMAX2 data model supports relations of the types *markable-set* and *markable-pointer*. A relation of the former type can express undirected relations between any number of markables. It can be interpreted as set-membership, i.e. markables having the same value in a relation of the type *markable-set* constitute an unordered set. A relation of the type *markable-pointer*, on the other hand, can express a directed relation between one source and one or more target markables. As the name suggests, this relation can be interpreted as the source markable pointing to its target markable(s). Both types of relations are defined in formal terms only, so that they can be associated with any kind of semantic interpretation as required. A common way to use a markable-set relation is for the annotation of coreference, i.e. for representing the information that two or more markables refer to the same entity.

The MMAX2 annotation scheme is also capable of expressing *dependencies* between attributes or relations. The usual type of dependency is that the

availability of one or more attributes or relations depends on the value of some other attribute.

Consider the example of the role of so-called pleonastic *it* in coreference annotation. Pleonastic (or non-referential) *it* appears in constructions like 'It is raining'. While it makes sense to annotate these cases on the coreference level as special cases of pronoun use, they should not be allowed to participate in any coreference relations, because this attribute does not apply to them. This type of constraint can be expressed in the annotation scheme file by means of a special attribute. The following annotation scheme fragment for the markable level 'coreference' defines the attribute `exp_type` as a nominal attribute displayed in the form of a row of buttons. The attribute `coref_class` is defined as a dependent markable-set relation, which is rendered by means of curved green lines. The dependency is expressed by means of the `next` attribute, which causes the `coref_class` attribute to be available only if the current value of the `exp_type` attribute is *`noun_phrase`*.

```
<?xml version="1.0" encoding="US-ASCII"?>
<annotationscheme>
<attribute id="exp_type_attribute" name="exp_type"
type="nominal_button"
 description="exp_type_level.html">
<value name="none" description="default_tag.html"/>
<value name="noun_phrase" description="np_type.html"
 next="coref_class_attribute"/>
<value name="pleonastic_it" description="pleoit_type.html"/>
...
</attribute>

<attribute id="coref_class_attribute" name="coref_class"
type="markable_set"
 style="lcurve" color="green">
<value name="coref_class"/>
</attribute>
...
</annotationscheme>
```

Constraining the availability of attributes and relations to only those that make sense in a given situation is a means to ensure the quality and consistency of the annotation. At the same time, it also provides some guidance for the annotator.

The following is a fragment from the markable level 'coreference', containing the information that the noun phrases 'Eva' and 'she' and 'a laptop' and 'it' respectively are coreferent.

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE markables SYSTEM "markables.dtd">
<markables xmlns="www.eml-research.de/ns/coreference">
...
<markable id="markable_953" span="word_360" exp_type="noun_phrase"
```

```
 coref_class="set_3"/>
<markable id="markable_954" span="word_363..word_364"
exp_type="noun_phrase"
 coref_class="set_4"/>
<markable id="markable_955" span="word_366" exp_type="noun_phrase"
 coref_class="set_3"/>
<markable id="markable_956" span="word_371" exp_type="noun_phrase"
 coref_class="set_4"/>
...
</markables>
```

Note that the `exp_type` attribute can be rather general, only differentiating between *`none`*, *`noun_phrase`*, and *`pleonastic_it`*, because more fine-grained information is already available on the markable level 'POS'.

2.3 Creating an annotation

Performing the actual annotation is demanding, both intellectually and physically. Both aspects can be directly translated into requirements for the graphical user interface (GUI) of the employed annotation tool.

The GUI should be customizable in such a way that all and only the relevant information is displayed to the annotator. What counts as relevant can be dependent on the annotation task itself, but also on subjective factors like the level of experience of the annotators. Irrelevant information can easily distract them, while too little information can render the task unnecessarily hard, especially for less experienced annotators.

Apart from the intellectual demands, corpus annotation is essentially manual labor. Using mainly the mouse, the annotator interacts with the display, which serves as the interface to the data. Typical forms of user actions include

- selecting portions of text by clicking a mouse button and dragging the mouse over a part of the display,
- clicking parts of the display in order to perform some action on the items displayed there, and
- selecting from drop-down lists or rows of buttons the values for certain attributes.

Given the fact that for a corpus of realistic size and an annotation scheme of realistic complexity quite a few of these actions have to be carried out, the performance of the display is extremely important. The main measure of display performance is the time that it takes for the display to respond to a user action. If this time is typically above a certain threshold, it can render a tool practically unusable. This is because even short delays add up in the long run, and, more importantly, every perceivable delay annoys the annotator.

The MMAX2 annotation tool tries to maximize both the customizability and the performance of the display by employing a rather elaborate mechanism. It is based on the distinction between display *content* (i.e. which text is displayed to the annotator) and display *style* (i.e. how this text is displayed).

The display content is specified by means of a user-modifiable XSL style sheet. Simple style sheets might only output the base data without any additional structure. Slightly more refined style sheets can add line breaks at markable boundaries, e.g. for markables representing structural units like segments. Markable boundaries can also be visualized by means of so-called *markable handles*, i.e. brackets that are inserted in the text directly before and after a markable. This is particularly useful if several embedded or overlapping markables exist on either the same or different markable levels. Finally, the values of nominal markable attributes can also be shown in the display. It is possible, e.g., to write a style sheet that directly displays the POS tag for each word in the display. Each MMAX2 document can have any number of style sheets, each defining a different view on the data. Changes to the display content require that the style sheet processor is reinvoked. Depending on the size of the document and the complexity of the style sheet, this can take some time. However, this does not pose a problem since the display *content* does not require frequent updates.

This is in contrast to the display *style* which is controlled by markables and their attributes. For each level, users can customize how markables should be visualized depending on their attributes and relations. Display parameters include foreground and background color, size, and font attributes like bold, italic, underlined, etc. User actions like adding or deleting a markable, adding a relation between two markables, or modifying the attributes of a markable require frequent display updates. MMAX2 contains optimized methods for this type of display modification, which allows for short response times for most types of user actions.

One markable at a time can be selected by left-clicking it with the mouse. If the markable is part of one or more markable relations, these are visualized by means of lines that are drawn onto the display, connecting the selected markable to all markables it is related to. The attributes of the currently selected markable are displayed and can be modified in a special attribute window. The attribute window renders freetext attributes as editable text boxes, and nominal attributes as either lists of entries or rows of buttons. The attribute window enforces dependencies between attributes by only displaying those attributes and possible values that are defined in the annotation scheme for the current attribute constellation. If the annotator modifies the value of an attribute, the attribute window is immediately updated to reflect any changes resulting from that modification. By default, modifications in the attribute window are not immediately applied to the current markable, but only after the annotator clicked

a special 'Apply' button. An 'Auto-Apply' mode is also available in which each modification is immediately applied. For more simple types of annotations (or more experienced annotators), this can reduce the number of required mouse clicks considerably.

2.4 Checking inter-annotator agreement

Another very important step in the life cycle of an annotation is to check the inter-annotator agreement. Depending on the difficulty of the task, annotation can involve non-trivial interpretation on the part of the annotator, so that different annotators can hardly ever be expected to produce completely identical annotations. Inter-annotator agreement is a quantitative index for the degree of agreement between the annotations produced by two or more annotators on the same base data, using the same annotation manual. It is commonly calculated using the Kappa statistic (Carletta 1996). The Kappa statistic is special in that it takes into account that a certain degree of agreement between the annotations produced by two or more annotators can also be ascribed to chance.

If an annotation yields a Kappa value below some critical threshold, and if it can be ruled out that the lack of agreement between annotators is due to ambiguities or impreciseness in the annotation manual, this often indicates that the phenomenon to be annotated is inherently ill-defined, ambiguous or vague. Then, it may be impossible for the annotators to independently agree upon the correct solution because no such solution exists. If this is the case, the annotated data has only limited usability. This is especially true if the annotations are supposed to be used as the data basis for the development of an NLP component: If no reasonably objective correct solution does exist for a given task, there is no point in developing a software component for trying to find it.

Within MMAX2, the stand-off data model makes it easy to maintain several annotations of the same base data simultaneously for comparison. MMAX2 also contains a special tool for the quantitative and qualitative comparison of annotations. Means of qualitative comparison for nominal attributes include simple list-based visualization and highlighting of differences. If only two annotations are compared, differences can also be displayed in the form of a confusion matrix. The tool can also calculate the Kappa statistic for nominal attributes. Markable relations can be compared qualitatively by simultaneously visualizing the markable sets created by different annotators. For quantitative comparison of markable sets, the inter-annotator agreement can also be determined by calculating a statistical measure that takes the special properties of sets into account (Vilain et al. 1995).

2.5 Using the annotation

An annotated corpus is not an end in itself, but rather a resource to be used for various tasks. The methods to be used for a given task depend on the complexity of the latter. We distinguish three types of methods with different capabilities. Corpus querying deals with simply retrieving examples of certain phenomena from the corpus. Corpus transformation has to do with converting the corpus into other formats. Finally, corpus processing is the most powerful method, dealing with any type of automatic corpus manipulation beyond what can be accomplished with the first two methods.

2.5.1 Corpus querying

The most simple use an annotated corpus can be put to is exploratory data analysis or browsing. The annotation tool should support this by allowing the user to query the annotation, and to visualize query results. An important requirement of querying is that users should be allowed to relate diverse phenomena in a fairly unrestricted and ad-hoc way. In general, querying should be as simple as possible, thus facilitating exploratory data analysis also for non-expert users.

The MMAX2 annotation tool comprises the multi-level query language MMAXQL which allows the formulation of queries that can access and relate to each other markables from diverse levels. The focus of MMAXQL lies on simplicity and ease of use, rather than maximal expressive power. It allows queries of low and medium complexity to be formulated easily, but leaves more complicated things to be solved by other means (see below).

A query in MMAXQL consists of a sequence of query tokens which are combined by means of relation operators. Each query token queries exactly one base data element or one markable.

Base data elements can be queried by matching regular expressions. Each base data query token consists of a regular expression in single quotes, which must exactly match a base data element. Sequences of base data elements can be queried by simply concatenating several space-separated tokens. The query

```
'[Tt]he [A-Z].+'
```

will match sequences beginning with a definite article and a word with a capital first letter.

Markables can be queried by means of string matching and attribute-value combinations. A markable query token has the form

```
string/conditions
```

where `string` is an optional regular expression and `conditions` specifies which attribute(s) the markable should match. The most simple condition is just the name of a markable level, which will match all markables on that level. If a regular expression is also supplied, the query will return only the matching markables. The query

```
[Aa]n?\s.*/coreference
```

will return all markables from the level 'coreference' beginning with the indefinite article.

The `conditions` part of a markable query token can indeed be much more complex. A main feature of MMAXQL is that redundant parts of conditions can optionally be left out, making queries very concise. For example, the markable level name can be left out if the name of the attribute accessed by the query is unique across all markable levels in the MMAX2 document. Thus, the query

```
/!coref_class=empty
```

can be used to query markables from the level 'coreference' which have a non-empty value in the `coref_class` attribute, granted that only one attribute of this name exists on all markable levels. The same applies to the names of nominal attributes if the value specified in the query unambiguously points to this attribute. Thus, the query

```
/nnp
```

can be used to query markables from the level 'POS' which have the value *nnp* (proper name), granted that there is exactly one nominal attribute with the possible value *nnp*. Several markable levels can be accessed in one query by combining two or more query tokens with a relation operator specifying the relation that should hold between both. The results of these queries consist of markable tuples with one element for each query token. Thus, the following query

```
/{prp,prp$} = /!coref_class=empty
```

returns 2-place markable tuples where the first markable comes from the level 'POS' and the second from the level 'coreference'. The first markable has a `tag` attribute value of either *prp* or *prp$*, the second has a non-empty value in the `coref_class` attribute. The equals sign specifies that the span of the markables in each tuple should be identical. Thus, the query combines information from the levels 'POS' and 'coreference' to retrieve personal and possessive pronouns that

are part in coreference relations. Other structural relations besides identity are left and right alignment, embedding, or overlap.

2.5.2 Corpus transformation

Corpus transformation may be necessary if annotations have to be converted from their original source format into some different target format. This target format might be the data format required by another annotation tool or by some other program. By supporting conversions into other data formats, an annotation tool increases the usability of the annotations that are created with it.

MMAX2 supports the transformation of annotated corpora by means of XSL. The tool's style sheet engine provides some special functions for handling stand-off annotation as realized in MMAX2. These functions are integrated into the style sheet engine in such a way that they can be used like normal XSL functions.

The following example shows how a MMAX2 document can be transformed into an HTML document by means of an XSL style sheet. HTML is a popular target format for visualizing annotated corpora because it is very common and can be easily viewed both locally and over the Internet.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
                xmlns:mmax="org.eml.MMAX2.discourse.MMAX2DiscourseLoader"
                 xmlns:meta="www.eml-research.de/ns/meta"
                 xmlns:pos="www.eml-research.de/ns/pos"
                 xmlns:coreference="www.eml-research.de/ns/coreference"
                 xmlns:segment="www.eml-research.de/ns/segment">
 <xsl:output method="html" indent="no"/>

 <xsl:template match="words">
  <html><body style="font-family:Arial;">
   <table>
    <xsl:apply-templates/>
   </table>
  </body></html>
 </xsl:template>

 <xsl:template match="word">
  <xsl:apply-templates select="mmax:getStartedMarkables(@id)" mode="opening"/>
   <xsl:text> </xsl:text>
   <xsl:value-of select="text()"/>
   <xsl:apply-templates select="mmax:getEndedMarkables(@id)" mode="closing"/>
 </xsl:template>

 <xsl:template match="segment:markable" mode="opening">
  <xsl:text disable-output-escaping="yes">&lt;tr&gt;</xsl:text>
   <td><b>
     <xsl:value-of select="@participant"/>
    </b></td>
  <xsl:text disable-output-escaping="yes">&lt;td&gt;</xsl:text>
 </xsl:template>
```

```
 <xsl:template match="segment:markable" mode="closing">
  <xsl:text disable-output-escaping="yes">&lt;/td&gt;</xsl:text>
  <xsl:text disable-output-escaping="yes">&lt;/tr&gt;</xsl:text>
 </xsl:template>

 <xsl:template match="meta:markable" mode="opening">
  <xsl:text disable-output-escaping="yes">&lt;em&gt;</xsl:text>
 </xsl:template>

 <xsl:template match="meta:markable" mode="closing">
  <xsl:text disable-output-escaping="yes">&lt;/em&gt;</xsl:text>
 </xsl:template>

 <xsl:template match="coreference:markable" mode="opening">
  <b> [</b>
 </xsl:template>

 <xsl:template match="coreference:markable" mode="closing">
  <b>]</b>
 </xsl:template>

 <xsl:template match="pos:markable" mode="closing">
  <xsl:if test="mmax:inMarkableFromLevel(@id,'pos','coreference')">
   <sub><xsl:value-of select="@tag"/></sub>
  </xsl:if>
 </xsl:template>

</xsl:stylesheet>
```

If the above style sheet is stored in a file named *mmax2html.xsl*, the command

```
org.eml.MMAX2.Process -in Bed017.mmax -style mmax2html.xsl -out out.html
```

can be used to create an HTML file from dialog Bed017. The resulting HTML file (*out.html*) is rendered as in Figure 2 (overleaf).

2.5.3 Corpus processing

If the annotations are to be used as a resource for the development of NLP components, more complex operations than simple querying or transformation are normally required. This is also true for very complex or specialized corpus queries. Therefore, the annotation tool should also supply a flexible interface that allows unrestricted access to the annotation by means of a common programming language.

The MMAX2 Discourse API allows access to MMAX2 documents from the programming language Java. It takes advantage of the fact that MMAX2 itself is entirely written in Java. When using the annotation tool, users do not have to deal with the XML files directly. Instead, the tool parses the files comprising a MMAX2 document, and resolves the various relations between markables in the

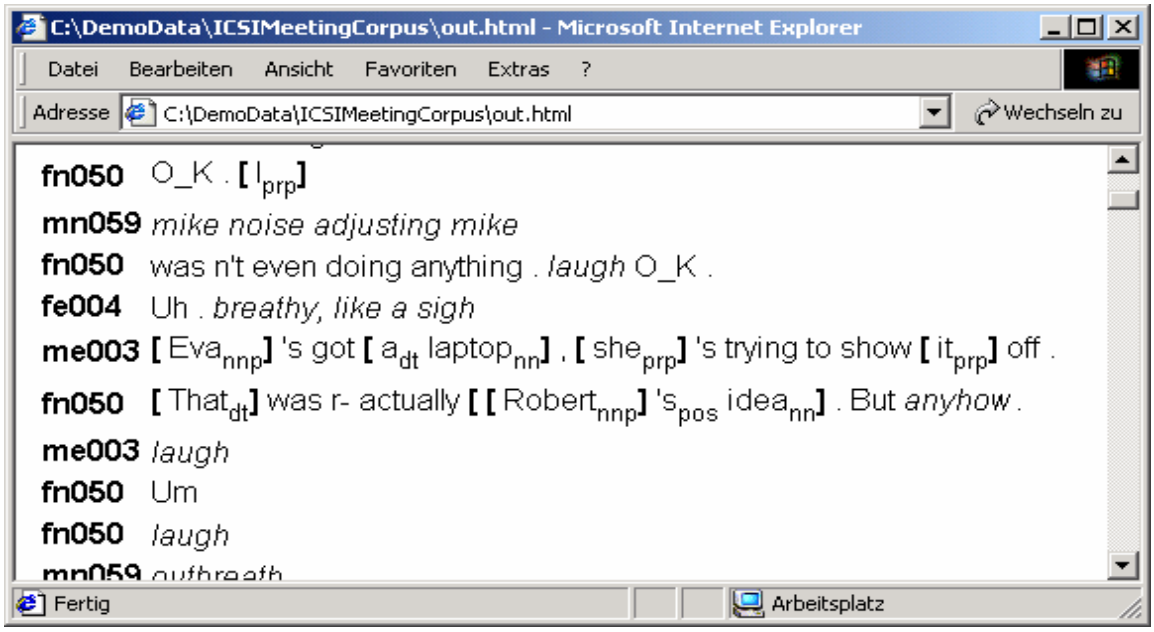

Figure 2: *Possible HTML rendering of a MMAX2 document, viewed in web browser*

same or different files. The MMAX2 Discourse API supplies the same convenience for dealing with MMAX2 documents outside the annotation tool. The Discourse API defines Java classes that serve as wrappers around elements of the annotation. The most important of these classes are `MMAX2Discourse`, `MarkableLevel`, `Markable`, and `MMAX2DiscourseElement`. The latter class is used for wrapping base data elements. The following code fragment demonstrates how an instance of the class `MMAX2Discourse` can be created from a MMAX2 document.

```
MMAX2Discourse discourse = MMAX2Discourse.buildDiscourse("Bed017.mmax");
```

The object `discourse` can then be used as the single entry point to the entire annotated corpus. The class `MMAX2Discourse` contains basic methods for retrieving all base data elements in the corpus. Access to markables is a bit more complex and consists of two steps. The first step is to identify and retrieve the required markable level. Markable levels are identified by their names, which must be unique within a MMAX2 document. The following code fragment retrieves the markable level 'coreference ' (assuming that the `discourse` object has been created as described above). The second parameter (`false`) controls the behavior of the API in case a markable level with the required name cannot be found.

```
MarkableLevel corefLevel =
          discourse.getMarkableLevelByName("coreference",false);
```

Access to individual markables on the markable level 'coreference ' is then provided by various methods. It is possible to retrieve all markables on a level, or only those that span a particular base data element. The latter method is used in the following example to retrieve all coreference markables whose span contains the base data element *word_360* (i.e. the token 'Eva'). The second parameter specifies that the returned list of markables should be sorted in discourse order.

```
ArrayList mList = corefLevel.getMarkablesAtDiscourseElementID("word_360",
                  new DiscourseOrderMarkableComparator());
```

The above code fragment will return a list containing exactly one markable. Nominal attributes can be accessed by means of methods defined in the `Markable` class. Assuming that `corefMarkable` is an instance of class `Markable`, the following code fragment can be used to retrieve the markable's `exp_type` value. The second parameter (`"none"`) specifies the value to return if the required attribute is undefined for the markable.

```
String exp_typeValue = corefMarkable.getAttributeValue("exp_type","none");
```

The MMAX2 API also defines wrappers for elements related to the annotation scheme, including `MMAX2AnnotationScheme`, `MMAX2Attribute`, `MarkableRelation`, and `MarkableSet`. They are required when accessing markables based on their relations. The following code fragment demonstrates how the markable set defined by the `coref_class` attribute can be retrieved for a given markable, `corefMarkable`, and how the list of all markables (including `corefMarkable`) can be accessed.

```
MMAX2AnnotationScheme annoScheme = corefLevel.getCurrentAnnotationScheme();
MMAX2Attribute corefAttribute =
                          annoScheme.getMMAX2AttributeByName("coref_class");
MarkableRelation corefRelation = corefAttribute.getMarkableRelation();
MarkableSet corefSet =
       corefRelation.getMarkableSetContainingMarkable(corefMarkable);
ArrayList corefMarkables = corefSet.getMarkables();
```

3 Conclusion

This paper described how the annotation tool MMAX2 can be used to create richly annotated, multi-level corpora. The description focused on the creation of corpora in Computational Linguistics. The tool has already been applied for the creation of several annotations which are used as resources for the development of NLP components. Annotated phenomena include POS tags, word senses, coreference, disfluencies (in transcribed spoken language), grammatical

dependency relations, and others. In the near future, it is also planned to employ the tool for the creation of a resource for automatic summarization. While all these applications are mostly NLP-related, it should have become clear that both the MMAX2 data model and the tool itself are sufficiently flexible to be used for other types of applications as well.

List of References

Carletta, J. (1996): "Assessing agreement on classification tasks: The kappa statistic", *Computational Linguistics* 22(2), 249-254.

Ide, N. & G. Priest-Dorman (1996): "The corpus encoding standard". Available at <http://www.cs.vassar.edu/CES>.

Janin, A., D. Baron, J. Edwards, D. Ellis, D. Gelbart, N. Morgan, B. Peskin, T. Pfau, E. Shriberg, A. Stolcke & C. Wooters (2003): "The ICSI Meeting Corpus", *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Hong Kong, April 2003. 364-367.

Jurafsky, D. & J.H. Martin (2000): *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Upper Saddle River, NJ: Prentice Hall.

Manning, C.D. & H. Schütze (1999): *Foundations of Statistical Natural Language Processing.* Cambridge, MA: MIT Press.

Mani, I. (2001): *Automatic Summarization.* Amsterdam: John Benjamins.

Maybury, M.T. (ed.) (2004): *New Directions in Question Answering.* Cambridge, MA: MIT Press.

Mitkov, R. (2002): *Anaphora Resolution*. London: Longman.

Nirenburg, S., H. Somers & Y. Wilcks (eds.) (2003): *Readings in Machine Translation.* Cambridge, MA: MIT Press.

Thompson, H. S. & D. McKelvie (1997): "Hyperlink semantics for standoff markup of read-only documents", *Proceedings of SGML Europe '97*, Barcelona, Spain, May 1997.

Vilain, M., J. Burger, J. Aberdeen, D. Connolly & L. Hirschman (1995): "A model-theoretic coreference scoring scheme", *Proceedings of the 6th Message Understanding Conference (MUC-6)*, San Mateo, CA: Morgan Kaufmann. 45-52.