Representing and Accessing Multi-Level Annotations in MMAX2

Christoph Müller
EML Research gGmbH
Villa Bosch
Schloß-Wolfsbrunnenweg 33
69118 Heidelberg, Germany
christoph.mueller@eml-research.de

1 Introduction

MMAX2[1] is a versatile, XML-based annotation tool which has already been used in a variety of annotation projects. It is also the tool of choice in the ongoing project DIANA-Summ, which deals with anaphora resolution and its application to spoken dialog summarization. The project uses the ICSI Meeting Corpus (Janin et al., 2003), a corpus of multi-party dialogs which contains a considerable amount of simultaneous speech. It features a semi-automatically generated segmentation in which the corpus developers tried to track the flow of the dialog by inserting segment starts approximately whenever a person started talking. As a result, the corpus has some interesting structural properties, most notably overlap, that are challenging for an XML-based representation format. The following brief overview of MMAX2 focuses on this aspect, using examples from the ICSI Meeting Corpus.

2 The MMAX2 Data Model

Like most current annotation tools, MMAX2 makes use of *stand-off* annotation. The base data is represented as a sequence of `<word>` elements with exactly one PCDATA child each. Normally, these PCDATA children represent orthographical words, but larger or smaller units (e.g. characters) are also possible, depending on the required granularity of the representation.[2]

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE words SYSTEM "words.dtd">
 <words>
  ...
  <word id="word_3710">That</word>
  <word id="word_3711">'s</word>
  <word id="word_3712">just</word>
  <word id="word_3713">a</word>
  <word id="word_3714" meta="true">Pause</word>
  <word id="word_3715">specification</word>
  <word id="word_3716">for</word>
  <word id="word_3717">the</word>
  <word id="word_3718">X_M_L</word>
  <word id="word_3719">Yep</word>
  <word id="word_3720">.</word>
  <word id="word_3721">format</word>
  <word id="word_3722">.</word>
  ...
 </words>
```

The order of the elements in the base data file is determined by the order of the segments that they belong to.

Annotations are represented in the form of `<markable>` elements which reference one or more base data elements by means of a `span` attribute. Each markable is associated with exactly one *markable level* which has a unique, descriptive name and which groups markables that belong to the same category or *annotation dimension*. Each markable level is stored in a separate XML file which bears the level name as an XML name space. The ID of a markable must be unique for the level that it belongs to. Markables can carry arbitrarily many features in the common `attribute=value` format. It is by means of these features that the actual annotation information is represented. For each markable level, admissible attributes and possible values are defined in an *annotation scheme* XML file (not shown). These annotation schemes are much more powerful for expressing attribute-related constraints than e.g. DTDs. The following first example shows the result of the segmentation of the sample base data. The `participant` attribute contains the identifier of the speaker that is associated with the respective segment.

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE markables SYSTEM "markables.dtd">
 <markables xmlns="www.eml.org/NameSpaces/segment">
  ...
  <markable id="markable_468"
            span="word_3710..word_3714"
            participant="me012"/>
  <markable id="markable_469"
            span="word_3715..word_3718"
            participant="me012"/>
  <markable id="markable_470"
            span="word_3719..word_3720"
            participant="mn015"/>
  <markable id="markable_471"
            span="word_3721..word_3722"
            participant="me012"/>
  ...
 </markables>
```

[1] http://mmax.eml-research.de

[2] The `meta` attribute in the example serves to distinguish meta information from actual spoken words.

The next example contains markables representing the nominal and verbal chunks in the sample base data.

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE markables SYSTEM "markables.dtd">
 <markables xmlns="www.eml.org/NameSpaces/chunks">
  ...
  <markable id="markable_7834"
            span="word_3710"
            type="demonstrative"/>
  <markable id="markable_7835"
            span="word_3711"
            type="copula"/>
  <markable id="markable_7836"
            span="word_3713..word_3715"
            type="nn"/>
  <markable id="markable_7837"
            span="word_3711..word_3715"
            type="predication"
            subject="markable_7834"/>
  <markable id="markable_7838"
            span="word_3717..word_3718,word_3721"
            type="nn"/>
  ...
 </markables>
```

The following basic facts about markables in MMAX2 are worth noting:

1. Every markable is defined with reference to the base data. Markables on the same or different levels are independent and ignorant of each other, and only related *indirectly*, i.e. by means of base data elements that they have in common.[3] Structural relations like embedding (*[['s] just [a specification]]*) can only be determined with recourse to the base data elements that each markable spans. This **lazy** representation makes it simple and straightforward to add markables and entire markable levels to existing annotations. It is also a natural way to represent non-hierarchical relations like **overlap** between markables. For example, a segment break runs through the nominal chunk represented by markable `markable_7836` (*[a specification]*) in the example above. If the segment markables were defined in terms of the markables contained in them, this would be a problem because the nominal chunk crosses a segment boundary. The downside of this lazy representation is that more processing is required for e.g. querying, when the structural relations between markables have to be determined.

2. Markables can be *discontinuous*. A markable normally spans a sequence of base data elements. Each *connected* subsequence of these is called a markable *fragment*. A discontinuous markable is one that contains more than one fragment, like markable `markable_7838` (*[the XML format]*) above. Actually, this markable exemplifies what could be called **discontinuous overlap** because it does not only cross a segment boundary, but it also has to omit elements from an intervening segment by another speaker.

[3]Note that this merely means that markables are not *defined* in terms of other markables, while they can indeed *reference* each other: In the above example, markable `markable_7837` (*['s just a specification]*) uses an associative relation (in this case named `subject`) to represent a reference to markable `markable_7834` (*[That]*) on the same level. References to markables on other levels can be represented by prefixing the markable ID with the level name.

3 Accessing Data From Within MMAX2

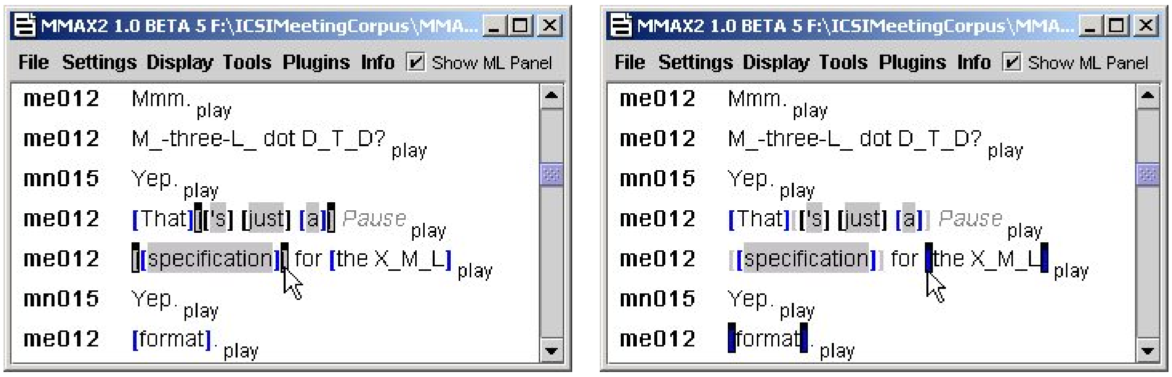
3.1 Visualization

When a MMAX2 document is currently loaded, the main display contains the base data text plus annotation-related information. This information can comprise

- line breaks (e.g. one after each segment),
- markable feature's values (e.g. the `participant` value at the beginning of each segment),
- literal text (e.g. a tab character after the `participant` value),
- markable customizations, and
- markable handles.

The so-called *markable customizations* are in charge of displaying text in different colors, fonts, font styles, or font sizes depending on a markable's features. The order in which they are applied to the text is determined by the order of the currently available markable levels. Markable customizations are processed bottom-up, so markable levels should be ordered in such a way that levels containing *smaller* elements (e.g. POS tags) should be on top of those levels containing *larger* elements (chunks, segments etc.). This way, smaller elements will not be hidden by larger ones.

When it comes to visualizing several, potentially embedded or overlapping markables, the so-called *markable handles* are of particular importance. In their most simple form, markable handles are pairs of short strings (most often pairs of brackets) that are displayed directly before and after each fragment of a markable. When two or more markables from different levels start at the same base data element, the *nesting order* of the markables (and their handles) is determined on the basis of the order of the currently available markable levels. The color of markable handles can also be customized depending on a markable's features. Figure 1 gives an idea of what the


MMAX2 1.0 BETA 5 F:\ICSIMeetingCorpus\MMA...
File Settings Display Tools Plugins Info
Show ML Panel

me012 Mmm. play
me012 M_-three-L_ dot D_T_D? play
mn015 Yep. play
me012 [That] ['s] [just] [a] Pause play
me012 [[specification]] for [the X_M_L] play
mn015 Yep. play
me012 [format]. play

me012 Mmm. play
me012 M_-three-L_ dot D_T_D? play
mn015 Yep. play
me012 [That] ['s] [just] [a] Pause play
me012 [[specification]] for [the X_M_L] play
mn015 Yep. play
me012 [format]. play

Figure 1: Highlighted markable handles on a discontinuous (left) and an overlapping (right) markable.

MMAX2 main window can look like. Both handles and text background for `chunk` markables with `type=predication` are rendered in light gray. Other handles are rendered in a darker color. Markable handles are sensitive to mouse events: resting the mouse pointer over a markable handle will highlight all handles of the pertaining markable. Reasonable use of markable customizations and handles allows for convenient visualization of even rather complex annotations.

3.2 Querying

MMAX2 includes a query console which can be used to formulate simple queries using a special multi-level query language called *MMAXQL*. A query in MMAXQL consists of a sequence of *query tokens* which describe elements (i.e. either base data elements or markables) to be matched, and *relation operators* which specify which relation should hold between the elements matched by two adjacent query tokens. A single markable query token has the form

`string/conditions`

where `string` is an optional regular expression and `conditions` specifies which features(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.+/chunks`

will return all markables from the `chunks` level that begin with the indefinite article[4]. Markables with particular features can be queried by specifying the desired attribute-value combinations. The following query e.g. will return all markables from the `chunks` level with a `type` value of either `nn` or `demonstrative`:

`/chunks.type={nn,demonstrative}`

If a particular value is defined for *exactly* one attribute on *exactly* one markable level only, both the level and the attribute name can be left out in a query, rendering queries very concise (cf. the access to the `meta` level below).

Relation operators can be used to connect two query tokens to form a complex query. The set of supported sequential and hierarchical relation operators[5] includes `meets` (default), `starts`, `starts_with`, `in`, `dom`, `equals`, `ends`, `ends_with`, `overlaps_right`, and `overlaps_left`. Whether two markables stand in a certain relation is determined with respect to the base data elements that they span. In the current early implementation, for all markables (including discontinuous ones), only the first and last base data element is considered. The result of a query can directly be used as the input to another query. The following example gives an idea of what a more complex query can look like. The query combines the `segment` level, the `meta` level (which contains markables representing e.g. pauses, emphases, or sounds like breathing or mike noise), and the base data level to retrieve those instances of *you know* from the ICSI Meeting corpus that occur in segments spoken by female speakers[6] which also contain a pause or an emphasis (represented on the `meta` level):

```
'[Yy]ou know' in (/participant={f.*} dom /{pause,emphasis})
```

The next query shows how overlap can be han-

[4]The space character in the regular expression must be masked as `\s` because otherwise it will be interpreted as a query token separator.

[5]Associative relations are not discussed here, (Müller, 2005).

[6]The first letter of the `participant` value encodes the speaker's gender.

dled. It retrieves all chunk markables along with their pertaining segments by getting two partial lists and merging them using the operator or.

```
(/chunks in /segment) or (/chunks overlaps_right /segment)
```

4 Accessing Data by Means of XSL

MMAX2 has a built-in XSL style sheet processor[7] that can be used from the console to read a MMAX2 document and process it with a user-defined XSL style sheet. The XSL processor provides some special extensions for handling stand-off annotation as it is realized in MMAX2. In the current *beta* implementation, only some basic extensions exist. The style sheet processor can be called from the console like this:

```
org.eml.MMAX2.Process -in INFILE.mmax -style STYLEFILE.xsl
```

The root element of each MMAX2 document is the words element, i.e. the root of the base data file, which will be matched by the supplied XSL style sheet's default template. The actual processing starts in the XSL template for the word elements, i.e. the root element's children. A minimal template looks like this:

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

The above template inserts a white space before the current word and then calls an extension function that returns a NodeList containing all markables starting at the word. The template then inserts the word's text and calls another extension function that returns a NodeList of markables ending at the word. The markables returned by the two extension function calls are themselves matched by XSL templates. A minimal template pair for matching starting and ending markables from the chunks level and enclosing them in bold brackets (using HTML) looks like this:

```
<xsl:template match="chunks:markable" mode="opening">
 <b>[</b>
</xsl:template>
<xsl:template match="chunks:markable" mode="closing">
 <b>]</b>
</xsl:template>
```

Note how the markable level name (here: chunks) is used as a markable name space to control which markables the above templates should match. The following templates wrap a pair of <p> tags around each markable from the segment level and adds the value of the participant attribute to the beginning of each.

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

Creating HTML in this way can be useful for converting a MMAX2 document with multiple levels of annotation to a *lean* version for distribution and (online) viewing.

5 Conclusion

MMAX2 is a practically usable tool for multi-level annotation. Its main field of application is the manual creation of annotated corpora, which is supported by flexible and powerful means of visualizing both simple and complex (incl. overlapping) annotations. MMAX2 also features a simple query language and a way of accessing annotated corpora by means of XSL style sheets. While these two data access methods are somewhat limited in scope, they are still useful in practice. If a query or processing task is beyond the scope of what MMAX2 can do, its simple and open XML data format allows for easy conversion into other XML-based formats, incl. those of other annotation and query tools.

Acknowledgements

This work has partly been funded by the Klaus Tschira Foundation (KTF), Heidelberg, Germany, and by the Deutsche Forschungsgemeinschaft (DFG) in the context of the project DIANA-Summ (STR 545/2-1).

References

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. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Hong Kong, pp. 364–367.

Müller, C. (2005). A flexible stand-off data model with query language for multi-level annotation. In *Proceedings of the Interactive Posters/Demonstrations session at the 43rd Annual Meeting of the Association for Computational Linguistics*, Ann Arbor, Mi., 25-30 June 2005, pp. pp. 109–112.

[7]Based on Apache's Xalan