Multi-Level Annotation in MMAX

Christoph Müller and **Michael Strube**
European Media Laboratory GmbH
Villa Bosch
Schloß-Wolfsbrunnenweg 33
69118 Heidelberg, Germany
{Christoph.Mueller, Michael.Strube}@eml.villa-bosch.de

Abstract

We present a light-weight tool for the annotation of linguistic data on multiple levels. It is based on the simplification of annotations to sets of *markables* having *attributes* and standing in certain *relations* to each other. We describe the main features of the tool, emphasizing its simplicity, customizability and versatility.

1 Introduction

In recent years, the development and use of annotation tools has been a recurrent topic in corpus-based computational linguistics. Currently, specialized tools for the annotation of a wide range of phenomena on different levels of linguistic description are available. In the more recent of these tools, principles of design and implementation are realized which over the years have emerged as quasi-standards:

- XML as data storage format,
- file-level separation of base data (i.e. the data to be annotated) from the annotation, use of *stand-off annotation* (Ide and Priest-Dorman, 1996),
- implementation in Java for the sake of platform independence.

Most of the available tools handle well the phenomena on the linguistic level they are intended for, be it coreference, dialogue acts, or discourse structure, to name just a few. The annotations they yield do exist independently of each other and cannot easily be combined or applied to the same language data. This, however, would be highly desirable because it would allow for simultaneous browsing and annotating on several linguistic levels. In addition, annotation tasks could be distributed to several research groups with different expertise, with one group specializing in e.g. dialogue act tagging, another in coreference annotation, and so on. After completion of the individual annotation tasks, the annotations could be combined into one *multi-level* annotation that a single group could not have produced.

The MMAX[1] tool presented in this paper is intended as a light-weight and highly customizable implementation of multi-level annotation of (potentially multi-modal) corpora. It is based on the assumption that any annotation can be simplified to sets of so-called *markables* carrying *attributes* and standing in certain *relations* to each other. Consequently, all a tool has to supply is efficient low-level support for the creation and maintenance of markables on different levels.

The remainder of this paper is structured as follows: In Section 2 we describe in more detail the basic concepts underlying our approach. Section 3 describes how *annotation* (or *coding*) *schemes* can be defined by the user and how they are enforced by the tool during the annotation process. Section 4 deals with how our approach extends naturally to cover multiple linguistic levels simultaneously. Section 5 gives a detailed description of both the tool

[1] MultiModal Annotation in XML. The current release version of the tool can be downloaded at http://www.eml.org/nlp.

itself and its Discourse API which offers high-level Java access to annotated corpora in MMAX format. In Section 6 we briefly discuss some related work.

2 Concepts

Linguistic annotation is the process and result of (manually) adding new information to existing language data. This existing data can consist of written texts (e.g. a newspaper corpus), but also of spoken language (which may even be multi-modal, i.e. contain e.g. pointing gestures). Before it can be annotated, this data must be converted into some machine-readable format. In addition, some rudimentary structure has to be imposed on it. What is important for both of these preprocessing steps is that they should not alter the original data in any way. In particular, they should not introduce arbitrary decisions or implicit assumptions. Instead, a format should be created that is as simple and theory-neutral as possible. In our approach, written text is simply modelled as a sequence of sentence elements, each of which spans a number of word elements. For spoken language (or dialogues), sequences of turn elements are used, each of which spans sequences of word elements[2]. Since the tokenization into words and the segmentation into sentences or turns can be performed on a mere formal (i.e. surface-based) level, we believe these elements to be sufficiently objective to serve as the structure for what we call annotation *base data*. This is in contrast to e.g. utterance segmentation, which has been shown to require a considerable amount of interpretation from human subjects. Therefore, we do not support utterance elements on the level of the annotation base data, but regard utterance segmentation as one possible level of annotation.

As for the XML implementation of the annotation base data, we simply model sentence and turn elements as XML elements with the respective name and with two obligatory attributes: The *ID* attribute assigns a unique label to each element, and the *span* attribute contains a (condensed) list of IDs of those base data elements that the sentence or turn contains.

```
<sentence id="sentence_1"
                  span="word_1..word_8"/>
```

[2]For multi-modal dialogue, turns can contain gesture elements in addition to word elements.

The <turn> element may have an additional *speaker* and *number* attribute.

```
<turn id="turn_1" span="word_1..word_7"
                 speaker="A" number="1"/>
```

Each word element in the base data is modelled as a <word> XML element with an *ID* attribute as the only obligatory one. The word itself is represented as a text child of the <word> element. If the original language data was spoken language, this is the transcription of the originally spoken word. In this case, the <word> element may also have an additional *starttime* and *endtime* attribute relating the word to a time line.

```
<word id="word_1" starttime="0.000"
                  endtime="0.7567">
   This
</word>
```

All elements comprising a MMAX document are stored in a *sentences* or *turns* file and a *words* file (and an additional *gestures* file for multimodal dialogue). These files define the annotation base data and are not supposed to be modifiable through the annotation tool.

2.1 Markables

Markables are the sole carriers of annotation information. The concept of *markable* is defined in formal terms only, i.e. without any implicit semantics. A markable is simply an abstract entity which aggregates an arbitrary set of elements from the base data. It does so by means of a list of IDs of word elements (and/or gesture elements), which are interpreted as pointers to the respective elements. Obviously, the question which sequences of elements are to be represented as markables depends on the linguistic level or phenomenon one is interested in: In the case of coreference annotation, markables would identify *referring expressions* in the base data, because it is on this level that information has to be added. If the task is dialogue act tagging, markables would be used to represent utterances.

Markables are modelled as <markable> XML elements which are similar to <sentence> and <turn> elements in that they consist (in their most basic form) mainly of an *ID* and a *span* attribute. The latter attribute, however, can be more complex since it can reference discontinuous (or fragmented) sequences of base data elements.

```
<markable id="markable_1"
      span="word_1..word_5,word_7" ... />
```

The placeholder dots in the example above are to indicate that a markable can indeed have many more attributes. These are described in sections 2.2 and 2.3. Markables pertaining to the same linguistic level are stored together in a *markables* XML file. In its header, this file contains a reference to an annotation scheme XML file (cf. Section 3).

2.2 Attributes

In order to really add information to the base data, it is not sufficient for a markable to identify the set of elements it aggregates. It also has to associate some *attributes* with them. In our approach, markables can have arbitrarily many attributes in the form of name-value pairs. At this time, two types of attributes are supported: *Nominal* attributes can take one of a closed set of values, *freetext* attributes can take any string (or numerical) value. The attribute names, types and possible values to be defined depend on the nature of the markables for which they are intended: In dialogue act tagging, markables represent utterances, thus a nominal attribute *dialogue_act* with possible values like *initiation*, *response*, and *preparation* etc. would be relevant.

On the XML level, attributes are expressed in the standard `name="value"` format on markable elements in the *markables* file. Note, however, that both the type of the attributes and their possible values (for nominal attributes) cannot be determined from the *markables* file alone, but only with reference to the annotation scheme (cf. Section 3) linked to it.

2.3 Relations

While markables and their attributes are sufficient to add information to independent sequences of base data elements, they cannot *relate* these to each other for the expression of structural information. Therefore, our approach is complemented by a means to express *relations* between markables. Currently, attributes of type *member-relation* and *pointer-relation* are supported. Attributes of type *member-relation* express undirected relations between arbitrary many markables. This relation can be interpreted as set-membership, i.e. markables having the same value in an attribute of type *member-relation* constitute an unordered set. Attributes of type *pointer-relation*, on the other hand, express directed relations between single source markables and arbitrarily many target markables. As the name suggests, this relation can be interpreted as the source markable pointing to its target markable(s). It is important to note that *member-relation* and *pointer-relation* are not attributes themselves. Rather, they are *types* of attributes (like *nominal* and *freetext*) which can be realized by attributes of arbitrary names. That means that for one markable, several different attributes of type *member-* and *pointer-relation* can be defined within the same annotation scheme. The attribute type simply defines how these attributes are interpreted. Like the concept of *markable* itself, relations are also defined only formally, i.e. without any semantic interpretation. Like markables, relations can be associated with any kind of semantic interpretation, depending on the annotation task at hand: For coreference annotation, it would be natural to use a *member-relation* attribute *coref_class* to model classes of coreferring expressions. In addition, a (binary) *pointer-relation* attribute *antecedent* could be employed to annotate the direct antecedent of a coreferring expression. As another example, if the task is annotating the predicate-argument structure of verbs, (binary) *pointer-relation* attributes like *subject*, *direct_object* and *indirect_object* could be used to link a verb to its arguments.

On the XML level, relations are expressed like normal attributes, with the only difference that their values are (lists of) markable element IDs (*pointer-relation*) or strings of the form *set_x* (*member-relation*).

```
<markable id="markable_2"
      span="word_14..16"
      coref_class="set_4"
      antecedent="markable_1"   ...    />
```

3 Annotation Schemes

Even on the same linguistic level, not every attribute or relation is applicable all the time or to every kind of markable. In coreference annotation, e.g., a markable that has been explicitly annotated as discourse-initial should not be allowed to have an *antecedent* attribute. Along the same lines, in predicate-argument structure annotation, a so-called

weather-verb like *"rain"* should not be allowed to have a pointer to its subject. Restricting the availability of attributes to only those that make sense in a particular situation is an important means to ensure annotation quality and consistency.

Dependencies of this kind can best be captured by formulating *constraints* on which attributes can occur together or which are mutually exclusive. In our approach, constraints of various types can be modelled in the *annotation scheme*. Generally, *annotation* (or *coding*) *schemes* are of central importance to any annotation task. They describe which phenomena are to be annotated using which set of attributes. Within the MMAX tool, annotation scheme development has been of special importance, because the expressiveness and the degree of customizability of the annotation scheme strongly determine how versatile and generally applicable the tool is. The mechanism for defining and handling annotation schemes described in what follows has been developed in collaboration with the Brazilian-French project COMMOn-REFs (Unisinos, São Lepoldo-RS, Brazil; LORIA/INRIA, Nancy, France) (Salmon-Alt and Vieira, 2002).

An annotation scheme defines all attributes (nominal, freetext, member-relation and pointer-relation) valid for a linguistic level. It specifies possible values for nominal attributes, and it identifies *default* attribute values. Attributes can be either *branching* or *non-branching*: If an attribute is branching, its current value influences which other attributes are available. In a branching nominal attribute, at least one of the possible values is associated with a reference to one or more following attributes. In a branching freetext, member-relation or pointer-relation attribute, on the other hand, at most two references to following attributes are possible, depending on whether the attribute does or does not have a value. Consider the following example (see Figure 1 for an illustration): In recent work dealing with pronoun resolution in spoken dialogue (Strube and Müller, 2003), different types of expressions (noun phrases, verb phrases, whole utterances and disfluencies) had to be annotated. They were distinguished by setting for each expression the appropriate value in a nominal attribute called *ExpressionsType*. Since noun phrases have different attributes than e.g. verb phrases, the attribute *ExpressionsType* was a branching one because each of its possible values referenced a partially[3] different set of following attributes: For noun phrases, the whole range of linguistic features like case, grammatical role, semantic role, etc. is relevant, while e.g. verb phrases and utterances (for our purposes) needed only be distinguished according to their type (attributes *VP Type* resp. *Utt. Type*). For unclassified expressions (*none*) and disfluencies, on the other hand, no further attributes were defined at all.

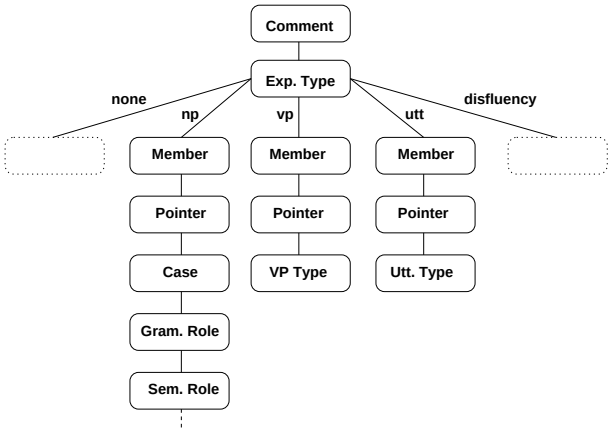

Figure 1: Annotation Scheme Diagram (Fragment)

Attributes that are referenced by branching attributes (e.g. *Member*, *Pointer*, *Case*, *VP Type*) are dependent in that they are only valid and accessible if the corresponding value is selected on the branching attribute (i.e. *ExpressionsType*). Thus, the availability of attributes can effectively be constrained. Since an attribute that is dependent on some other attribute can itself be branching, arbitrarily many levels of dependency are possible.

An annotation scheme of the above form can also be described as an annotation *tree*, where each node in the tree consists of a number of non-branching and (optionally) one branching attribute. If a node does have a branching attribute, the dependent attributes it references can be seen as child nodes. If a node does not have a branching attribute, it corresponds to a leaf in the annotation tree.

[3]The *Member* and *Pointer* attribute applies to noun phrases, verb phrases and utterances.

4 Levels

In Section 2, the linguistic levels of coreference, dialogue acts and predicate-argument structure were used for illustrative purposes. It was demonstrated how these different linguistic phenomena can be represented by means of a few simple concepts. The following section deals with how the same concepts lend themselves to the *simultaneous* representation of multiple levels of linguistic description.

Among others, the following levels of linguistic description could be envisaged:

- morpho-syntax,
- syntax,
- valency/predicate-argument structure,
- coreference,
- dialogue acts,
- prosody/intonation,
- ...

Relating e.g. the utterance level to the coreference level could be done, for instance, in order to find out whether utterance boundaries in spoken dialogues can be used to narrow down the search space for antecedents of discourse-deictic anaphors. Along similar lines, the prosody or intonation level could provide relevant information as well.

Though it would be tempting to merge markable files from different levels, this would have some serious disadvantages. First of all, subsequent modification or removal of a level would be cumbersome. Moreover, alternative versions of the same level (e.g. utterance segmentations performed by different annotators) cannot easily be compared without having to duplicate the other levels. For these reasons, our approach favours the separation of the different description levels to different markables files.

Since markables (as we define them) are not directly embedded into the base data, but reference base data elements by means of their *span* attribute, the simultaneous application of several description levels is straightforward: Given some annotation base data set, sets of markables pertaining to different description levels can simply be applied to it, i.e. be allowed to access the base data elements they reference, thus adding level by level of annotation. Since markables on different levels are related only indirectly by virtue of shared base data elements, issues like overlap or discontinuous elements do not arise. This is made possible through what can be seen as a rigorous implementation of the principle of *stand-off annotation* (Ide and Priest-Dorman, 1996; Thompson and McKelvie, 1997).

5 MMAX

5.1 The Annotation Tool

The MMAX annotation tool is written in Java. XML and XSL functionality is supplied by the Apache[4] Xerces and Xalan engines. The Java executable of the tool itself is very small (ca. 300 kb). Installing the tool (under Windows or Linux) is done by simply extracting a directory structure to the local hard disk; no further installation is required. Figure 2 shows a typical annotation situation with the most important GUI elements being visible, i.e. (clockwise, beginning in the upper left corner): the main annotation window, the *SearchWindow*, and the *AttributeWindow*. In the *SearchWindow*, a query for 3rd person neuter personal and possessive pronouns with oblique case is specified. Attributes can be queried by either selecting the desired value from a list, or by specifying a regular expression. The *AttributeWindow* shows the annotation scheme described in Figure 1.

Up to now, MMAX has been used for the creation of several annotated corpora, e.g. uni-modal text-only corpora (Salmon-Alt and Vieira, 2002; Müller et al., 2002; Strube and Müller, 2003) and multimodal human-machine corpora (Müller and Strube, 2001; Rapp and Strube, 2002).

In order to minimize the tool's system requirements and maximize its performance, we deliberately chose to use a text-only display (as opposed to an HTML display). This imposes a couple of restrictions with respect to user interaction. We distinguish between the display of **content-bearing** vs. merely **layout** information:

Content-bearing information is conveyed by markables and their properties. Within MMAX, it

[4] http://www.apache.org

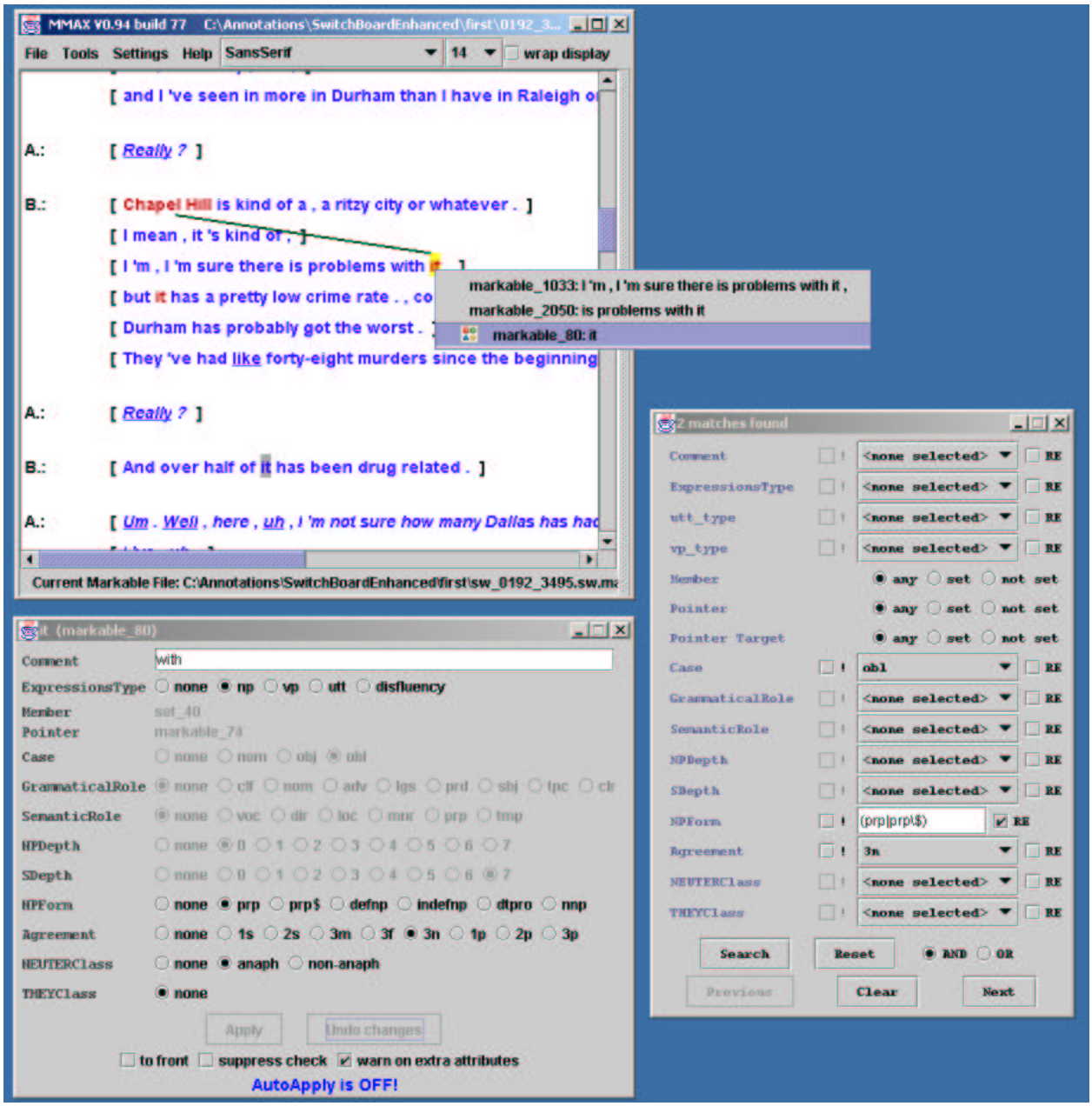

Figure 2: MMAX Screenshot

is visualized by means of text foreground and background colours and graphical arrows (for relations between markables). User actions like selecting, adding or deleting a markable, adding a relation between two markables, or modifying a markable's attributes change the content-bearing information and thus require frequent display updates. The MMAX display offers hard-coded (and highly optimized) methods for manipulating text colour and for drawing lines between markables. Thus we achieve very good performance (i.e. low response time) for these by far most frequent types of user interactions.

Layout information, on the other hand, contains *formal* properties of the display only. It includes mainly line breaks and indentations, but also font style properties like *bold*, *italic*, and *underlined*. Within MMAX, the XSL style sheet supplied in the .MMAX project file is responsible for rendering the display layout. By modifying this style sheet, the user can customize the display, e.g. by inserting pseudo-HTML tags like <bold>...</bold> or <italic>...</italic>. During a MMAX session, changes to the layout can only be made by explicitly reinvoking the style sheet processor, which, depending on the data and style sheet complexity, can take several seconds. In contrast to content-bearing information, however, layout information is not expected to require frequent updates. Utterance segmentation is one example of how the display layout might change as a result of markables being added to the annotation, i.e. if the user wishes to have line breaks inserted directly after markables respresenting utterances. This, however, can be performed reasonably well if the user does not rebuild the display after each single markable creation, but only after each, say, five or ten.

A MMAX session is started by opening a .MMAX project file. This file contains references to all files comprising a MMAX document:

- one *sentences* or *turns* XML file,
- a *words* XML file (and/or a *gestures* file),
- a list of *markables* XML files,
- an XSL style sheet file for rendering the *layout* of the MMAX display,
- an XML file specifying colour attributes for rendering the appearance of markables depending on their *content*.

When a .MMAX project file is opened, the tool first builds an XML DOM tree representation of the information supplied in the base data files. For the whole session, this tree serves as the read-only 'scaffold' to which annotations (given in the form of one or more markables files) are applied. Then, depending on which annotation levels the user chose to view, information about markables from these levels is added to the DOM tree as well. The DOM tree is then submitted to the XSL style sheet for transformation into a single string, which is then converted in a Java object of type *StyledDocument*. In the last step, markables in the *StyledDocument* are coloured according to their initial attributes, and the *StyledDocument* is finally presented to the user by assigning it to the MMAX display.

Users can explicitly activate markables on different annotation levels. Only if a level is active, markables on this level are displayed and can be accessed or modified. Users can select an active markable by left-clicking it. If the click is ambiguous, a popup menu is displayed containing all active markables in the clicked position. In this menu, markables are tagged with their respective level, so that users can easily select markables from a particular level (without having to temporarily deactivate all other levels).

Once a markable is selected, its attributes are displayed in a separate *AttributeWindow*. In addition, if it has a non-empty value for some *member-relation* or *pointer-relation* attribute, those are visualized by means of arrows drawn on the MMAX display. The *AttributeWindow* has access to the annotation scheme defined for the markable it currently displays. This enables the *AttributeWindow* to perform a consistency check on each markable by trying to 'reproduce' the annotation process that lead to this markable having this set of attributes.[5] It does so by traversing the annotation tree, beginning at the root, and recursively trying to match the attributes of the markable to the attributes defined at the current annotation tree node. If an attribute could be matched, it is consumed, and the *AttributeWindow*

[5]Thanks to Caroline Varaschin Gasperin (Unisinos, São Lepoldo-RS, Brazil) for providing some initial ideas on this.

is changed so that dependent attributes are accessible. If the matching process terminates before all attributes have been consumed, an annotation error or inconsistency has been detected. The same is true if an undefined attribute value is found on the markable. In both cases, a warning message is displayed to the user. Within MMAX, the *AttributeWindow* is the central location where the annotation scheme is enforced. Figure 3 gives an idea of the internal relations between different MMAX components. Bold boxes represent GUI elements.

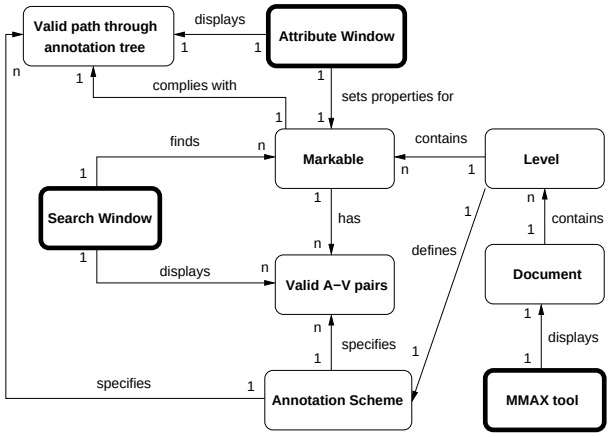

Figure 3: MMAX Components

Creating a new markable works straightforwardly by selecting a sequence of text on the display and right-clicking it. If only one annotation level is active, a pop-up menu with only one option, i.e. creating a markable on this level, will apear. Otherwise, the menu will contain options for creating a markable on any one currently active level.

When a newly created markable is selected, it does not have any attributes except for those defined on the root node of the annotation tree. The *AttributeWindow* utilizes the order of the annotation tree nodes to guide the user through the annotation process, allowing only those attributes to be displayed that are valid in the current annotation situation. As an additional help, each attribute and each value in the annotation scheme can have a textual description associated with it. During annotation, this description will be displayed to the user when they move the mouse over the corresponding item in the *AttributeWindow*.

Creation and deletion of relations between markables is performed in two steps: First, the source markable is selected as described above. Then the target markable is selected by right-clicking it. Then, another pop-up menu appears, the options of which depend on which relations have been defined for the source and target markable: If one or more *member-relation* attributes are defined for both markables, the user will have the option of adding the target markable to the set of the source markable (if it is already a member of one). If one or more *pointer-relation* attributes are defined for the source markable, the user will also have the option of pointing to the target markable. Deleting relations between markables works analogously. After each modification, the display is refreshed in order to reflect changes to the selected markable's attributes.

5.2 The Discourse API

The MMAX Discourse API[6] is intended as a platform for the exploitation and reuse of annotated documents in MMAX format. It maps the elements of the base data and the markables to Java classes and defines a set of basic operations to be performed on them. The entire document is wrapped in a Java *Discourse* object which serves as the single entry point. The *Discourse* object itself is created from the .MMAX project file by a *DiscourseLoader* class which parses the XML files and resolves references between elements. The result is a tree-like structure which can be navigated by accessing elements on a particular level and retrieving their child elements, which are Java objects themselves and can thus be used as entry points to their child elements as well. Consider the following example: *getSentenceCount()*, when called on the *Discourse* object, returns the number of sentences in the current document. This number can be used to iterate over all those elements by means of the *getSentence(position)* method, which returns the sentence at *position* as a Java *Sentence* object. Calling *getWordCount()* on this object returns the number of word elements the current sentence contains. *getWord(position)* returns the word at *position* as a Java *Word* object. These objects contain, among other things, a *getMarkables()* method which returns a list

[6]This section is based on (Müller and Strube, 2002) where an earlier version of the MMAX Discourse API is described in more detail.

of all markables (as Java *Markable* objects) a word is part of. Alternatively, *getMarkables(level)* returns only those markables on a particular level. On the level of *Markable* objects, the API contains a set of basic methods for e.g. retrieving attribute values. It also supports procedures like determining the formal relation between markables (identity, embedding, overlap, and the like).

6 Related Work

The work described in this paper is relevant for two distinct yet related topics: Representation models for linguistic data and development of annotation tools proper.

6.1 Linguistic Representation Models

The *Annotation Graph* model (Bird and Liberman, 2001) is a current representation model for linguistic data. Within this model, annotations are represented as labelled arcs connecting nodes on a common timeline. Each arc is associated with a particular type (like a phone, word, dialogue act, etc.), and a set of attribute-value pairs. While they are similar to MMAX markables in this respect, Annotation Graphs are much more powerful since they can model any phenomenon which can be mapped to sequentially aligned elements with a temporal extension. On the other hand, the dependence on time-aligned data might make it more difficult to model corpora without time stamps, like e.g. written text corpora. In principle, however, our approach and the Annotation Graph model serve rather different purposes: the former has been primarily designed as the internal representation format for the MMAX tool, and turned out to be useful as an independent representation model as well, while the ambition of the latter has been to create a general purpose model for the unification of diverse annotation system formats. Due to their similarity, however, both models are *compatible* with each other, and conversion from one into the other should be possible.

6.2 Annotation Tools

The NITE[7] (Bernsen et al., 2002) initiative is a project in which a workbench for multi-level, cross-level and cross-modality annotation of language data is developed. It is comparable to our tool only in that it explicitly addresses the simultaneous annotation on different levels. It is, however, much more ambitious than MMAX, both with respect to its intended scope of functionality and the features it offers for display customization. For instance, NITE offers plug-ins for speech signal visualization and even video annotation (Soria et al., 2002): The latter allows the user/annotator to insert information directly into the video data. In contrast to that, MMAX only supports read-only access for playback of audio (and possibly video) files associated with individual sentences or turns in the base data. NITE is even more advanced with respect to the display capabilities. Users have at their disposal not only plain text elements, but more powerful devices like tables, list, buttons and the like, which can be used to create highly functional displays by means of XSL style sheets. The downside, however, appears to be that even minor changes to the elements displayed make it necessary to reinvoke the style sheet processor, which may become time-critical for long or more complex documents. The NITE workbench, which still appears to be in a demo or prototype stage, is implemented in C++ and runs only on the Windows platform. This decision might be motivated by performance requirements resulting from the features mentioned above.

Apart from NITE, a number of smaller and more specialized tools for the annotation of individual linguistic phenomena exist, many of which are publicly available. The Linguistic Annotation website[8] contains pointers to a large number of those.

7 Conclusions

This paper presented the MMAX annotation tool which is based on the following major considerations. On the theoretical side, there is the simplification of annotations to a set of simple concepts based on the notion of *markable*. Markables are versatile in the sense that almost any kind of annotation can be expressed through them. In addition, arbitrarily many markables can refer to the same sequence of data without interfering with each other, even if they are overlapping or discontinuous. This makes it possible to use them for annotation of various levels of

[7] http://nite.nis.sdu.dk

[8] http://www.ldc.upenn.edu/annotation

linguistic description simultaneously. Another theoretical issue in the design of MMAX is its ability to express and enforce highly customizable annotation schemes. On the practical side, a main design feature is the deliberate restriction of the display capabilities. This, taken together with the rather simple markable concept, made it possible to implement a display which is quickly updatable and thus easily and conveniently usable, even if more than one annotation level (i.e. *markables* file) is displayed at the same time. The tool is implemented in Java, which has the additional advantage of being platform independent and easily extensible. We believe that all this taken together outweighs the disadvantages of a slightly 'impoverished' display.

Acknowledgements. The work presented here has been partially funded by the German Ministry of Research and Technology as part of the EMBASSI project (01 IL 904 D/2) and by the Klaus Tschira Foundation. We would like to thank the researchers from the COMMOn-REFs project, in particular Caroline Varaschin Gasperin, for their useful criticism and ideas on improving MMAX.

References

Niels Ole Bernsen, Laila Dybkjaer, and Mykola Kolodnytsky. 2002. THE NITE WORKBENCH – A tool for the annotation of natural interactivity and multimodal data. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation*, Las Palmas, Canary Islands, Spain, 29-31 May, 2002, pages 43–49.

Stephen Bird and Mark Liberman. 2001. A formal framework for linguistic annotation. *Speech Communication*, 33(1):23–60.

Nancy Ide and Greg Priest-Dorman. 1996. The corpus encoding standard. http://www.cs.vassar.edu/CES.

Christoph Müller and Michael Strube. 2001. MMAX: A tool for the annotation of multi-modal corpora. In *Proceedings of 2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, Seattle, Wash., 5 August 2001, pages 45–50.

Christoph Müller and Michael Strube. 2002. An API for discourse-level access to XML-encoded corpora. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation*, Las Palmas, Canary Islands, Spain, 29-31 May, 2002, pages 26–30.

Christoph Müller, Stefan Rapp, and Michael Strube. 2002. Applying Co-Training to reference resolution. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Penn., 7–12 July 2002, pages 352–359.

Stefan Rapp and Michael Strube. 2002. An iterative data collection approach for multimodal dialogue systems. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation*, Las Palmas, Canary Islands, Spain, 29-31 May, 2002, pages 661–665.

Susanne Salmon-Alt and Renata Vieira. 2002. Nominal expressions in multilingual corpora: Definites and demonstratives. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation*, Las Palmas, Canary Islands, Spain, 29-31 May, 2002, pages 1627–1634.

Claudia Soria, Niels Ole Bernsen, Niels Cadée, Jean Carletta, Laila Dybkjaer, Stefan Evert, Ulrich Heid, Amy Isard, Mykola Kolodnytsky, Christoph Lauer, Wolfgang Lezius, Lucas P.J.J. Noldus, Vito Pirrelli, Norbert Reithinger, and Andreas Vögele. 2002. Advanced tools for the study of natural interactivity. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation*, Las Palmas, Canary Islands, Spain, 29-31 May, 2002, pages 357–363.

Michael Strube and Christoph Müller. 2003. A machine learning approach to pronoun resolution in spoken dialogue. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan, 7–12 July 2003. To appear.

Henry S. Thompson and David McKelvie. 1997. Hyperlink semantics for standoff markup of read-only documents. In *Proceedings of SGML Europe '97*, Barcelona, Spain, May 1997.