An API for Discourse-level Access to XML-encoded Corpora

Christoph Müller, Michael Strube

European Media Laboratory GmbH
Villa Bosch
Schloß-Wolfsbrunnenweg 33
69118 Heidelberg, Germany
{christoph.mueller, michael.strube}@eml.villa-bosch.de

Abstract

We describe a simple and efficient Java object model and application programming interface (API) for (possibly multi-modal) annotated natural language corpora. Corpora are represented as elements like Sentences, Turns, Utterances, Words, Gestures and Markables. The API allows linguists to access corpora in terms of these discourse-level elements, i.e. at a conceptual level they are familiar with, with the flexibility offered by a general purpose programming language. It is also a contribution to corpus standardization efforts because it is based on a straightforward and easily extensible data model which can serve as a target for conversion of different corpus formats.

1. Introduction

A lot of effort is normally put into the development of powerful and expressive representation formats for natural language corpora. Initiatives like EAGLES[1] attempt to develop recommendations and guidelines for corpus encoding (like CES[2]), with the overall goal of establishing standards or quasi-standards to foster reusability and exchange of corpora among researchers.

In contrast to this, natural language corpora exploitation often is highly specialized and implies development overhead on the part of the researcher. This is because the corpus, even if it does conform to some standard, does not normally offer a defined set of methods to access its contents in a task-independent way. The result is that in many research projects software is created from scratch, used once for a particular corpus processing task, and is then discarded.

One way to maximize reusability not only of corpora, but also of the software components processing them, is the definition and implementation of programming interfaces to complement standardization efforts of natural language corpus representation.

The remainder of this paper is structured as follows. In section 2., we argue in favor of standardization in natural language corpora access methods. We then (section 3.) outline two existing approaches to this problem. Section 4. describes our own approach, the MMAX Discourse API, which is focussed on discourse-level phenomena. In section 5. we briefly sketch a selection of possible and actual use cases.

2. The Need for Standards in Corpus Access

In the field of corpus encoding, it is common practice to distinguish between the logical and the physical level of representation.

- The logical level defines an abstract data model which describes a corpus and its annotation in terms of conceptual elements, their attributes and possible values, and their relations to each other (Ide and Brew, 2000).
- The physical level implements the logical level data model in a way which allows its elements to be manipulated computationally and to be stored in and read from files.

For the latter task, XML is most often used nowadays. It allows the modelling of arbitrary elements with arbitrary relations, and thus supports the straightforward mapping of elements of the data model to elements of the physical level (i.e. XML tags). The XML-related technologies XLink and XPointer support "stand-off annotation"[3], a formalism which has emerged as a quasi-standard for the modelling of (possibly overlapping) relations between XML elements.

While the use of XML makes different language corpora similar on a technical level, they are obviously not compatible with one another unless there is agreement on the logical level as well. Required agreement would include basic issues like

- the granularity of the model, i.e. what the smallest independently accessible corpus element should be (e.g. phones, morphemes, words, discourse entities, or otherwise unspecified timed units), or
- the internal principle of organization of the model, i.e. which higher-level elements are used to impose a structure on the basic elements (e.g. sentences, turns, dialogue moves, or just time spans).

Corpus standardization initiatives like EAGLES attempt to provide answers to these questions. One of the major principles behind their work is theory-independence of the model, because it facilitates exploitation for as many different uses as possible. The problem with many standardized natural language corpora as they are used nowadays is that standardization ends as soon as their exploitation begins. To a large extent, this is due to the highly specialized tasks that corpora are used for: In the area of discourse processing, e.g., automatic dialogue-act tagging has entirely different requirements than anaphora resolution, and accordingly

[1] http://www.ilc.pi.cnr.it/EAGLES/home.html
[2] http://www.cs.vassar.edu/CES

[3] Introduced as "remote markup" by (Ide and Priest-Dorman, 1996).

different operations have to be performed on the corpus. Thus, what is common to all but the most simple tasks is that they require computational corpus processing of some kind. For many corpora there are tools for querying and extracting information (e.g. *TIGERSearch*[4] for treebanks like the Penn Treebank, the Susanne corpus, and the Negra corpus, *tgrep*, or *TTT*, the Text Tokenization Tool[5]), but these are often very specialized. If more flexibility is needed, linguists either have to fall back on generic XML or SGML processing tools, or they have to implement the required parsing capabilities themselves from scratch. The MUC-7 event, for which textual data was distributed as a set of SGML files, is a relevant example of how this was done in the recent past. This way of natural language corpus exploitation has several shortcomings:

- Linguists have to deal with technical intricacies of mark-up languages instead of being allowed to focus on the data at the conceptual level.
- Rapid-prototyping approaches to corpus processing are not encouraged because of pre-processing overhead.
- Algorithms (e.g. for anaphora resolution) implemented for use with certain corpora cannot easily be applied to other corpora because they probably access the data in a non-standard way. This complicates their re-use and in particular their evaluation.

3. Corpus Access via APIs

One way to avoid the disadvantages described above is to extend natural language corpus standardization to include the definition of standard *corpus access methods* as well. Like the model they operate on, these methods should be theory-independent and impose as little constraints as possible in terms of the applications that can be created with them. At the same time, technicalities of the physical level should be hidden from the user.

Corpus access methods can be defined in terms of application programming interfaces (APIs) to corpora. On a logical level, these formally describe

- the names and properties of the objects in terms of which the corpus is represented,
- the names of procedures that can be applied to them,
- the input parameters that a particular procedure may require, and
- the result of the procedure, which may be e.g. a string or a number, but also an object of a type defined in the model.

In order for an API to be practically applicable to existing corpora, it has to be implemented for a particular programming language like e.g. Java, C/C++, Tcl, or Perl. It is this language that users can then write their corpus processing applications in. Obviously, the choice of programming language should restrict as little as possible the type of application that can be created with it. It has to be noted, however, that there is some kind of trade-off between the power and flexibility of the programming language on the one hand and the demands it makes on the user in terms of programming skills and experience on the other. As the following brief discussion of some existing approaches shows, this trade-off appears to be commonly resolved in favor of greater power and flexibility.

3.1. GATE

GATE (General Architecture for Text Engineering)[6] is the result of research conducted at the University of Sheffield (Cunningham et al., 1996). It consists of, among other things, a number of Java applications for performing recurrent tasks in natural language processing, like e.g. tokenization, sentence boundary detection, and named entity recognition, and the GATE Document Manager, a data infrastructure through which these applications communicate. In GATE 1, this infrastructure is based on an implementation of the TIPSTER[7] document model. In this model, documents and their annotations are represented by associating attributes with spans of characters in the text, with the spans being identified by character offset and length. While this formalism is sufficiently powerful to model conventional documents, it does not appear to be extensible to cover non-hierarchical structures like overlapping in dialogues. The main reason for this is that the TIPSTER model was primarily designed for information retrieval and extraction tasks in a text-only domain. It is interesting to note that in the current version (GATE 2) (Cunningham, 2000) the original data model has been modified in order to comply with "Annotation Graphs", a much more flexible formalism also underlying the ATLAS approach (cf. section 3.2.). The field of application of GATE, however, appears to be restricted to uni-modal and non-dialogue domains.

3.2. ATLAS

ATLAS (Architecture and Tools for Linguistic Analysis Systems)[8] is a joint effort of NIST, LDC and MITRE. It comprises, among other things, a data model for language corpora and an API to access these corpora from programming languages like Java or C++. The data model is based on the notion of "Annotation Graphs" (Bird and Liberman, 1999). Within this framework, corpora and their annotations are represented as a set of labelled arcs connecting nodes on a common timeline. Each arc is associated with a particular type (like e.g. phone, word, dialogue act etc.) and a set of arbitrary attribute-value pairs. Embedded or overlapping phenomena on different levels can be represented in an elegant and straightforward way by arcs of different types which may share none, one or both of their nodes. Annotation graphs are read from and written to files in a special XML-based ATLAS Interchange Format (AIF).

The ATLAS Java API offers access to a corpus in terms of elements like *Annotation*, *Region*, *Content*, *Anchor*, and

[4] http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERSearch
[5] http://www.ltg.ed.ac.uk/software/ttt
[6] http://gate.ac.uk
[7] http://www.itl.nist.gov/iad/894.02/related_projects/tipster
[8] http://www.nist.gov/speech/atlas

Signal.[9] These are generalisations of arcs and nodes, which have been adopted to be able to cover data with more than one dimensionality, like e.g. video (Bird et al., 2000).

The ATLAS approach has several strengths: Since it is so general, it is very powerful and can be used to model a wide variety of phenomena as long as these can be mapped to sequentially aligned elements which have a temporal extension. Moreover, graphs and graph traversal are mathematically well-understood notions for which efficient algorithms exist. Therefore, ATLAS is certainly going to establish itself as a useful and widespread standard for natural language (including multi-modal) corpora.

On the other hand, the expressive power of ATLAS is based on a high level of detail in the data model. This complicates access to higher-level elements via the API. What is more, the elements in terms of which this access takes place are rather abstract and not of the kind that a linguist is familiar with. It could thus turn out that technical intricacies of representation languages, from which the API abstracts, are replaced by intricacies on another level.

4. The MMAX Discourse API

The MMAX Discourse API is an approach towards the development of reusable software components for discourse processing tasks. It is centered around a Java *Discourse* object building on a simple discourse model which has originally been developed as the basis for the MMAX annotation tool[10]. The logical level discourse model is physically implemented in XML. Figure 1 shows the structure of a small fragment of a sample dialogue which has been converted from the Switchboard SWBD-DAMSL corpus[11].

4.1. Logical Level

The discourse model can represent two basic types of corpora: textual and (possibly multi-modal) dialogue corpora. Depending on the type, different structural, pragmatic and base level elements are modelled. Apart from a set of system attributes (some of which will be mentioned in the following), each of these elements can be associated with a set of arbitrary attribute-value pairs. These are mapped from the XML files from which a particular instance of a *Discourse* object is created.

Structural level The structure of texts is described in terms of *text* elements, which contain a number of *sentence* elements.

Accordingly, dialogue structure is described in terms of a *dialogue* element containing a number of *turn* elements. System attributes of *turn* elements include e.g. a *speaker* attribute (*A* and *B* in turns 7-10 in the dialogue in figure 1).

Pragmatic level On the pragmatic level, both texts and dialogues can optionally be segmented into *utterances*, which can, but do not have to correspond to elements on the structural level. For dialogues, in particular, this allows for the representation of discontinuous utterances resulting from speakers interrupting each other or speaking at the same time (cf. e.g. utterance 13 and 15 in figure 1). Among others, *utterance* objects contain a *dialogue_act* attribute (e.g. *aa*, *qy* in figure 1)[12].

Base element level The model at this time supports up to three base level elements corresponding to three different modalities: *word*, *gesture*, and *keyaction*, the latter being intended to capture human-machine interaction (like the pressing of buttons) through a GUI or on a remote control. For text corpora, only *word* elements are permitted, whereas dialogue corpora may be multi-modal. In that case, the base level elements' system attributes *starttime* and *endtime* must be specified in order to assure correct chronological ordering of elements across modalities.

Annotations Finally, corpus annotations are represented in terms of *markable* elements which specify (single or sequences of) base level elements and associate an arbitrary set of attribute-value pairs with them. A generic *type* attribute can be used to distinguish different classes[13] of markables. Relations between markables are modelled by two attributes: a *member* attribute, which describes unordered sets of markables, and a *pointer* attribute which states an ordered relation between two markables. The semantics associated with each relation is not pre-defined by the discourse model itself. Instead, it is supplied by the annotation scheme used in the production of the annotation and can be different for different annotations, but also for different types of markables within the same annotation. Though simple, the two relations are sufficiently powerful to express a number of relevant concepts: E.g., if a particular annotation scheme uses markables to identify discourse entities, sets of markables with identical values in their *member* attributes can be interpreted as equivalence classes, thus allowing the modelling of coreference, and the *pointer* attribute can be used to identify a discourse entity's direct antecedent within such a class (Müller and Strube, 2001). If, on the other hand, markables represent syntactic phrases, hierarchical tree structures can be modelled using *pointers* from child to parent constituents.

4.2. API Level

The MMAX Discourse API maps the elements of the logical level to Java classes and defines a set of basic operations to be performed on them. The entire discourse is wrapped in a Java *Discourse* object which serves as the single entry point to the corpus and its annotation. The *Discourse* object itself is created from a set of XML files by a *DiscourseLoader* class which parses the files and resolves references between elements on different levels. The result is a tree-like structure (cf. figure 2) which can be navigated by accessing elements on a particular level and retrieving their "child" elements, which can then be used as entry points to their "child" elements as well.

The syntax and semantics of the methods which implement these navigation operations conform to the standard object-oriented way. Example: *getTurnCount()*, when

[9] http://www.nist.gov/speech/atlas/develop/core.html

[10] Download at http://www.eml.org/nlp

[11] http://www.colorado.edu/ling/jurafsky/ws97/dataindex.html

[12] http://www.colorado.edu/ling/jurafsky/manual.august1.html

[13] Our annotation tool MMAX, e.g., uses the *type* attribute to implement user-definable constraints on permitted markable attributes and possible values.

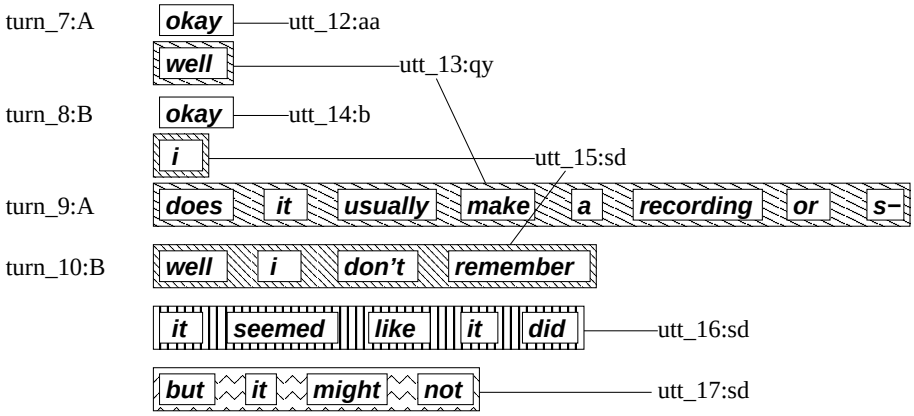

Figure 1: SWBD-DAMSL corpus sw_0001_4325 (fragment)

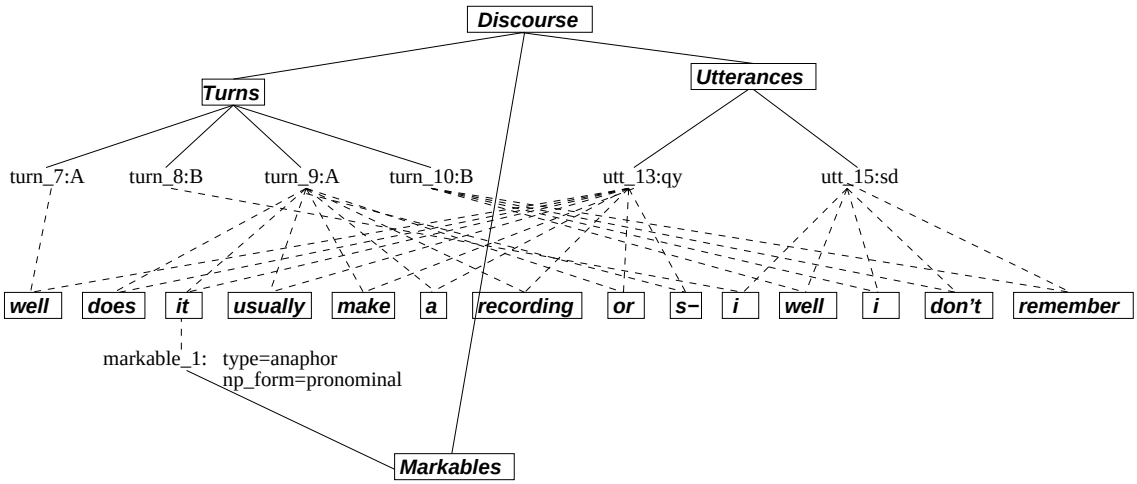

Figure 2: Discourse "tree" representation of a fragment of the Switchboard SWBD-DAMSL corpus

called on the *Discourse* object in figure 2, returns the number of *Turn* elements contained in the discourse (i.e., four, in this fragment). This number can be used to iterate through all *Turn* objects in the discourse, using *getTurn(position)*. For position=3, this method would thus return turn_9 (the third in the fragment). Since each turn retrieved in this way is itself a Java object of type *Turn*, processing on this level can continue in the same manner: *getElementCount()*, when called on turn_9, returns the number of base level elements this turn consists of (i.e. eight). The method *getElement(position)*, finally, returns the element at the specified position as a Java object of type *Word*, *Gesture*, or *Keyaction*. Thus, *getElement(2)* returns the word "it". The base level element representing this word is at the same time annotated as being a *Markable* with one user-defined (*np_form*) and one system attribute (*type*).

On each level in the entire *Discourse* object, user-defined as well as system attributes can be accessed using a generic *getAttributeValue(attribute_name)* method: *getAttributeValue("np_form")*, e.g., when called on markable_1, returns "pronominal". In addition to vertical navigation through the discourse, horizontal navigation between elements on the same level (e.g. from one word in a turn or sentence to the next), is supported in the same way.

Apart from turn- and sentence-based access, the discourse can also be traversed in terms of its pragmatic composition, i.e. based on its segmentation into utterances. Since one step in the creation of a *Discourse* object is the mapping of elements from the structural and the pragmatic level onto each other, it is possible at every stage to access corresponding elements on the other level. Example: *getTurns()*, when called on utterance_13 in the above example, returns the set of *Turn* objects (i.e. turn_7 and turn_9) that it is mapped to. Mapping of elements of the structural and the pragmatic level is done by virtue of shared base level elements: Each base level element can be part of at most one pragmatic and one structural element, which can be retrieved by *getUtterance()* and *getTurn()* resp. *getSentence()*.

In addition, the API supports a set of basic methods for procedures like determining the formal relation between elements (e.g., embedding or overlap of markables), filtering elements on particular attribute values, and the like.

5. Sample Use Cases

Although still in an early stage of development, the MMAX Discourse API has already been applied to a number of discourse processing tasks (in a wider sense). These include

- implementation of a number of reference resolution algorithms,
- generation of training and test data for machine learning for coreference resolution from annotated corpora,
- implementation of the coreference resolution evaluation algorithm according to (Vilain et al., 1995),
- generation of training and test data for machine learning from an annotated corpus of multi-modal human-computer interaction within the EMBASSI project[14],
- generation of training data for an HMM-based dialogue act tagger from a converted Switchboard SWBD-DAMSL corpus.

Despite the relatively low level of detail (as compared to e.g. the ATLAS approach), the model and API proved sufficient for each of these tasks. In fact, it had a positive effect in many cases because the Java code that needed to be written was more self-explanatory than it would have been if the application had accessed the corpus using generic XML processing methods. The simplicity of the data model is also an advantage when it comes to converting corpora from different formats. While the availability of our annotation tool, which produces annotations in the required format, is certainly a major advantage, we believe that our model, due to its generality, is equally applicable as the target format for conversion from other formats. Thus, the volume of available corpora can easily be extended by writing conversion programs which, once written, can be applied to any number of available corpora in the particular format. Conversion from other formats is supported in particular by the fact that our model is open in the sense that arbitrary data from the source corpus (e.g. POS tags, but also syntactic markup) can be stored as XML attributes on the appropriate level (i.e. the *word* element, or a *markable* element of type "constituent", in which case the *markable's pointer* attribute can be used to relate it to its parent element), and is mapped to the respective Java object in the *Discourse* object.

6. Conclusions and Future Work

This paper described a simple Java object model and API for the representation and discourse-level processing of annotated corpora. Both the model and the API work in terms of straightforward concepts that linguists are familiar with. We outlined the structure of the model and the way in which the API makes it accessible from Java. Apart from continuing use of the API as outlined in section 5., future work on the topic will include the following: We will define and implement more higher-level elements which are derived from the simple basic elements supported so far. The suitability of our model as the target format for corpus conversion will be further tested by developing conversion tools from additional formats. Finally, we will investigate ways to add methods not only for element *access*, but also for element *creation*, which would allow the discourse model and API to be used as the basis for the development of annotation tools.

Acknowledgements We thank the anonymous reviewers for their useful comments. The work presented here has been partially funded by the German Ministry of Research and Technology under grant 01 IL 904 D/2 (EMBASSI) and by the Klaus Tschira Foundation.

[14] http://www.embassi.de

7. References

Steven Bird and Mark Liberman. 1999. Annotation graphs as a framework for multidimensional linguistic data analysis. In *Proceedings of the ACL '99 Workshop Towards Standards and Tools for Discourse Tagging*, College Park, Md., 21 June, 1999, pages 1–10.

Steven Bird, David Day, John Garofolo, John Henderson, Christophe Laprun, and Mark Liberman. 2000. Atlas: A flexible and extensible architecture for linguistic annotation. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation*, Athens, Greece, May, 2000.

Hamish Cunningham, Yorick Wilks, and Robert Gaizauskas. 1996. Gate – a general architecture for text engineering. In *Proceedings of the 16th International Conference on Computational Linguistics*, Copenhagen, Denmark, 5–9 August 1996.

Hamish Cunningham. 2000. *Software Architecture for Language Engineering*. Ph.D. thesis, University of Sheffield.

Nancy Ide and Chris Brew. 2000. Requirements, tools and architectures for annotated corpora. In *Proceedings of Data Architectures and Software Support for Large Corpora. Paris: European Language Resources Association*, pages 1–5.

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

Christoph Müller and Michael Strube. 2001. Annotating anaphoric and bridging relations with MMAX. In *Proceedings of 2nd SIGDial Workshop on Discourse and Dialogue*, Aalborg, Denmark, 1-2 September 2001, pages 90–95.

Marc Vilain, John Burger, John Aberdeen, Dennis Connolly, and Lynette Hirschman. 1995. A model-theoretic coreference scoring scheme. In *Proceedings of the 6th Message Understanding Conference (MUC-6)*, pages 45–52, San Mateo, Cal. Morgan Kaufmann.