

JAAE: The Java Abstract Annotation Editor

Ivan Habernal¹, Miloslav Konopík¹

¹University of West Bohemia, FAS, DCSE
Univerzitní 8, 306 14 Pilsen, Czech Republic

konopik@kiv.zcu.cz

Abstract

Recent trends in NLP (Natural Language Processing) are heading towards a stochastic processing of natural language. Stochastic methods, however, usually demand a lot of annotated training data. In most cases, the annotation of the data has to be done manually by a team of annotators and it is a highly time-consuming and expensive process. Thus we tried to develop an efficient and user-friendly editor that would aid human annotators to create the annotated data. We offer this editor for free. The developed editor is described in this article.

Index Terms: data annotation, annotation editor, semantic analysis

1. Introduction

This article describes a graphical editor that was designed to enable creation of the annotated data for stochastic models training. Annotated data are the data that are augmented with some kind of additional information. For example during semantic annotation words of a sentence are associated with a meaning representation.

The described editor was originally designed for annotation of sentences by the Abstract Annotation methodology (see [1]) and it was developed in the Java programming language. Therefore we call the software the Java Abstract Annotation Editor (JAAE). The editor was, however, designed to be general enough to be used for various annotation purposes (not only semantic annotation). The only conditions that have to be fulfilled to use the software are:

- the data that are being annotated should consist of string tokens,
- there has to be a predefined annotation schema that determines the choices of an annotator.

In this article we concentrate particularly on the problematic of semantic analysis (see [2]) and on the use of the editor for annotation of semantics.

There is a large amount of different annotation editors that can be found on Internet. They are mostly too specific to be reused in other projects. Moreover, the semantics is often stored in the so-called bracket notation¹. See e.g. the DAE² which is a dialog acts and semantic annotation editor that despite its qualities uses the bracket notation for representing the semantics. The bracket notation requires the annotators to keep the brackets consistency as well as the correctness of node names.

¹The bracket notation stores the hierarchic structure within a structure of nested brackets. E.g. A1(B1,B2(C1)) means that A1 is on the top, B1 and B2 are directly below A1, etc.

²Dialogue Act Editor can be found at <http://ui.zcu.cz/projects/dae/>

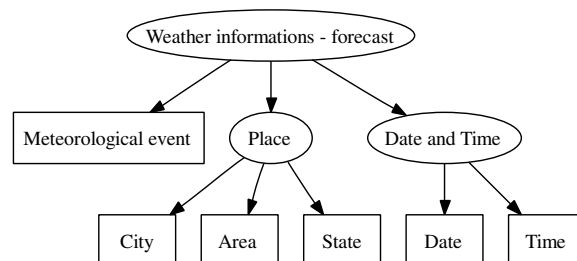


Figure 1: A fraction of weather forecast annotation scheme

Annotators can also make another mistake by including a node to a place where it is not allowed. Our annotation editor eliminates the danger of these mistakes because it uses an *annotation schema* and a GUI³ that guides the annotator during annotation process. There is also a possibility to create a *custom annotation schema* that enables the editor to be used in a variety of annotation tasks (see section 5).

2. Principles of Sentence Annotation

In this section, the basic principles of how to use the editor for sentences annotation are explained. The annotation process will be demonstrated by a simple example of a semantic annotation in the weather forecast domain.

As a first step of using the editor, it is necessary to specify the annotation scheme. The annotation scheme is used to lead an annotator during the annotation process. The annotation scheme is a hierarchical structure (a tree) that defines options that the annotator has at certain levels of the annotation. For example see figure 1. The figure shows a fraction of the annotation scheme. Namely it shows the one from the "Weather Forecast" theme. The scheme defines that the "Weather Forecast" theme has "Meteorological event", "Place" and "Date and Time" concepts. E.g. the "Place" concept consists of "City", "Area" and "State" sub-concepts and so on. This scheme thus says that if the annotator selects "Weather Forecast" theme then he or she can choose only from "Meteorological event", "Place" and/or "Date and Time" concepts. The theme is connected with the whole sentence. The concepts are connected with segments of a sentence (e.g. there could be a segment or a phrase that relates to a place in the sentence; then we select "Place" concept). If a concept belongs under another concept in the scheme it means that it is a subsegment of the upper concept. E.g. the "City" could be a subsegment of the segment that relates to the "Place" concept.

³GUI = Graphical User Interface

When the annotation scheme is specified and a set of sentences is available, the process of annotation can begin. For each sentence the annotator at first chooses the theme of the sentence (from the set of the themes that is defined in the annotation scheme). Then he or she can select concepts that belong to that theme. At last, when a leaf (the bottommost node of the theme tree) is reached, the annotator can optionally assign a word or a phrase (group of words) to the leaf. When all relevant concepts are assigned the sentence is annotated and annotator can proceed to a next sentence.

The annotator can optionally mark every part of the annotation tree as a problematic part. It is also possible to write down a reason why the part is problematic. This option is quite important because such a disputable sentence can be then consulted with an annotation coordinator.

3. Technical realization

The JAAE is a desktop application written in the Java programming language. All the data files are in the XML format.

There are two input files and one output file. The first input file is a set of sentences. The second one is an annotation schema. The output file stores the sentences and their annotation trees.

The XML file with the set of sentences has a very simple structure. The root XML element is `<dialog>`. The text of the sentences is placed within the `<request>` elements. All other elements and attributes of elements are ignored so that the input file can carry additional information (for example system responses in a dialog transcription). Thus in the following example the elements `<response>` and `<event>` will be ignored by the editor.

```
<?xml version="1.0" encoding="utf-8"?>
<dialog>
  <request>Text of the sentence 1.</request>
  <response>Optional text of a
    system response.</response>
  <request>Text of the sentence 2.</request>
  <event>Additional optional
    information</event>
  ...
</dialog>
```

The annotation schema XML file is a little more complicated. It has a strict structure. Thus we created an XSD⁴ document for it. The XSD defines the structure of the annotation schema. The names of elements are self-explanatory and so they do not need any comment.

With the XSD defined, it is easy to create the annotation schema because any XML editor that supports XSD will serve as a guide during the editing of the annotation scheme. An XML editor with XSD support ensures that the annotation scheme is syntactically valid and such an editor usually supports auto completion. Creating the XML schema is then simple and fault proof.

⁴XSD is an abbreviation of XML Schema Definition. XSD is used as a structure definition of a XML document. The XML Schema language has been published by W3C. It can be used to express a schema: a set of rules to which an XML document must conform in order to be considered 'valid' according to that schema. The XML Schema allows to define elements and attributes which can appear in an XML document, relations between elements, element content and element or attribute datatypes as well.

More information about the XML Schema can be found at W3C website <http://www.w3.org/XML/Schema>.

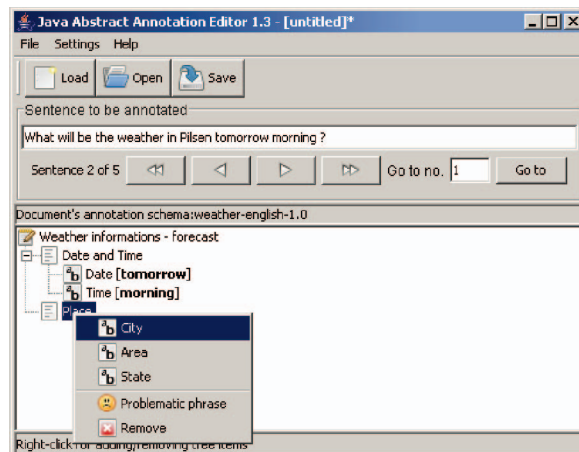


Figure 2: Graphical user interface of JAAE

We also created a XSLT⁵ script that transforms an annotation schema into a graph description language used by Graphviz⁶. Creating a graph like the one in the fig. 1 is done very simply by applying the transformation and then by running the Graphviz software.

The output file of the JAAE editor is also an XML file. The structure of the output file is also described in the enclosed XSD file so it does not need to be explained in this article.

4. Editor features

This section describes and summarizes the main features of the JAAE editor.

Platform independence. The application can be run on various platforms (e.g. Windows, Linux, MAC OS, etc.). It only requires JRE 1.5+⁷ to be installed.

User interface. The GUI is user-friendly and the application has an intuitive interface (see fig. 2). The graphical user interface is created in the JFC/Swing⁸.

Custom annotation schemas. Users can create their own annotation schemas in the XML format according to the XSD validation schema. Custom schemas can be loaded manually or automatically during the application start.

Problematic phrases. Any part of an annotation tree can be marked as problematic stating an appropriate reason.

Internationalization. The application supports localized messages; translations can be added easily by editing an XML file. Currently, the English and Czech versions are available.

Auto-save. The document that is being annotated is periodically saved to a temporary file.

⁵Extensible Stylesheet Language Transformation (XSLT) is a language for transforming XML documents into other documents (XML, HTML, text).

⁶Graphviz (acronym for Graph Visualization Software) is a package of open source tools originally created by AT&T Research Labs for drawing graphs specified in a simple text language for graph description.

⁷Java Runtime Environment, can be obtained from <http://www.java.com/en/download/>

⁸See <http://java.sun.com/docs/books/tutorial/uiswing/start/about.html> for more details.

4.1. XOM Library

The XOM Java library is used for processing XML files. It is an open source tree-based API for processing XML. The XOM has been developed as a correct, simple and effective XML library, in comparison with the standard Java XML API⁹. The main advantages of this library are:

- *memory efficient*; allows to filter document as it is being built and skip uninteresting nodes immediately, can process documents greater than 1 GB
- *dual streaming/tree-based API*; individual nodes of the tree can be processed while the document is still being built.
- *supports other XML technologies*; for instance namespaces, XPath, XSLT, XInclude, etc.

5. The Usage of the Editor

We use the editor in our COT-SEWing¹⁰ project. One part of the project deals with semantic analysis of user queries on the internet. We selected several typical areas of user interests (e.g. weather forecast, accommodation demands, requests to find a product to buy on internet, etc.) that have reasonable databases on internet. We try to develop a system where the user could use a natural language sentence (rather than keywords) to find the requested answer. Thus, our effort is focused on development of a system for semantic analysis and interpretation of user queries (we work with Czech language).

We plan to use a stochastic algorithm for semantic analysis (see [2]). Namely, we will use some ideas from Abstract Annotation methodology (see [3],[1]). Thus a large data set of an annotated training data is required to train the stochastic semantic analyzer.

We have collected 20 292 user queries (in sentences rather than in keywords) by a system simulation. Because we needed a collection of a real data we created the system that simulates an information retrieval system working in natural language. Then we asked our students to type questions into the simulation system. All the questions were stored in a database. The collection of the sentences took half a year. Approximately 450 people participated in the collection of the data.

Then we assembled a team of 4 annotators and we trained them for sentence annotations.

In the first run of the annotations our annotators only distinguished whether a query is suitable for further processing. A suitable sentence had to fit the predefined area of questions (the area in which we can answer the question according to an internet database). Their task was to annotate the theme of a sentence and its problems (off-topic, not answerable, not complete question, grammatical error and colloquial expression).

Then we created another annotation scheme for the annotation of a meaning of the sentences that were designated as suitable for further processing.

Each sentence is annotated by 2 annotators. Finally, the results of the annotations are compared and the annotation collisions are resolved.

During the time of the first run of annotations (3 months) we found and solved many flaws of the editor. Now the edi-

tor (version 1.3) is being used in the second run for 4 months without any important change.

5.1. Editor efficiency

The editor efficiency could be measured according to the reports given by the annotators. Nowadays they are able to annotate a sentence in an average of 34 seconds. The annotations contains in average 5-6 concepts in approx. 3 levels.

We have never performed a comparison with the bracket notation approach. However it is obvious that the editor saves some time because the bracket notation requires annotators to write down the names of concepts instead of clicking on the names with a mouse. It also requires the annotator to check the brackets consistency. At last bracket notation without an editor support could be syntactically or semantically incorrect.

5.2. Annotation Schema Creation

In the case of the semantic annotations, the annotation schema is usually based on a real database in practice. It is reasonable if the annotation schema reflects the structure of the available database. The ability to analyze and understand the meaning of a query that we are not able to answer is usually not that much useful.

Another point of view that should be taken into account during the schema creation is the form of the result we need. It is useful if the annotations are easily transformable to a final meaning representation. That means that the structure of the schema should not be very distinct from the structure of the final meaning representation (i.e for example a frame structure, an SQL query, etc.).

5.3. Other Uses of the Editor

The editor is usable for every type of annotation where we have a hierarchical structure that defines the annotations. The editor do not impose any limitation on the number of levels (the depth) of the scheme. The editor is most suitable for semantic annotations, though it can be used for other purposes.

One rather straightforward use is the one for morphological annotations. Here, we can use a part-of-speech as the top-most element in the annotation schema. Then the morphological categories that relate to the part-of-speech are subparts of the part-of-speech (e.g. gender is a subpart of the noun part-of-speech). This approach is suitable particularly for structured morphological tags that are commonly used in languages with rich morphology, such the Czech language (see [4]) is.

6. Conclusion

The Java Abstract Annotation Editor proved to be an efficient and easy-to-use tool for the semantic annotation in our project. The editor ensures that the annotations are syntactically correct and conform to a predefined schema. We believe that the editor helps to reduce the effort to create the annotated data for training significantly.

The purpose of this article is to offer information about a possibly useful piece of software rather than present a novel approach to semantic annotation. We hope that the software we created for our project would be useful in other projects too. We gratefully welcome any suggestions for further improvement and information about flaws of the editor.

⁹At <http://www.xom.nu/whatswrong/> there can be found more information about disadvantages of XML API

¹⁰Complex Knowledge Base Tools for Natural Language Communication with the Semantic Web

7. Download

The described software could be obtained from <https://lks.fav.zcu.cz/mediawiki/index.php/JAEE> for free.

8. Acknowledgements

This work was supported by grant no. 2C06009 Cot-Sewing.

9. References

- [1] H. Yulan and S. Young, "Semantic processing using the hidden vector state model," *Computer speech & language*, vol. 19, no. 1, pp. 85–106, 2005.
- [2] M. Konopík, "Stochastic semantic parsing," University of West Bohemia in Pilsen, Pilsen, Tech. Rep. DCSE/TR-2006-01, 2006.
- [3] S. Young, "The statistical approach to the design of spoken dialogue systems," Cambridge University Engineering Department, Tech. Rep. CUED/F-INFENG/TR.433, 2002. [Online]. Available: citeseer.ist.psu.edu/young02statistical.html
- [4] J. Hajič and B. Vidová-Hladká, "Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset," in *Proceedings of the COLING - ACL Conference*, Montreal, Canada, 1998, pp. 483–490.