2nd Linguistic Annotation Workshop, 26-27 May 2008, pp. 61-64

Multiple Purpose Annotation using SLAT — Segment and Link-based Annotation Tool —

Masaki Noguchi[†], Kenta Miyoshi[†], Takenobu Tokunaga[†]
Ryu Iida[‡], Mamoru Komachi[‡], Kentaro Inui[‡]

[†] Department of Computer Science, Tokyo Institute of Technology
Tokyo Meguro Ôokayama, 152-8552, Japan
{mnoguchi,kmiyoshi,take}@cl.cs.titech.ac.jp

[‡] Graduate School of Information Science, Nara Institute of Science and Technology
Nara Ikoma Takayama 8916-5, 630-0192, Japan
{ryu-i,mamoru-k,inui}@is.naist.jp

Abstract

In recent years, the use of large scale corpora in NLP applications, such as statistical parsing, has become prominent. As their use gained credibility, naturally so did the types of information they provided. There exist today many groups that create corpora: ANC, SFB at the University of Potsdam, just to name a few. In many cases these groups also provide specialized annotation tools for their corpora. However, these tools are just that: specialized, i.e. designed to work with a very specific annotation definition, without flexibility in mind. In the early stages of a project, often times the specification for annotating changes. This makes it difficult to use a tool with such rigid boundaries. In this paper, we propose a browser-based annotation tool SLAT, which allows for easily adding and customizing annotations. We also explain the steps involved in customizing SLAT to meet a user's project needs.

1. Introduction

In recent years, the use of large scale corpora in NLP applications, such as statistical parsing, has become prominent. As their use gained credibility, naturally so did the types of information they provided.

There are many projects which construct corpora, such as ANC[1] and Sonderforschungsbereich (SFB) on information structure at the University of Potsdam[2] just to name a few. The annotation of sentences by hand is not only extremely time consuming, but also leads to various kinds of errors. These errors combined with other user-entered biases have a large effect on the performance (and subsequent evaluation) of systems trained on these corpora. Thus, the information provided by corpora must be both accurate and consistent. To this end, annotation tools for simplifying and constraining human input have been developed in various projects, and have decreased the costs of constructing corpora. These tools are developed to work with a very well defined annotation specification. In the early stages of a project often times the specification for annotating will change, making it difficult to use a tool with such rigid boundaries. The format for storing information also differs by tool, so their data is not immediately interoperable. The conversion of one format to another is required each time an experiment is conducted or a method evaluated.

In the next section, we briefly review some existing annotation tools and then describe our motivations for developing a new annotation tool. We introduce SLAT [sléit] (Segment and Link-based Annotation Tool), aimed at satisfying these motivations and briefly explain its features. Lastly, we summarize this paper and describe future work.

[1] http://www.americannationalcorpus.org
[2] http://www.sfb632.uni-potsdam.de/

2. Requirements for Annotation Tools

Stefanie Dipper et al.(Dipper et al., 2004) compared existing tools that use XML as their data storage format. They compared twelve individual research projects from several disciplines, having corpora that mostly consisted of 5 types of annotations: semantic, discourse and focus annotations, as well as diachronic data and typology. To manage these types of annotations, they described seven requirements for annotation tools: diversity of data, multi-level annotation, diversity of annotation, simplicity, customizability, quality assurance and convertibility. First three relate to data annotation while the latter four relate to the usability of the annotation tool. They compared five annotation tools to test the validity of these criteria: TASX Annotator[3], EXMARaLDA[4](Thomas, 2001), MMAX[5](Müller, 2006), PALinkA[6](Orăsan, 2003) and Systemic Coder[7].

3. Requirements during the Early Stages of a Project

As presented in the previous section, an annotation tool must satisfy these requirements to be successful in corpus annotation. Previously developed annotation tools have mostly focused on the usability of the system regarding the annotation task itself, i.e. how easy/difficult it is to add/remove annotations. Usability is clearly important. In designing an annotation tool, however, it is also crucially important to take the while demands of a corpus project, which typically not only annotate text but also designing the tag set and evaluate and maintain the resultant corpus,

[3] http://tasxforce.lili.uni-bielefeld.de/
[4] http://www.exmaralda.org/
[5] http://www.eml-research.de/english/research/nlp/download/
[6] http://clg.wlv.ac.uk/projects/PALinkA/
[7] http://www.wagsoft.com/Coder/

into account as design issues. More specifically, at least the following three issues should be addressed so that the tool can effectively support a project even during its initial unstable stages:

1. **Cost to install an annotation tool**
   Creating a corpus involves a large number of hands engaging in the task of annotation. It is particularly the case for those unfamiliar with computers that merely installing an annotation tool can become a burden.

2. **Variation of data schemes for each annotation task**
   Past annotation tools have been developed with a specific annotation scheme in mind, making it unsuitable for other types of annotation. A multipurpose annotation tool must use a flexible data scheme that can incorporate various types of annotation, and must have an interface adaptable to various annotation tasks.

3. **Quality of the corpus**
   As previously mentioned, the initial phases of a project are often filled with adjustments to how a corpus will be annotated. Since typical annotators work individually while referring to a specification, this period can result in poor consistency. These errors affect the quality of a corpus which in turn affects the performance and subsequent evaluation of a system.

We introduce SLAT (Segment and Link-based Annotation Tool), in the next section. For tackling the first issue, we adopt a client/server architecture. We present annotation abstraction for resolving the second issue and discuss some already-developed annotation tools and their own implementations. Finally, we summarize our findings and briefly touch upon the third issue enumerated above.

4. SLAT

SLAT is a web-based annotation tool that employs a client/server architecture. With the ubiquitousness of the internet, this means that SLAT can be accessed almost anywhere; the only prerequisite for beginning annotation is having access to the URL via a browser. This also serves to reduce the cost and time of installation on an annotator's machine. The server-end of SLAT is composed of a computer running a database and a PHP-enabled web-server. The SLAT server stores all documents to be annotated, annotation information and customized user configurations.
In this section, we first propose an abstraction of annotations using segments and links, which allows SLAT to adapt to many different annotation tasks. We then address the interface issues, detailing the components of the current SLAT interface, and finally demonstrate how SLAT can be easily customized.

4.1. Abstraction of Annotations

To explore a universal data scheme applicable to various types of annotations, we discuss the abstraction of annotations using a simple POS annotation example shown in Figure 1. In this example, annotation is carried out by affixing POS and named entity tags to specific regions of text, called *segments*. Thus, "John" is annotated as N and N-PER and "New York" as N and N-LOC etc. Relations between segments are then identified, such as coreference or a certain semantic role. This is called *linking*. Using this abstraction, almost any annotation can be represented. SLAT adopts stand-off annotation, i.e. all annotated data is stored separately from the original data.

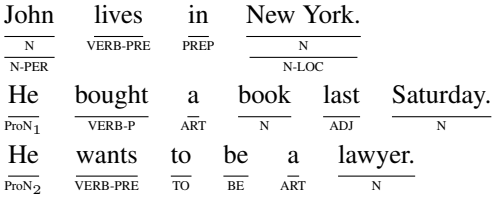

Figure 1: An example of POS annotation

4.1.1. Segments

When annotating a text, it is important to both indicate the particulars of a region as well as its relation to other parts of the text. A segment is indicated by marking the starting and ending offsets of a region. For representing this information, tags are inserted into the text. A fragment of the text can be multiple segments such as "John" and "New York" in Figure 1. Furthermore, segments can be nested and overlap, such as 'XXX YYY ZZZ'.

4.1.2. Links

As mentioned above, segments may have several types of relations to one another, e.g. "John" and "he" (coreference), or "bought" and "a book" (semantic role). All relations have at least two properties: transitivity and directionality. By combining these two properties, we can divide relations into four general groups:

1. **transitive and directed** E.g. "car"→"door"→"glass", *part-of* relations belong to this group. Temporal relations between events also belong to this group.
2. **transitive and undirected** Coordination and coreference, such as the relations between "John (N-PER)", "He (ProN$_1$)" and "He (ProN$_2$)"in Figure 1.
3. **non-transitive and directed** Semantic role labeling, e.g. the relation between "bought (VERB-P)" and "book (N)" belongs to this group.
4. **non-transitive and undirected** Relations in this group represent a special case only, and consist of only a pair.

4.2. Interface

SLAT's interface has been designed to allow for intuitive, visual annotation. It has two main panes in the center of the screen, as shown in Figure 2. The left pane, an editor pane, displays the text to be annotated while the right pane displays a list of all current segments and links. Annotating a segment is as easy as marking a region of text with the mouse.
The upper pane shows information of *selected* and *focused* segments. In Figure 2, "support systems" is *selected* and "adopt" is *focused*. The notion of the selected and forcued segments roughly corresponds to the source and destination segments of a link. A new link is annotated by regarding

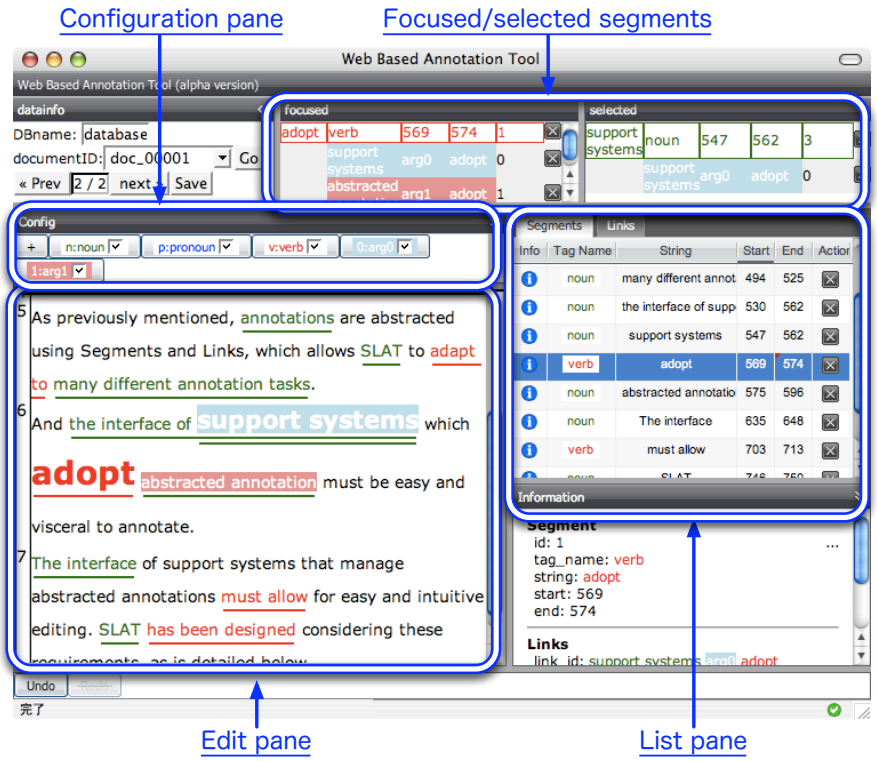

Figure 2: Snapshot of SLAT

selected segment as the destination and focused segment as the source of that link. And these segments have a difference in an operational respect. That is, the system allows users to move around focused segments by using arrow keys, while selected segments are determined by clicking the segments with the mouse. This operational distinction is useful for annotations where multiple links extend from a single segment, such as with predicate-argument annotation. The focusable segments are defined in the configuration as described below.

In the editor pane a segment is displayed as colored and underlined strings. Strings that are comprised of more than one annotation will have multiple underlines. A segment may be selected by clicking on an underlined region. When a segment is selected, links attached to that segment are presented by highlighting the counterpart segments with colors and underlines. In Figure 2, there are two links displayed: one is a link between "adopt" and "support systems" and the other is a link between "adopt" and "abstracted annotation".

The right list pane contains a table-view list of segments and links. Clicking a column header allows for sorting by properties such as *offsets*, *segment/link names* and so on. By clicking on a segment within this list, the left editor pane will scroll to display the selected item. Selecting a link item will identify both the destination and source segment within the editor pane.

4.2.1. Interface Design

Research shows that there are essentially two ways of representing relations: one using edges and the other table-based. In an interface that displays links using edges, identifying a link can become difficult if there is a large number of annotated links. However, a table-based interface has the obvious shortcoming of lacking good visual representation of source/destination. SLAT's interface was designed with both these points in mind. Relations with focused segments are highlighted by underlined and colored strings to avoid congestion in the editor pane. Highlighting can be toggled by a check-box in order to allow annotators concentrate on specific tags during annotation.

Many treebank projects represent the phrase structure of sentences using a tree representation. Phrase structures can be represented in terms of segments and links though the interface today is less than ideal for displaying its hierarchical structure. We designed our interface to be as adaptable to various annotation tasks as possible; segments and links are more versatile than tree representations, and in particular allow for overlapped segments which are troublesome to deal with using trees. That being said, a tree representation might be more suitable when annotating phrase structures and we have plans to incorporate another type of view pane for displaying trees, based on a user's configuration options.

4.3. Customization

SLAT allows users to customize tag-sets in two ways, (1) by using the GUI directly, and (2) by uploading a file containing tag-set definitions. Figure 3 shows a snapshot of the configuration interface, through which the user can create segment and link definitions.

A SLAT configuration can define different types of annota-

tions simultaneously e.g. coreference, predicate-argument structure and syntactic structure and whatsoever. Users can toggle the visibility of each tag by using the configuration pane just above the edit pane.

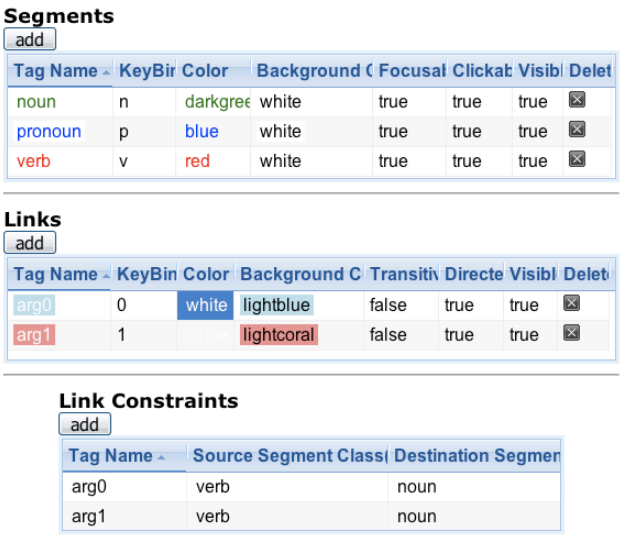

Figure 3: Snapshot of configuration pane

4.3.1. Segments

Tag-name defines the name of the segment, *key-bind* is an optional keyboard shortcut for creating a new segment while annotating a text; *color* and *background-color* define display colors, and *focusable* toggles whether or not a segment can be focused using arrow keys; *clickable* and *visible* each define whether a segment is selectable by clicking and if it is visible, respectfully. Sample definitions are shown in the upper table of Figure 3.

4.3.2. Links

Tag-name defines the name of the link, *key-bind* is the same as explained above, only for links; *transitivity* and *directed* define whether a link has each attribute as defined earlier. Based on these settings, SLAT can constrain the selection and pairing of source/destination tags. For allowing several source/destination combinations, they should all be defined here. Sample definitions are shown in the lower tables of Figure 3.

4.4. Other Features

When a segment is selected, the user's selection can be limited to only the focused/selected segment's tag name. This greatly decreases annotation errors related to accidentally selecting wrong segments. After annotation, a user may easily retrieve annotated text from SLAT via the web browser. SLAT supports undo/redo as well as customization and configuration of tag-sets. SLAT supports any language that can be encoded using UTF-8.

5. Summary and Future Work

With the goal of covering a broad range of annotation tasks, we have proposed a data scheme that is easier to understand and to use. In addition, we have introduced a tool SLAT, which implements many features, including several requirements designated especially important during the early stages of a project. SLAT's use of abstracted annotations, i.e. segments and links resolves many of the challenges presented in this paper, though there are still some issues to be solved.

Supporting annotators in assuring the consistency and quality of a corpus is a remaining challenge. The following is our reseach agenda for achieving this goal.

- Introduction of batch operations for keeping consistency
- Annotation help based on the workflow context
- Retrieval of cases similar to the current annotation target
- Visual methods for reporting errors
- Mining annotation data by multiple annotators to find annotation tips

6. Acknowledgment

This work is partially supported by the Grant-in-Aid for Scientific Research in Priority Areas JAPANESE CORPUS[8].

7. References

Stefanie Dipper, Michael Götze and Manfred Stede. (2004). Simple Annotation Tools for Complex Annotation Tasks: an Evaluation. In Proceedings of the LREC Workshop on XML-based Richly Annotated Corpora. pp.54-62. Lisbon. Portugal.

Coreference Task Definition. (1995). The sixth in a series of Message Understanding Conferences (MUC-6). http://cs.nyu.edu/cs/faculty/grishman/COtask21.book_5.html

Olga Babko-Malaya. (2005). PROPBANK ANNOTATION GUIDELINES. http://verbs.colorado.edu/~mpalmer/projects/ace/PBguidelines.pdf

Takahashi Tetsuro, Inui Kentaro. (2006). A multi-purpose corpus annotation tool: Tagrin. Proceedings of the 12th Annual Conference on Natural Language Processing. pp.228-231. Yokohama. Japan.

Christoph Müller. (2006). Representing and Accessing Multi-Level Annotations in MMAX2. In Proceedings of the 5th Workshop on NLP and XML (NLPXML-2006): Multi-dimensional Markup in Natural Language Processing. pp.73-76. Trento. Italy.

Constantin Orăsan. (2003). PALinkA: A highly customisable tool for discourse annotation. In Proceedings of the 4th SIGdial Workshop on Discourse and Dialogue. Sapporo. Japan.

Schmidt Thomas. (2001). The transcription system EXMARaLDA: An application of the annotation graph formalism as the Basis of a Database of Multilingual Spoken Discourse. In Proceedings of the IRCS Workshop On Linguistic Databases, 11-13. Philadelphia. USA.

[8] http://www.tokuteicorpus.jp