Annotation Tools Based on the Annotation Graph API

Steven Bird, Kazuaki Maeda, Xiaoyi Ma and Haejoong Lee
Linguistic Data Consortium, University of Pennsylvania
3615 Market Street, Suite 200, Philadelphia, PA 19104-2608, USA
{sb,maeda,xma,haejoong}@ldc.upenn.edu

Abstract

Annotation graphs provide an efficient and expressive data model for linguistic annotations of time-series data. This paper reports progress on a complete open-source software infrastructure supporting the rapid development of tools for transcribing and annotating time-series data. This general-purpose infrastructure uses annotation graphs as the underlying model, and allows developers to quickly create special-purpose annotation tools using common components. An application programming interface, an I/O library, and graphical user interfaces are described. Our experience has shown us that it is a straightforward task to create new special-purpose annotation tools based on this general-purpose infrastructure.

1 Introduction

In the past, standardized file formats and coding practices have greatly facilitated data sharing and software reuse. Yet it has so far proved impossible to work out universally agreed formats and codes for linguistic annotation. We contend that this is a vain hope, and that the interests of sharing and reuse are better served by agreeing on the data models and interfaces.

Annotation graphs (AGs) provide an efficient and expressive data model for linguistic annotations of time-series data (Bird and Liberman, 2001). Recently, the LDC has been developing a complete software infrastructure supporting the rapid development of tools for transcribing and annotating time-series data, in cooperation with NIST and MITRE as part of the ATLAS project, and with the developers of other widely used annotation systems, Transcriber and Emu (Bird et al., 2000; Barras et al., 2001; Cassidy and Harrington, 2001).

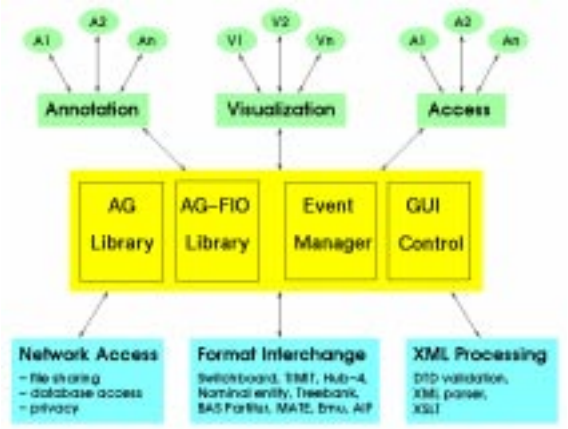

Figure 1: Architecture for Annotation Systems

The infrastructure is being used in the development of a series of annotation tools at the Linguistic Data Consortium. Two tools are shown in the paper: one for dialogue annotation and one for interlinear transcription. In both cases, the transcriptions are time-aligned to a digital audio signal.

This paper will cover the following points: the application programming interfaces for manipulating annotation graph data and importing data from other formats; the model of inter-component

communication which permits easy reuse of software components; and the design of the graphical user interfaces.

2 Architecture

2.1 General architecture

Figure 1 shows the architecture of the tools currently being developed. Annotation tools, such as the ones discussed below, must provide graphical user interface components for signal visualization and annotation. The communication between components is handled through an extensible event language. An application programming interface for annotation graphs has been developed to support well-formed operations on annotation graphs. This permits applications to abstract away from file format issues, and deal with annotations purely at the logical level.

2.2 The annotation graph API

The application programming interface provides access to internal objects (signals, anchors, annotations etc) using identifiers, represented as formatted strings. For example, an AG identifier is qualified with an AGSet identifier: `AGSetId:AGId`. Annotations and anchors are doubly qualified: `AGSetId:AGId:AnnotationId`, `AGSetId:AGId:AnchorId`. Thus, the identifier encodes the unique membership of an object in the containing objects.

We demonstrate the behavior of the API with a series of simple examples. Suppose we have already constructed an AG and now wish to create a new anchor. We might have the following API call:

```
CreateAnchor("agSet12:ag5", 15.234, "sec");
```

This call would construct a new anchor object and return its identifier: `agSet12:ag5:anchor34`. Alternatively, if we already have an anchor identifier that we wish to use for the new anchor (e.g. because we are reading previously created annotation data from a file and do not wish to assign new identifiers), then we could have the following API call:

```
CreateAnchor("agset12:ag5:anchor34",
             15.234, "sec");
```

This call will return `agset12:ag5:anchor34`.

Once a pair of anchors have been created it is possible to create an annotation which spans them:

```
CreateAnnotation("agSet12:ag5",
                 "agSet12:ag5:anchor34",
                 "agSet12:ag5:anchor35",
                 "phonetic" );
```

This call will construct an annotation object and return an identifier for it, e.g. `agSet12:ag5:annotation41`. We can now add features to this annotation:

```
SetFeature("agSet12:ag5:annotation41",
           "date", "1999-07-02" );
```

The implementation maintains indexes on all the features, and also on the temporal information and graph structure, permitting efficient search using a family of functions such as:

```
GetAnnotationSetByFeature(
  "agSet12:ag5", "date", "1999-07-02");
```

2.3 A file I/O library

A file I/O library (AG-FIO) supports input and output of AG data to existing formats. Formats currently supported by the AG-FIO library include the TIMIT, BU, Treebank, AIF (ATLAS Interchange Format), Switchboard and BAS Partitur formats. In time, the library will handle all widely-used signal annotation formats.

2.4 Inter-component communication

Figure 2 shows the structure of an annotation tool in terms of components and their communication.

The main program is typically a small script which sets up the widgets and provides callback functions to handle widget events. In this example there are four other components which

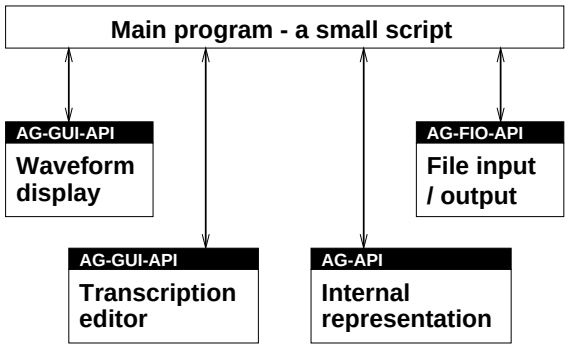

Figure 2: The Structure of an Annotation Tool

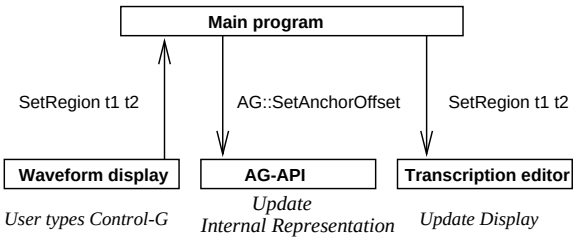

Figure 3: Inter-component Communication

are reused by several annotation tools. The AG and AG-FIO components have already been described. The waveform display component (of which there may be multiple instances) receives instructions to pan and zoom, to play a segment of audio data, and so on. The transcription editor is an annotation component which is specialized for a particular coding task. Most tool customization is accomplished by substituting for this component.

Both GUI components and the main program support a common API for transmitting and receiving events. For example, GUI components have a notion of a "current region" — the timespan which is currently in focus. A waveform component can change an annotation component's idea of the current region by sending a `SetRegion` event (Figure 3). The same event can also be used in the reverse direction. The main program routes the events between GUI components, calling the annotation graph API to update the internal representation as needed. With this communication mechanism, it is straightforward to add new commands, specific to the annotation task.

2.5 Reuse of software components

The architecture described in this paper allows rapid development of special-purpose annotation tools using common components. In particular, our model of inter-component communication facilitates reuse of software components. The annotation tools described in the next section are not intended for general purpose annotation/transcription tasks; the goal is not to create an "emacs for linguistic annotation". Instead, they are special-purpose tools based on the general purpose infrastructure. These GUI components can be modified or replaced when building new special-purpose tools.

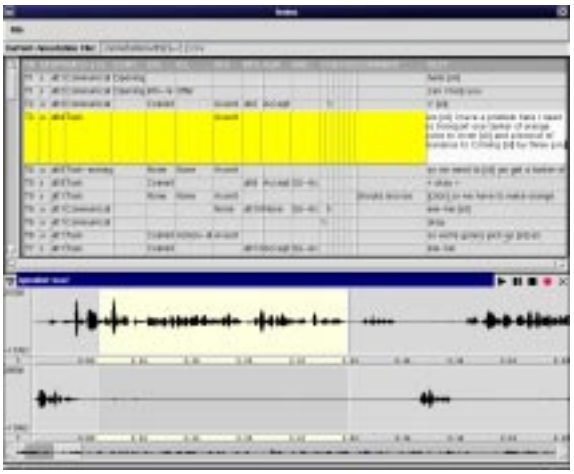

Figure 4: Dialogue Annotation Tool for the TRAINS/DAMSL Corpus

3 Graphical User Interfaces

3.1 A spreadsheet component

Dialogue annotation typically consists of assigning a field-structured record to each utterance in each speaker turn. A key challenge is to handle overlapping turns and back-channel cues without disrupting the structure of individual speaker contributions. The tool side-steps these problems by permitting utterances to be independently aligned to a (multi-channel) recording. The records are displayed in a spreadsheet; clicking on a row of the spreadsheet causes the corresponding extent of audio signal to be highlighted. As an extended recording is played back, annotated sections are highlighted, in both the waveform and spreadsheet displays.

Figure 4 shows the tool with a section of the TRAINS/DAMSL corpus (Jurafsky et al., 1997). Note that the highlighted segment in the audio channel corresponds to the highlighted annotation in the spreadsheet.

3.2 An interlinear transcription component

Interlinear text is a kind of text in which each word is annotated with phonological, morphological and syntactic information (displayed under the word) and each sentence is annotated with a free translation. Our tool

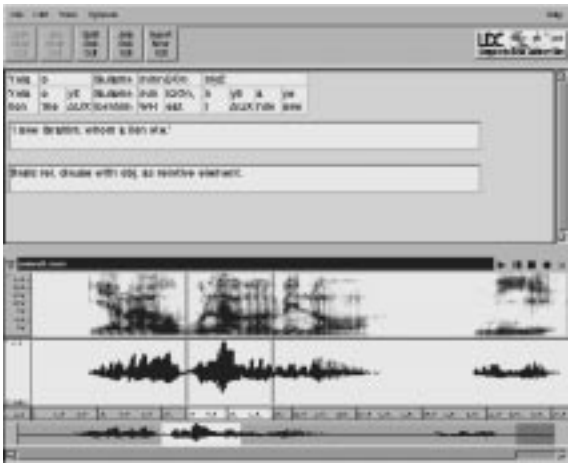

Figure 5: Interlinear Transcription Tool

permits interlinear transcription aligned to a primary audio signal, for greater accuracy and accountability. Whole words and sub-parts of words can be easily aligned with the audio. Clicking on a piece of the annotation causes the corresponding extent of audio signal to be highlighted. As an extended recording is played back, annotated sections are highlighted (both waveform and interlinear text displays).

The screenshot in Figure 5 shows the tool with some interlinear text from Mawu (a Manding language of the Ivory Coast, West Africa).

3.3 A waveform display component

The tools described above utilize WaveSurfer and Snack (Sjölander, 2000; Sjölander and Beskow, 2000). We have developed a plug-in for WaveSurfer to support the inter-component communication described in this paper.

4 Available Software and Future Work

The Annotation Graph Toolkit, version 1.0, contains a complete implementation of the annotation graph model, import filters for several formats, loading/storing data to an annotation server (MySQL), application programming interfaces in C++ and Tcl/tk, and example annotation tools for dialogue, ethology and interlinear text. The supported formats are: xlabel, TIMIT, BAS Partitur, Penn Treebank, Switchboard, LDC Callhome, CSV and AIF level 0. All software is distributed under an open source license, and is available from http://www.ldc.upenn.edu/AG/.

Future work will provide Python and Perl interfaces, more supported formats, a query language and interpreter, a multichannel transcription tool, and a client/server model.

5 Conclusion

This paper has described a comprehensive infrastructure for developing annotation tools based on annotation graphs. Our experience has shown us that it is a simple matter to construct new special-purpose annotation tools using high-level software components. The tools can be quickly created and deployed, and replaced by new versions as annotation tasks evolve.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant Nos. 9978056, 9980009, and 9983258.

References

Claude Barras, Edouard Geoffrois, Zhibiao Wu, and Mark Liberman. 2001. Transcriber: development and use of a tool for assisting speech corpora production. *Speech Communication*, 33:5–22.

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

Steven Bird, David Day, John Garofolo, John Henderson, Chris Laprun, and Mark Liberman. 2000. ATLAS: A flexible and extensible architecture for linguistic annotation. In *Proceedings of the Second International Conference on Language Resources and Evaluation*. Paris: European Language Resources Association.

Steve Cassidy and Jonathan Harrington. 2001. Multi-level annotation of speech: An overview of the emu speech database management system. *Speech Communication*, 33:61–77.

Daniel Jurafsky, Elizabeth Shriberg, and Debra Biasca. 1997. Switchboard SWBD-DAMSL Labeling Project Coder's Manual, Draft 13. Technical Report 97-02, University of Colorado Institute of Cognitive Science. [stripe.colorado.edu/~jurafsky/manual.august1.html].

Kåre Sjölander and Jonas Beskow. 2000. Wavesurfer – an open source speech tool. In *Proceedings of the 6th International Conference on Spoken Language Processing*. `http://www.speech.kth.se/wavesurfer/`.

Kåre Sjölander. 2000. The Snack sound toolkit. `http://www.speech.kth.se/snack/`.