The Story Workbench: An Extensible Semi-Automatic Text Annotation Tool

Mark Alan Finlayson
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
32 Vassar Street, Room 32-258
Cambridge, MA 02138 USA
markaf@mit.edu

Abstract

Text annotations are of great use to researchers in the language sciences, and much effort has been invested in creating annotated corpora for an wide variety of purposes. Unfortunately, software support for these corpora tends to be quite limited: it is usually *ad-hoc*, poorly designed and documented, or not released for public use. I describe an annotation tool, the Story Workbench, which provides a generic platform for text annotation. It is free, open-source, cross-platform, and user friendly. It provides a number of common text annotation operations, including representations (e.g., tokens, sentences, parts of speech), functions (e.g., generation of initial annotations by algorithm, checking annotation validity by rule, fully manual manipulation of annotations) and tools (e.g., distributing texts to annotators via version control, merging doubly-annotated texts into a single file). The tool is extensible at many different levels, admitting new representations, algorithm, and tools. I enumerate ten important features and illustrate how they support the annotation process at three levels: (1) annotation of individual texts by a single annotator, (2) double-annotation of texts by two annotators and an adjudicator, and (3) annotation scheme development. The Story Workbench is scheduled for public release in March 2012.

Text annotations are of great use to researchers in the language sciences: a large fraction of that work relies on annotated data to build, train, or test their systems. Good examples are the Penn Treebank, which catalyzed work in developing statistical syntactic parsers, and PropBank, which did the same for semantic role labeling. It is not an exaggeration to say that annotated corpora are a central resource for these fields, and are only growing in importance. Work on narrative shares many of the same problems, and as a consequence has much to gain from advances in language annotation tools and techniques.

Despite the importance of annotated data, there remains a missing link: software support is not given nearly the same amount of attention as the annotations themselves. Researchers usually release only the data; if they release any tools at all, they are usually *ad-hoc*, poorly designed and documented, or just not released for public use. Tools do not build on one another.

The language sciences need to move to a standard where, if annotated data is released, software for accessing and creating the data are released as a matter of course. Researchers should prepare for it, reviewers should demand it, and readers should expect it. One way of facilitating this is to lower the barrier for creating tools. Many of the phases of the annotation cycle are the same no matter what sort of annotation you are doing - a freely available tool, or suite of tools, to support these phases would go a long way.

I describe the Story Workbench (Finlayson 2008), a major step toward just such a tool suite. The Story Workbench is free, open-source, extensible, cross-platform, and user friendly. It is a working piece of software, having been in beta testing for over three years, with a public release scheduled for March 2012. It has been used by more than 12 annotators to annotate over 100k words across 17 representations. Two corpora have been created so far with it: the UMIREC corpus (Hervas and Finlayson 2010) comprising 25k words of news and folktales annotated for referring expression structure, and 18k words of Russian folktales annotated in all 17 different representations.

The Story Workbench is especially interesting to researchers working on narrative. Understanding a narrative requires not just one representation, not just two, but a dozen or more. The Story Workbench was created specifically to overcome that problem, but is now finding application beyond the realm of narrative research. In particular, in the next section I describe three phases of the annotation process; many, if not most, annotation studies move through these phases. In the next section I enumerate some of the more important features of the Story Workbench, and show how these support the phases.

Three Loops of the Annotation Process

Conceptually, the process of producing a gold-standard annotated corpus can be split into at least three nested loops. In the widest, top-most loop the researchers design and vet the annotation scheme and annotation tool; embedded therein is the middle loop, where annotation teams produce gold-annotated texts; embedded within that is the loop of the individual annotator working on individual texts. These nested loops are illustrated in Figure 1.

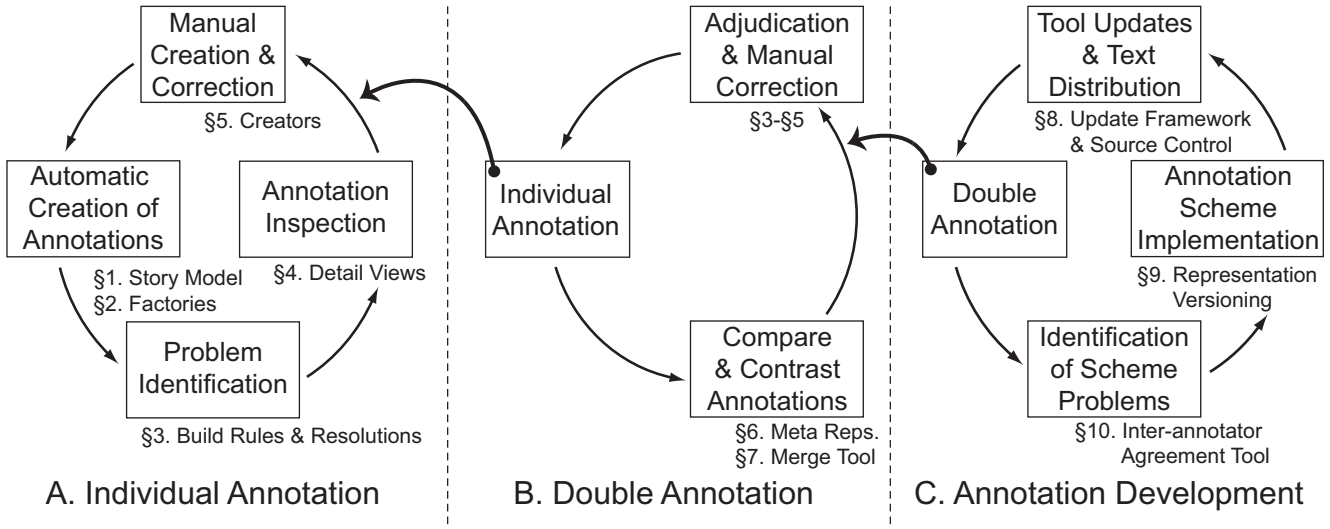

Figure 1: Three Loops of the Annotation Process. Letters and numbers correspond to sections in the text. Underneath each stage are the Story Workbench features that support that stage.

A. Individual Annotation

Individual annotation occurs when an annotator annotates a text in a particular annotation scheme. This is the most fundamental part of the annotation process: it is in this tight loop that most of the effort in annotation is expended, and therefore is where most of the Story Workbench support is focused. Stages of this loop include automatic production of initial annotations, finding problems with those annotations by either the annotator or the tool, and then manual creation, correction or elaboration of the annotations by the annotator. This loop is repeated for each text.

B. Double Annotation

Double annotation[1] is the norm when annotating complex representations or large amounts of text. It is done to guard against errors and to obtain measures of annotation quality. Although differences between annotations may be caused by simple mistakes, it is often the case that, for complex representations, the difference is a legitimate ambiguity and must be adjudicated by a third party. This adjudicator is responsible for comparing and contrasting the two annotation sets, and correcting the discrepancies. Operationally, this loop involves distributing the texts to the individual annotators, comparing the resulting annotations, and discussing and correcting the differences. This loop repeats itself, either in parallel or in serial, for each text being annotated.

C. Annotation Development

The top-most loop is only necessary when one is developing a new representation, a new tool, or evaluating which texts to annotate. This loop involves developing the annotation scheme and annotation software, distributing the texts to the annotation teams (or individual annotators, if no double annotation), and finally evaluating the results to determine what changes should be made to the representation or to the software.

[1] I call it *double* annotation for ease of exposition; of course, in the general case, one may have as many annotators as one likes.

Story Workbench Functionality

As indicated in the figure, the Story Workbench supports the three annotation loops with specific features, functions, and tools. Each section here is numbered to correspond with the figure.

1. Story Model

The Story Workbench distinguishes between *representations* and *descriptions*. A representation is the programmatic format of the annotation, while descriptions are actual annotations in that format. Within the Story Workbench, a representation is defined first by a data structure that specifies all the information possibly contained in a single annotation in that representation, and second by a singleton lifecycle object that controls serialization and deserialization of descriptions for that representation, and defines on which other representations that representation depends. A description is a concrete instance of data that conforms to the representation specification and is linked to the representation lifecycle object.

The core of the Story Workbench is the story model, which is the data structure that describes a whole text and all of its annotations. Most importantly it comprises maps from representation lifecycle objects to (1) sets of descriptions in that representation; (2) configuration information for that representation; and (3) an annotation factory for that representation (see §2). The model has a primary, root representation, called the *character* representation, whose single description contains the actual characters for the text. All descriptions in the model are indexed to character offsets in the this singleton character description.

The two most important features of representations are that they are (1) layered, in that more complex representations can build upon simpler representations, and (2) extensible, in that new representations may be added. There are currently more than 17 representations implemented in the Story Workbench, with more on the way. They include:

1. Tokens - the constituent characters of each token
2. Multi-word Expressions - words with multiple tokens
3. Sentences - the constituent tokens of each sentence
4. Part of Speech Tags - a tag for each token
5. Lemmas - root form for each inflected word
6. Word Senses - a dictionary sense for each word
7. Context-Free Grammar Parse - one per sentence
8. Referring Expressions - expressions that refer
9. Referent Attributes - unchanging properties of referents
10. Co-reference Bundles - referring expressions that co-refer
11. Time Expressions - cf. TimeML (Pustejovsky et al. 2003)
12. Events - TimeML happenings and states
13. Temporal Relationships - TimeML time order
14. Referent Links - static non-temporal relationships
15. Semantic Roles - PropBank verbal arguments
16. Proppian Functions - as identified in (Propp 1968)
17. Proppian Archetypes - Propp's *dramatis personae*

2. Factories

One of the most important functions of the workbench is the ability to provide automatic annotation. Automatic annotation is triggered by any change to the text or existing annotations, and are created by *annotation factories*, each of which is specific to a representation. An example of an annotation factory is a part-of-speech tagger (associated, naturally, with the part of speech representation) which tags new tokens as they are created; a second example is a a semantic role tagger, that tags new verbs with semantic roles.

Factories are also responsible for keeping the model consistent: for example, if a character is inserted at the beginning of the text, all the annotations in the model, which are indexed to character offsets, must be shifted forward by one. This is taken care of automatically by the factories. Factories are extensible in that it is easy to add new factories to existing representations.

3. Build Rules & Resolutions

Each representation is associated with a number of *build rules*. Build rules are provided by the representation implementer; they check for known or suspected annotation errors. Build rules are run when the text is saved, and the problems they find are displayed in the editor (highlighted for easy identification) and in separate views. In many of the annotation tasks for which the Story Workbench has so far been used, the task is set up so that the text starts with one problem per potential annotation. The annotator's job is then to eliminate all the errors, which, when done, indicates the annotation in that representation is finished.

Problems found by build rules can be associated with problem resolutions. These resolutions are mini programs that, when run, fix in a specific way the problem with which they are associated. These resolutions are shown to the annotator when any graphical representation of the problem is clicked. As an example, for the multi-word expression representation, it is considered an error to have two multi-words that share a token. Such an occurrence is marked by a red X in the problems list, and the offending token or tokens are underlined in red in the editor. When the annotator clicks on either the underline or the X, a dropdown list appears with four resolution options: delete the first multi-word, delete the second, remove the token from the first multi-word, or remove the token from the second. The annotator may choose any one of those resolutions, or go about fixing the problem in his own way. Like most everything else, build rules and resolutions are extensible.

4. Detail Views

Also associated with each representation is a detail view, which provides detailed information about the annotations for that representation. Detail views can provide information such as the internal structure of annotations or a detailed breakdown of field values with explanations. As an example, the detail view for the Semantic Role representation shows a list of all the verbs in the text; each verb's semantic arguments are listed underneath, and clicking on the verb in the details view, or an argument to the verb, highlights the associated text in the editor. The detail viewer infrastructure is extensible: additional detail viewers can be added by third parties for existing representations.

One can also include in this category a number of specialized viewers and information transmission mechanisms which make the Story Workbench a user friendly environment. The *text hover* infrastructure allows representation implementers to show information in a tooltip when the mouse hovers over a specific section of the text. For example, when the mouse hovers over a word, a tooltip displays the part of speech tag and the root form (if the word is inflected). The workbench also allows specialized viewers outside of the detail view - a good example of this is the parse tree viewer, which, when the cursor is placed on a sentence with a CFG parse, the parse is displayed in graphical tree form.

5. Creators

The heavy lifting of annotation happens in the creator view, where the annotator can create and modify individual annotations. Each creator view is customized to its associated representation. For example, in the TimeML event representation, the creator view allows (1) identification of tokens involved in the event, (2) identification of head tokens, (3) marking the part of speech, tense, and aspect of verbal events, (3) identification of polarity tokens, and associated polarity flag, (4) identification of tokens indicating cardinality, and the number, and (5) identification of tokens indicating modality.

6. Meta Representations

In addition to the main representations that encode the syntactic and semantic information of the text, there are so-called *meta* representations that track information common

to all descriptions. Meta representations are also extensible. I identify four of the most important below. Perhaps the most useful of these is the Note meta representation, which allows annotators and adjudicators to record their thoughts and observations for later consideration.

1. Notes - arbitrary text attached to a description to record observations, thoughts, ideas; see §7 & §9
2. Origin - automatically added upon description creation to record the identity of the creator; see §5
3. Timing - automatically added upon description creation to record creation and editing times; see §5
4. Check - marks a problem as resolved or checked relative to a particular description; see §3

7. Merge Tool

A key function for double annotation and adjudication is the ability to compare and contrast annotations done by two different annotators. The Story Workbench provides a general tool for merging two sets of annotations into a single text. Each representation implementor provides a small amount of code that allows the workbench to decide if two annotators are equivalent - this is provided in the implementation of the representation data structure (§1). The tool then allows an adjudicator to decide which representations to merge, which should already be identical, and which should be ignored, and enforces constraints imposed by the representation hierarchy. The resulting annotated text can then be edited and corrected in the normal way (§3-5); the representation build rules may be defined so that, where possible, conflicting annotations from different annotators show up as errors to be corrected.

8. Update Framework & Source Control

Integrated with the workbench is a general means for interfacing with source control systems. Such systems, such as Subversion or CVS, can be used to great effect in annotation studies. Let us take Subversion as a concrete example. Researchers upload texts to be annotated to the Subversion repository. Annotators then *check out* the texts and do their annotation. The workbench tracks when they have made changes, and displays the presence of those changes to the annotator; he then knows to *check in* his changes to the repository, where they are then available to the adjudicator or the researchers.

Also included in the workbench is a sophisticated update functionality. This can be used to to fix bugs or add new features without requiring a re-install of the tool.

9. Representation Versioning

Developing an annotation scheme is facilitated by a versioning system for representation implementations. By way of example, imagine a researcher designs a representation that allows a single tag to be applied to any token. He implements the scheme, gives it a version number (say, 1.0), and has his annotators annotate some text with that representation. After considering the results (e.g., inter-annotator agreement scores, see §10), he decides there should be a second tag to supplement the first. He then modifies the implementing code, increments the version (to 2.0), and specifies in the versioning system that when an v1.0 annotation is transformed into a v2.0 annotation, the new fields should be filled in with a tag called UNKNOWN. He writes a build rule that marks all annotations with an UNKNOWN tag as an error. He uses the update feature to distribute the new code to the annotators. When the annotators re-open their old texts, the Story Workbench automatically transforms the old annotations into the new, and they are presented with new errors that, when corrected, will naturally lead them to fill in the second field.

10. Inter-annotator Agreement Tool

Finally, a general facility is provided for measuring inter-annotator agreements. The facility allows for automatic measurement of F-scores for all representations. A facility is also provided for representation implementers to provide code that allows annotator agreements to be measured in terms of the kappa score.

Contributions

I have described the Story Workbench annotation tool, a general text annotation platform that is free, open-source, extensible, cross-platform, and user friendly. It addresses a long-standing need in the language sciences, that of a reliable tool that provides a number of commonly needed functions. I illustrated how various features of the workbench support three levels of the annotation process.

Acknowledgments

This work was supported in part by the Defense Advanced Research Projects Agency, under contract FA8750-10-1-0076, and in part by the Office of Naval Research under award number N00014-09-1-0597. Any opinions, findings, and conclusions or recommendations expressed here are those of the author and do not necessarily reflect the views of the Office of Naval Research. Many thanks to Patrick H. Winston for his helpful comments.

References

Finlayson, M. A. 2008. Collecting semantics in the wild: The story workbench. In *Naturally Inspired Artificial Intelligence, Technical Report FS-08-06, Papers from the AAAI Fall Symposium*, 46–53. Menlo Park, CA: AAAI Press.

Hervas, R., and Finlayson, M. A. 2010. The prevalence of descriptive referring expressions in news and narrative. In *Proceedings of the 48th ACL*, 49–54.

Propp, V. 1968. *Morphology of the Folktale*. Austin: University of Texas Press.

Pustejovsky, J.; Castano, J.; Ingria, R.; Sauri, R.; Gaizauskas, R.; Setzer, A.; and Katz, G. 2003. TimeML: Robust specification of event and temporal expressions in text. In *Proceedings of IWCS-5, the Fifth International Workshop on Computational Semantics*.