
SBAT: A Simple Browser-Based Annotation Tool with Interactive Visualization and GitHub Support

Author

Jia Sheng

jia.sheng@student.uni-tuebingen.de

Supervisor

Çagri Çöltekin

ccoltekin@sfs.uni-tuebingen.de

A thesis submitted in partial fulfilment
of the requirements for the degree of

Bachelor of Arts

in

International Studies in Computational Linguistics

Seminar für Sprachwissenschaft
Eberhard Karls Universität Tübingen

September 2024

Abstract

In computational linguistics, annotating linguistic data is a crucial step for many studies, and many tools can be used to facilitate the creation of annotated data. This thesis introduces SBAT, a lightweight and easy-to-use annotation tool. It supports annotation tasks that require attaching labels to text spans, such as POS tagging or labeling for named entity recognition. Among the various applications previously developed for the purpose of text annotation, many require a dedicated server or a local installation, which complicates the usage. On the contrary, SBAT is designed to be a browser-based annotation tool that is easy to access and simple to use. It is built on the BRAT annotation editor, but is completely server-free, while preserving most of the key functionalities. Additionally, it provides support for integration with GitHub's file management system, which facilitates version control and user management in annotation projects.

Contents

1	Introduction	1
2	Related Work	3
3	Features	5
3.1	Interactive User Interface	5
3.2	GitHub Support	6
3.3	Browser-Based Application	8
3.4	Flexible Configuration	8
4	Implementation	10
4.1	Annotation UI	10
4.2	Browser-Based Construction	10
4.3	Usage of GitHub API	11
5	Conclusion	12

List of Figures

1	Pop-Up Menu for New Annotation	6
2	Visualization of Annotations	7
3	Drop-Down List for File Selection	9

1 Introduction

Annotated data serves as the foundation for many linguistic studies, based on which it is possible to examine patterns, study linguistic phenomena, train statistical models or test linguistic theories. And among all types of data, text annotation is the most prevalent. Text annotation refers to the labeling or classification of text data, which is then used for various purposes. For instance, the text data with named entity tags can be used to train a NER (named entity recognition) model for Natural Language Processing. But annotating is not limited to named entities, tagging semantic roles, dependencies, parts of speech, sentiments, relations and so on are all considered text annotations in common sense. Moreover, although automatic annotations by the computer achieved good performance in some cases, most annotation tasks still require the annotation or validation of human annotators, highlighting the necessity of annotation tools.

For manual or computer-assisted creations of annotated data, a variety of applications have been developed over the years, each with a specific focus or for a specific annotation scheme. Those tools can recognize spans of characters and their links, and enable the user to associate labels to those spans and links [Ide, 2017]. Notable examples include BRAT for text span annotation and relation annotation, UD Annotatrix for Universal Dependencies annotation, and INCEpTION for Semantic annotation. Those tools not only provide an editor-like user interface for annotating operations, but many also enable suggestions generated by machine-learning models to speed up the creation of annotations and improve accuracy. Designed for usage in annotation projects, most of them provide user management functionalities through user registration and role assignment, in order to facilitate the collaboration of annotators in the annotation process. In terms of usage, many of those tools are dependent on Java or Python libraries, and therefore require installation of Java or Python prior to use. Some of them have a web platform, while others are desktop applications.

While those existing annotation tools provide various solutions across different schemes for the conduction of annotation projects, they often require complicated configuration on either local devices or the server side [Yang et al., 2018], which is not ideal for smaller annotation projects with limited time and low budget. Furthermore, most of them do not have an efficient version control system, which means keeping track of changes or reverting to previous versions is either impossible or requires a lot of effort. Taking those limitations into account, SBAT¹ is designed to be a browser-only, lightweight text span annotation tool, with integrated support for reading and writing files from and to GitHub repositories. By making SBAT browser-based, its only software dependency is

¹GitHub repository: <https://github.com/jiasheng1100/SBAT>

a web browser, which is already available on every modern computer. And its support for GitHub repository operations opens the door to the adoption of GitHub’s version management functionalities in annotation projects.

The rest of this thesis is therefore divided into four sections, all dedicated to the introduction of SBAT and its comparison with the current popular annotation tools. The second chapter deals with the relevant works in the field, presenting a few representative annotation tools developed over the years, along with an overview of their functionalities, advantages and limitations. The third chapter demonstrates the main features of SBAT, such as its interactive user interface and its support for Github, supplemented with detailed instructions and examples. Following that, the fourth chapter talks about the technical details of implementation in the development of SBAT. Finally, this thesis ends with concluding remarks about SBAT, illustrating its strengths and limitations.

2 Related Work

Among the annotation tools, BRAT [Stenetorp et al., 2012] is a powerful scheme-neutral annotation tool with an intuitive user interface. It can be used for tasks including but not limited to POS tagging, named entity recognition, semantic role labeling, dependency and verb-frame annotations, and enables user-defined constraints-checking. Moreover, it provides a high-quality visualization of annotations through its UI components implemented in XHTML, Scalable Vector Graphics (SVG) and JavaScript. Last but not least, BRAT integrates an ML-based semantic class disambiguator that outputs multiple annotations with their possibility estimates, which has been proven to increase the efficiency of annotators in several annotation projects.

Despite the success of BRAT, it is not the simplest tool to start annotating with. A CGI-capable server is required by its installation, in order to combine all user modifications in real-time with the stored data. That is where `brat-frontend-editor` [Hébert-Legault, 2017], a forked project of Brat, differentiates itself from the original project. `brat-frontend-editor` is a standalone browser version of Brat that keeps some of its essential editing functions but removes its server-side code. It can be imported as a module in vanilla JavaScript, Angular or React. This project is since 2018 no longer maintained, so many dependencies have broken, and it no longer keeps up with the new updates of the original Brat. However, its transformation of server-side operations into browser-side operations serves as an important reference in the development of SBAT.

While BRAT serves as a general-purpose annotation tool, more recent tools target specific annotation schemes. UD Annotatrix [Tyers et al., 2017], for example, is specifically designed for annotations of universal dependencies. It accepts several input formats including plain text and CoNNL-U, and offers graphical editing functions. Unlike BRAT, it offers a stand-alone module written in JavaScript with locally saved dependencies, which stores imported corpora in `localStorage` and allows offline usage. Otherwise, there is also a server module, which saves corpora on the server when enabled.

Another domain-specific annotation platform is INCEpTION [Klie et al., 2018], which specializes in semantic annotation. One of its characteristics is the extensive adoption of machine-learning models to assist the annotation process. It not only utilizes recommenders based on trained models to provide users with suggestions for possible labels, but also provides an active learning mode where user feedback can be used to further improve the quality of those suggestions. It also has a comprehensive user management system, where the admin can create multiple user accounts for the project and assign them different roles. In terms of the software itself, INCEpTION is a Java application with

Spring Boot backend and a web user interface, so its usage requires the installation of Java as well as the software on the local device.

Lastly, the previous version of SBAT [Weirich, 2022] serves as one of the foundations for the current version. The previous version shares the same goal: to create a simple but functional annotation tool that operates entirely in the browser, without any server-side configuration. It is implemented using HTML and JavaScript, and can be used to attach user-defined labels to whole texts. It also allows to import JSON or txt files from the local device or export annotations in a JSON file. Building on that, the current version addresses the limitations of the previous one by enabling functionalities such as span annotations, editor UI, visualization, and GitHub support.

3 Features

3.1 Interactive User Interface

Built upon BRAT, the user interface of SBAT is responsive, interactive and intuitive. It enables all basic operations in the editor, including creating, editing, or deleting annotations, with dragging and clicking operations of the mouse. The size and display of the editor can also be manually toggled by the option button.

When SBAT is started in the browser for the first time, an interactive editor with the default example file as well as the user authentication slot will be displayed. The editor view can be utilized by new users to experiment with the annotation operations and configurations without the need to log in. If the user has already logged in previously using the same browser, however, the log-in step will be skipped. Then, depending on whether the information about file paths has been provided in the config file, the user might be directly redirected to the editing page of an annotation file or will be prompted to select the file from a drop-down list. All those steps are guided with clear instructions in the user interface.

In the editor, to create a new annotation, one can simply double-click or drag to select a span of characters with his mouse. A pop-up window will then appear [Figure 1], with a list of label options that the user has pre-defined in the config file. The user can then select the corresponding label in the pop-up menu, click the "OK" button, and the selected label is added as an annotation. In the editor, all annotated spans are highlighted with different colors, and the appended labels are visualized in tags above the spans [Figure 2]. If there is any unclarity, users can also add their notes to the span, which will become visible with a mouse hover.

Likewise, with a double click on a previously added annotation in the editor, a pop-up window will appear that allows the edit or deletion of this annotation. For editing, one can select a different label or edit the notes, and then hit the "OK" button. The changes will then be saved and displayed in the visualization. And for deletion of an annotation, clicking the "Delete" button will do. There is also a button for adding a fragment to an existing annotation, which, once clicked, will allow the user to select another text span and add it as a fragment. The corresponding fragments will then share the same label, and can be moved or deleted.

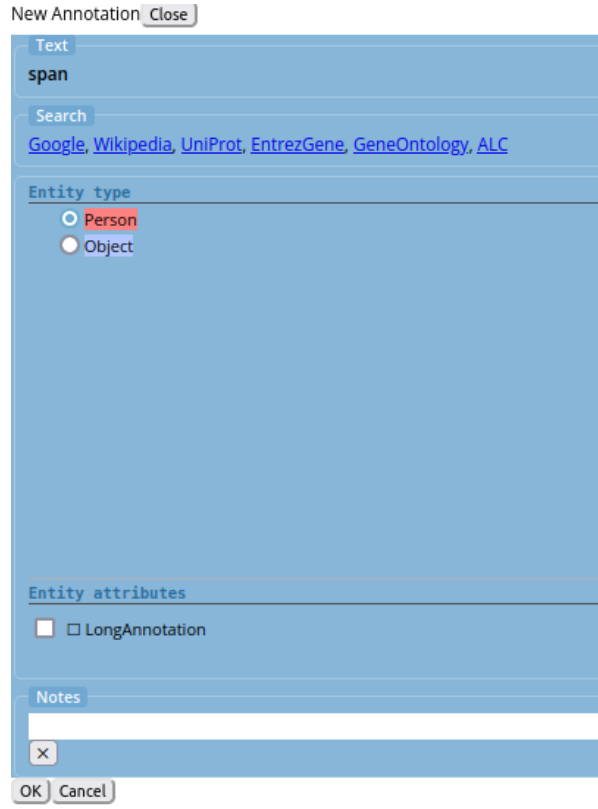


Figure 1: Pop-Up Menu for New Annotation

This pop-up window appears when double-clicking or dragging to select a span of text in the editor. The displayed options are defined in the config file, in this case two entity types, "Person" and "Object".

3.2 GitHub Support

SBAT enables users to import files from or export annotations to the GitHub repositories to which they have access, which means both the text content to be annotated as well as the annotations can be stored on GitHub to track or restore any changes. By incorporating those functionalities, one can not only conveniently document the versions of annotations, but also efficiently manage the tasks of members in a larger annotation project.

To make use of these functions, one needs to first pass a user authentication, which is done by entering his GitHub Personal Access Token in the SBAT user interface. After successful authentication, the user will be prompted to select the branch and the filename – if not already provided in the config file – to open the selected file in SBAT. Once completed, the authentication or file selection area will be hidden, the commit button as well as a temporary notification indicating the successful

loading of the file from the given path will appear in the user interface, and the chosen file will be displayed in the editor ready for annotating.

For importing files, all file formats are supported, and they will be treated as plain text files, i.e. the complete content will be displayed as plain text in the editor for annotation. The only exception is the JSON files which have been previously exported from SBAT. These JSON files have standard attributes including text, tokens, entities, and so on, and when imported in SBAT, their attribute values will be parsed, and the annotations saved in the file will be visualized in the editor [Figure 2].

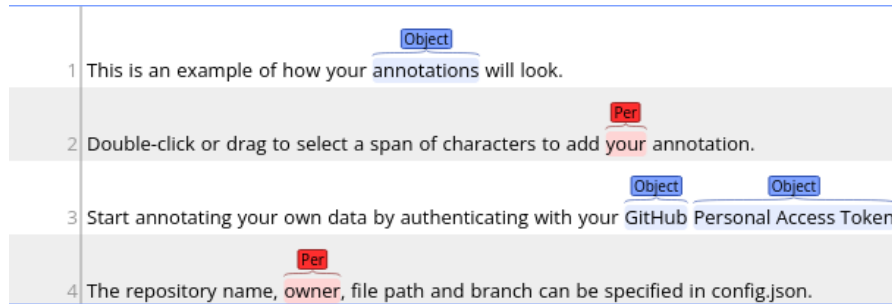


Figure 2: Visualization of Annotations

Annotations can be exported into a JSON file, and when loading this file in SBAT, all previous annotations will be visualized in the editor.

Once the user successfully loads a file from GitHub in the SBAT editor, the exportation of annotations is also straightforward. For that purpose, the user can optionally leave a commit message to document his changes in the input field, and then click the "commit" button. As a result, a JSON file with the original text and its annotation metadata will be committed to the same GitHub repository from which the file was originally imported. If the previously imported file is already a JSON file, this operation will overwrite the previous file. Otherwise, a new JSON file with the same file name will be created and appended to the repository. One example of such SBAT-generated JSON files can be found in the "example data" folder. The JSON format not only enables the visualization of those annotations in SBAT, but also facilitates programmatic parsing for future purposes.

In any step of interacting with GitHub's repository system, if an error occurs, the user will be prompted with the error message with hints to fix it, or, in terms of successful commit, also a message that indicates success. In most cases, it is recommended to check the validity of the Per-

sonal Access Token as well as the repository name, owner, branch and file path entered in the config file. The user account also needs to have or be granted access to the source of the content on GitHub.

SBAT's support for GitHub also enables user management within annotation projects. To accept contributions from multiple annotators in an annotation project, the organizer can create a GitHub repository for the project and share its access with project members. By doing so, all members can contribute to the same project. Moreover, separate branches can be created for different users to create and compare multiple versions. The utilization of GitHub's collaboration features in this way facilitates the management of annotation projects.

3.3 Browser-Based Application

SBAT is browser-based, which means one can access its website to start annotating in the browser. But unlike Brat, all logics of SBAT are executed in the browser, so there is no need for software installation or server configuration. This is to avoid dependencies on external software or devices. As almost all modern computers or mobile devices are equipped with a browser, its accessibility for all users is ensured at a low cost. Moreover, due to its simple construction, it is fast to start and convenient to adapt for different purposes.

Since SBAT functions in the browser, it is able to utilize the localStorage of the browser to save the user's Personal Access Token once logged in, and automatically logs the user in the next time when the site is started in the same browser. That design intends to simplify the log-in process and allow users to concentrate on real annotation tasks. If one wishes to log in with a different account, however, he can utilize the "clear history" function of the browser to clear the previously saved token. By doing so, the authentication page will be shown when restarting SBAT, allowing a new token to be entered.

3.4 Flexible Configuration

Configuration of SBAT is allowed in the config.json file, which is located in the root folder of SBAT. The config file allows centralized management of all annotation-project-related configurations, for instance, the definition of annotation labels and file locations. It aims to simplify user customisation and entitle SBAT to be tailored to a variety of annotation tasks.

For importing files from GitHub, the user is able to enter the repository name, owner, branch and file path under the "admin config" attribute in advance, which will allow SBAT to skip the

intermediate file selection process in every start [Figure 3] and to directly display the pre-defined file in the annotation editor. This also serves as an alternative solution for importing files that are not located in the first level of the repository but in folders, as they can not be directly found by the user's manual selection. In that case, by providing the complete path of the file in the config file, e.g. "folder1/folder3/example.json", the user can easily find and open any file that is located in deeper folder structures.

Select a file

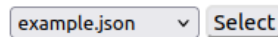


Figure 3: Drop-Down List for File Selection

The "branch" and "filePath" attributes in "admin config" are optional. If no value is provided, SBAT will prompt the user with a drop-down list with available branch or file names to select from.

Details about annotation labels are also defined in the config file. For example, the user can define multiple entity types, their attributes, their children, and the background color for visualization when annotated. They will then appear in the pop-up menu as available options when the user adds a new annotation in the editor [Figure 1]. Besides entity types, there are also event types, event attribute types, relation types, and relation attribute types that can be defined for annotations, each for different annotating tasks. A more complete example of the definition of such types can be found in the config.json file in the "example data" folder.

4 Implementation

4.1 Annotation UI

SBAT's annotation editor is rooted in the frontend component from BRAT, which is implemented using XHTML, Scalable Vector Graphics (SVG), JavaScript, and jQuery. In more detail, in order to replicate the annotation editor, the HTML – including the SVG graphics – of the BRAT editor and pop-up windows is included in the user interface of SBAT. Besides, to enable the interactivity of editor components, the JavaScript scripts from the BRAT editor, including the annotator and visualizer modules, are converted into ECMAScript modules and implemented in SBAT. Two CSS files are imported as well to add style to those HTML components. As the editor component is more time-consuming and requires data from other parts, it is always the last to load in a fresh start. That is to say, it will only be initialized in a JavaScript function after the DOM tree of the HTML file is constructed and the config file is successfully loaded.

On top of that, to remove BRAT's server-side dependence, the user operations in the editor, such as creating, editing or deleting annotations, are all handled by the browser. This is realized by the LocalAjax class, adapting the LocalAjax function from brat-frontend-editor, which locally simulates AJAX requests to process the data without actually sending real network requests. As a result, whenever a new event is triggered by the user operation, it is handled by the functions to change the annotation value that is temporarily stored in the memory of the browser. And after that, the new value is sent to the editor component to update the displayed content as well.

Other UI functions, such as user authentication, and import and export functions, are also implemented natively in JavaScript. And so forth, all functionalities of the user interface are realized in the script files attached to an HTML webpage and are locally executed in the browser.

4.2 Browser-Based Construction

As SBAT is a static web app without server communications, it can be hosted on GitHub Pages for free. It is not mandatory, though, to host the app on the Internet in order to use it, as the index.html file can be used as an entry point to start the app locally in the browser. Nonetheless, if the user plans to access it on the web instead of locally, he can directly upload SBAT's source code to his GitHub repository to host a copy on his GitHub Pages. For the development stage, a copy of SBAT is also hosted on the GitHub Pages, which is done by heading to the SBAT code repository on GitHub -> Settings -> Pages and selecting the option of deploying from the "main" branch.

Once hosted on the web, its link can be shared with other project members for online access. Under that circumstance, it is recommended to leave the "branch" and "filePath" values in the config files empty, so that every project member can select their own branch and file to work on. Moreover, to cater to different use cases, it is also easy to modify any part of the app – SBAT replaces the client-side CommonJS modules of BRAT with ECMAScript modules, so all modules are natively loaded in the browser and no module bundler such as Webpack is used. This allows direct modification of each individual component without the need for re-bundling.

4.3 Usage of GitHub API

SBAT uses Octokit,² the official SDKs for the GitHub API, which enables to implement GitHub operations programmatically. The authentication through a user-provided GitHub Personal Access Token creates a new user-authenticated Octokit object, which can be used to obtain data from GitHub repositories to which the authenticated user has access. Likewise, committing annotations to a GitHub repository is realized by first obtaining the tree of all files from the last commit in the repository, then updating an existing file with the modifications or creating a new file, and finally creating a new tree with the updated files and committing it with the Octokit's REST API endpoint. Moreover, error handling is implemented for every API call, which catches any potential error and prompts the user with informative error messages and hints.

During development, a test GitHub Personal Access Token is generated under GitHub -> Settings -> Developer Settings -> Personal access tokens. SBAT only performs the absolutely necessary operations for file access and version control with GitHub authentication, namely retrieving lists of branches and file names under a specified repository, obtaining file content, getting information about the last commit, and creating new commits, so the scope of the Personal Access Token generated for SBAT only needs to cover those basic repository operations under the "Full control of private repositories" section.

²<https://esm.sh/octokit@2.1.0>, last accessed on September 20th, 2024

5 Conclusion

Text annotation is an important task in linguistic studies, and there have been various applications developed to assist the annotation process. They provide editor-like user interfaces, designed for general or specific annotation tasks according to different annotation schemes. Many of them also adopt machine-learning models to further speed up manual annotation or validation, and provide support for user management in annotation projects. However, they mostly require complicated local or server-side configuration, which increases the cost and time for starting an annotation project. Also, an integrated solution for version management is absent in those tools.

To address those limitations, SBAT aims to provide a simpler and no-server solution for annotation projects. It retains the essential features of BRAT’s annotation editor, such as annotation operations and visualizations, while offering integration with GitHub, enabling the import and export of annotations from and to user-authenticated GitHub repositories. Although its usage is currently limited to text span annotation, it can be easily adapted or extended to accommodate the needs of different annotation tasks due to its modular structure and flexible configuration.

References

- Renaud Hébert-Legault. Brat-frontend-editor [software]. github, version 0.3.36. <https://github.com/crim-ca/brat-frontend-editor>, 2017. URL <https://github.com/crim-ca/brat-frontend-editor>.
- Nancy Ide. *Introduction: The Handbook of Linguistic Annotation*, pages 1–18. 06 2017. ISBN 978-94-024-0879-9. doi: 10.1007/978-94-024-0881-2_1.
- Jan-Christoph Klie, Michael Bugert, Beto Boulosa, Richard Eckart de Castilho, and Iryna Gurevych. The INCEpTION platform: Machine-assisted and knowledge-oriented interactive annotation. In Dongyan Zhao, editor, *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 5–9, Santa Fe, New Mexico, August 2018. Association for Computational Linguistics. URL <https://aclanthology.org/C18-2002>.
- Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun’ichi Tsujii. Brat: a web-based tool for NLP-assisted text annotation. In Frédérique Segond, editor, *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107, Avignon, France, April 2012. Association for Computational Linguistics. URL <https://aclanthology.org/E12-2021>.
- Francis M. Tyers, Mariya Sheyanova, and Jonathan North Washington. UD annotatrix: An annotation tool for Universal Dependencies. In Jan Hajič, editor, *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories*, pages 10–17, Prague, Czech Republic, 2017. URL <https://aclanthology.org/W17-7604>.
- Charlotte Weirich. Sbat [software]. <https://github.com/charlotteweirich/sbat>, 2022. URL <https://github.com/CharlotteWeirich/SBAT>.
- Jie Yang, Yue Zhang, Linwei Li, and Xingxuan Li. Yedda: A lightweight collaborative text span annotation tool, 2018. URL <https://arxiv.org/abs/1711.03759>.