1. Introduction

Over the past years, a lot of annotation tools have been developed for various tasks (for a recent process and feature oriented overview see (Finlayson and Erjavec, 2017); for an overview of multimodal tools see (Cassidy and Schmidt, 2017)). For some reason, however, the domain of discourse annotation, most notably in terms of Rhetorical Structure Theory (RST) (Mann and Thompson, 1988), has not gained much attention for quite a long time. Most RST annotations, with the RST Discourse Treebank (Carlson et al., 2003) (Carlson et al., 2014) leading the way, are carried out with the original RST Annotation Tool (O’Donnell, 1997) or its extension, the ISI RST Annotation Tool (Marcu, 1999). While still usable, both tools, however, are not maintained any more and do not comply to current annotation frameworks. Due to such reasons, a third, browser-based RST annotation tool has been developed recently: rstWeb (Zeldes, 2016). Why, then, is there need for a new tool like TREEANNOTATOR?

We highlight three reasons in the following: Firstly, TREEANNOTATOR, unlike rstWeb, follows the UIMA framework, which supports current requirements of annotation pipelines (Wilcock, 2017). Note that also the Atomic/ANNIS frameworks (Druskat et al., 2014) (Krause and Zeldes, 2016) are not based on UIMA, and hence TREEANNOTATOR is a module following a technologically complementary approach, more related to the “NLP-affine” text annotation tool BRAT (Sstenorp et al., 2012) (which in turn is the technological and design source for UIMA-based WebAnno (de Castilho et al., 2014)). Even extending such frameworks, we developed a UIMA database interface in order to provide a fine-grained type system-compliant storage of documents.

Secondly, users of TREEANNOTATOR benefit from different visualizations. The usefulness of visualizing annotations has been independently highlighted, for instance by (Finlayson and Erjavec, 2017) p. 183 and (Biemann et al., 2017) p. 236). In case of polynuclear hierarchies there are several ways to display the resulting structures, which in turn emphasize different structural features. Accordingly, TREEANNOTATOR provides – in addition to the classical RST view – an inclusion view that adopts rhetorical structure representations in the tradition of Discourse Representation Theory (DRT) (Kamp and Reyle, 1993), namely Segmented Discourse Representation Theory (SDRT) (Asher and Lascarides, 2003). The visualization flexibility not only provides alternative views on given rhetorical structures, it also makes TREEANNOTATOR’s output more “readable” to modern theoretical linguistics. Thus, TREEANNOTATOR may reach a new group of users.

Thirdly, the newly designed graphical interface of TREEANNOTATOR significantly improves the tool’s usability for rhetorical annotations: this has been evaluated in a study in comparison to the RSTTool.

The components, annotation processes, and visualizations of TREEANNOTATOR are described in Sec. 2. The usability study is reported in Sec. 3. Future developments are summarized in Sec. 4.

2. TREEANNOTATOR

2.1. Annotating Tree-like Text Structures in TREEANNOTATOR

TREEANNOTATOR is an up-to-date, browser-based and freely available device for discourse annotation. It is designed as a module of the still-developing annotation suite TEXTANNOTATOR. TEXTANNOTATOR in turn draws on the architecture of TEXTIMAGER (Hemati et al., 2016), an UIMA-based framework that offers a wide range of NLP and visualization tools through a user-friendly GUI. All NLP tools that are available via TEXTIMAGER can be utilized for preprocessing input documents, say, in terms of tokenization, lemmatization or pos tagging. TREEANNOTATOR’s front-end is based on the ExtJS framework and

https://www.sencha.com/products/extjs/#overview
uses the $D3.js$ library for visualizations. It was designed for annotating RST trees (Mann and Thompson, 1988) and structures that jointly map dependency relations and inclusion hierarchies. As such, TREEANNOTATOR covers and extends the functionality of previous RST annotation tools, but is completely newly developed in order to meet current technological standards and requirements. The user interface characteristics of TREEANNOTATOR give rise to the following features, which are elaborated subsequently:

- the number of mouse clicks for carrying out an RST annotation is significantly reduced, thanks to a new approach of how to annotate rhetorical structures. In particular, spans no longer have to be added manually, but are included automatically based on selected relation types – a respective user study is reported in Sec. 3.
- the flexibility and interoperability exceeds comparable tools by offering new output formats (e.g. $\LaTeX$ and XMI; see Sec. 2.2 and Sec. 2.3, respectively);
- TREEANNOTATOR provides an alternative mode of interactive visualization (and, hence, annotation), including a split-window facility, which extends the classical RST view (Sec. 2.2);
- finally, TREEANNOTATOR includes a redo/undo functionality due to an underlying versioning system (Sec. 2.3).

### 2.2. Visualizing Annotations in TextAnnotator

TREEANNOTATOR’s visualization addresses current workflow needs more specifically than comparable tools: In particular, (a) a second mode of visualization is available, independent from classical RST trees; (b) $\LaTeX$ output for scientific publications can be generated; (c) a multiple window view to inspect annotations side by side is provided.

(a) **Visualization Modes** Representing data by means of alternative modes helps in understanding and interpreting the subject matter. To this end, TREEANNOTATOR supports two modes of visualizing tree-like structures: classical RST trees and dependency-oriented inclusion hierarchies (DIH) (see Fig. 1 and 2). TREEANNOTATOR allows for switching between the two available visualizations even during annotation.

Visualizations in RST mode are similar to those of (ISI) RSTTool and rstWeb. However, TREEANNOTATOR’s mouse wheel zooming and dragging functionality adds a useful feature, especially for annotating larger tree structures: TREEANNOTATOR has been tested with trees of more than one hundred Elementary Discourse Units (EDUs, the text spans connected by rhetorical relations); RSTTool and rstWeb in comparison are limited to displaying roughly two dozen EDUs simultaneously on screen. Unlike ANNIS, which was developed for generic graph visualizations, TREEANNOTATOR offers two dedicated RST modes, specifically designed for quick annotation work. ANNIS’ latest version, ANNIS3, provides a solution for RST trees by means of a visualization plugin (Krause and Zeldes, 2016). However, the application is limited to only displaying previously annotated corpora. The corresponding annotation tool Atomic (Druskat et al., 2014) is currently still in development and not available as a browser-based tool.

![](https://d3js.org/)

![Figure 1: TREEANNOTATOR’s classical RST mode, illustrating the same RST structure as shown in Fig. 2](https://d3js.org/)

![Figure 2: DIH: Mode to view integrated dependency and inclusion hierarchies in TREEANNOTATOR. The comment is added to the PGF/TikZ source code.](https://d3js.org/)
Figure 3: Multiple window view, showing two concurring annotations (Moore and Pollack, 1992) p. 542 f.). Several collapsible annotation views can be visualized.

boundary for anaphoric attachment sites and, hence, is of importance for any further co-reference annotation.

(b) Graphics Output Besides screenshot-like graphics (PNG), TREEANNOTATOR allows for generating \( \text{Latex} \) output: RST trees are exported using the \textit{rst package} (Reitter, 2002), DIHs are output to PGF/TikZ (Tantau, 2015). PGF/TikZ are formal languages that can be interpreted by \( \text{Tex} \) (K ruth, 1984), \( \text{Latex} \) (Lamport, 1994) (including Xe\( \text{La} \)tex), and Con\( \text{La} \)tex (Hagen and Hoekwater, 2013) (including Lua\( \text{La} \)tex) to produce vector graphics output. The PGF/TikZ format is not only suitable for direct inclusion in \( \text{Latex} \) documents – the standard format of many scientific publication organs –, but enables further modifications within the graphic’s source code. By this means, relevant features can be highlighted for publications – as illustrated in Fig. 2 –, or visualizations can be uncovered piecewise in presentations for talks.

(c) Multiple Window View A well-known problem of RST annotations is that there may be several, equally justified but concurring rhetorical analyses and structures of text spans (Moore and Pollack, 1992). Since this pluralism of rhetorical analyses is perfectly legitimate according to RST (Das et al., 2017), it has to be accounted for in the annotation workflow. Accordingly, a multiple window view was developed, which allows users to simultaneously inspect multiple annotations of the same text side by side, see Fig. 3. In the current version, this is achieved by simply creating a copy of the document. In future releases, the different annotations will be stored in the same document by using different \textit{Subjects of Analysis}, that is, specific UIMA objects\footnote{https://uima.apache.org} which will further open up the possibility of having different annotations even for fragments of a text.

2.3. Storing Annotations in TreeAnnotator

Since the Apache UIMA framework (Ferrucci et al., 2009) is one of the standards in projects within an NLP context (Wilcock, 2017), it is also used in the TEXTANNOTATOR suite. However, somewhat surprisingly this document-based schema does not provide native database support.

To this end, we developed the \textit{UIMA Database Interface} (UIMA DI) (Abrami and Mehler, 2018). It stores arbitrary UIMA documents based on their associated \textit{UIMA Type System Descriptions} (UIMA TSD) in any database back-end (see Fig. 4). In the current version, the connections to MongoDB and Neo4J\footnote{Available on GitHub under GPLv3. \( \text{https://www.mongodb.com/} \) \( \text{https://neo4j.com} \) \( \text{http://www.textannotator.hucompute.org/typesystem/graph} \) \( \text{http://graphml.graphdrawing.org/index.html} \) \( \text{http://www.omg.org/spec/XMI/} \) } are already implemented and further database back-ends can be added easily. For the general purpose of storing and using tree-like structures in UIMA frameworks, we developed a new UIMA TSD for graphs\footnote{https://uima.apache.org} compliant to the GraphML\footnote{https://www.w3.org/2001/06/xlink} format. Based on the UIMA TSD we defined an object model for RST trees, which is used in TEXTANNOTATOR (and thereby in TREEANNOTATOR). In order to efficiently store and retrieve annotations created by those tools, we employ established database technologies. Within this technological context, annotations are stored in the XMI\footnote{https://uima.apache.org} format during annotation processes. The resulting XMI document can be stored in any database integrated into UIMA DI or downloaded to a text file.

Table 1 summarizes the input/output formats which are cur...
3.2. Results

Using TREEANNOTATOR resulted in a decrease of mouse clicks per annotation. On average, annotators working with TREEANNOTATOR needed 14.9 clicks less compared to annotators using RSTTool; the median was improved by 18.0 clicks (or −32.7%, see Tab.[3]).

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>RSTTool</th>
<th>TREEANNOTATOR</th>
<th>improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg</td>
<td>36.8</td>
<td>22.4</td>
<td>−39.0%</td>
</tr>
<tr>
<td>med.</td>
<td>36.5</td>
<td>19.0</td>
<td>−47.9%</td>
</tr>
<tr>
<td>avg</td>
<td>45.4</td>
<td>32.1</td>
<td>−29.3%</td>
</tr>
<tr>
<td>med.</td>
<td>46.5</td>
<td>28.0</td>
<td>−39.8%</td>
</tr>
<tr>
<td>avg</td>
<td>54.9</td>
<td>37.8</td>
<td>−31.2%</td>
</tr>
<tr>
<td>med.</td>
<td>55.5</td>
<td>36.0</td>
<td>−35.1%</td>
</tr>
<tr>
<td>total</td>
<td>43.7</td>
<td>30.8</td>
<td>−32.7%</td>
</tr>
<tr>
<td></td>
<td>46.5</td>
<td>28.3</td>
<td>−38.7%</td>
</tr>
<tr>
<td>avg</td>
<td>38.7</td>
<td>22.4</td>
<td>−39.0%</td>
</tr>
<tr>
<td>med.</td>
<td>35.1</td>
<td>19.0</td>
<td>−47.9%</td>
</tr>
</tbody>
</table>

Table 3: Number of mouse clicks for 12 pre-specified RST annotations with RSTTool and TREEANNOTATOR

In terms of annotation time, TREEANNOTATOR again performed better in all measured categories. Using RSTTool,

<table>
<thead>
<tr>
<th>Feature</th>
<th>WebAnno</th>
<th>rstWeb</th>
<th>TREEANNOTATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>user management</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>group management</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>user permissions on projects / collections</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>user permissions on documents</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>group permissions on projects</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>group permissions on documents</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>project-independent document usage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>document organisation in repositories</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hierarchical repositories</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Comparison of TREEANNOTATOR, WebAnno and rstWeb with regards to rights and resource management

3. Evaluation

3.1. Setting

In a usability study, two groups of five test subjects each were asked to annotate pre-specified RST trees, comparing TREEANNOTATOR to the most widely used RST annotation software, RSTTool. The subjects had no prior acquaintance with RST tools and RST annotations. However, they had a brief introduction into Rhetorical Structure Theory and its polynuclear relations. Twelve text segments (with three, five and seven RST relations, respectively) of the Potsdam Commentary Corpus (Sede, 2004) (Applied CL Discourse Research Lab, 2017) were handed out on paper. Group one annotated using TREEANNOTATOR, group two worked with RSTTool. The results are shown below.

2.4. Versioning Annotations in TREEANNOTATOR

When dealing with error-prone, highly interpretive annotation tasks like RST analysis, undoing and redoing operations should be a fundamental part of the workflow (see e.g. the results in Sec. [3.2]). TREEANNOTATOR currently supports an undo/redo functionality within the same session by versioning the XMI file in the internal memory. TREEANNOTATOR inherits TREEANNOTATOR’s versioning facility and thus provides the same functionality as rstWeb (v2.0) and RSTTool (v3.0), but exceeds tools such as WebAnno (v3.2.2), which does not offer an undo function.

Thus, any annotation can be revised during the actual session. Since storing a copy of the complete file for every revision leads to a high memory usage, especially for larger documents, we are currently looking into genuine version control systems. Using such a system will not only lead to reduced memory usage, but will also allow TREEANNOTATOR to provide an undo/redo functionality even beyond sessions.

2.5. Managing Annotations in TREEANNOTATOR

Since it cannot be guaranteed in any case that the natural language texts to be annotated are free of copyright restrictions, a rights and resource management is required. To this end, TREEANNOTATOR uses components of the eHumantics Desktop (Gleim et al., 2012) which allows managing user rights and resources: AuthorityManager and ResourceManager (see Fig. [4]). By integrating UIMA DI into ResourceManager, any UIMA document becomes accessible as a resource and organized in projects, which are defined as repositories. That is, user rights can be inherited from higher-level groups via restriction or extension in a hierarchical manner: they can be shared with individual users or made public to a wider audience simply by assigning appropriate authentications to files and users.

Tab.[2] compares TREEANNOTATOR’s rights and resource management, as facilitated by AuthorityManager and ResourceManager, to that of rstWeb and the widespread WebAnno. The access options applied by TREEANNOTATOR are more elaborate than those of the compared tools. Especially, TREEANNOTATOR acknowledges the level of groups, allows to share results with third parties and provides data access for external tools.

In the context of research projects, files to be annotated are usually stored in a folder that is shared among the different project partners. All project partners have at least writing permissions and can therefore jointly work on the annotation sources. Now annotation results can be published in a targeted manner: they can be shared with individual users or made public to a wider audience simply by assigning appropriate authentications to files and users.

Table 1: Input/output formats of TREEANNOTATOR.

<table>
<thead>
<tr>
<th>Input</th>
<th>Data output</th>
<th>Graphical output</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain text, RST file format</td>
<td>UIMA-conform</td>
<td>\LaTeX{} (PGF/TikZ), PNG</td>
</tr>
</tbody>
</table>

Tab. [1] compares TREEANNOTATOR’s input/output formats to those of WebAnno and rstWeb.
annotators needed 104.9 sec for the twelve texts on average, while user of TreeAnnotator managed an average time of 96.3 sec (an improvement of −8.6 sec or −8.2%; see Tab. 4).

\[
\begin{array}{cccccc}
\text{RSTTool} & \text{TreeAnnotator} & \text{improvement} \\
\hline
\text{avg} & \text{med.} & \text{avg} & \text{med.} & \text{avg} & \text{med.} \\
\hline
\text{short} & 84.3 & 78.0 & 70.8 & 58.0 & −16.0 & −25.6 \\
\text{medium} & 101.9 & 101.5 & 92.9 & 82.5 & −8.8 & −18.7 \\
\text{long} & 128.6 & 124.5 & 125.3 & 116.5 & −2.5 & −6.4 \\
\hline
\text{total} & 104.9 & 101.5 & 96.3 & 95.0 & −8.2 & −6.4 \\
\end{array}
\]

avg – average (arithmetic mean), med. – median

Table 4: Required time (in sec) for 12 pre-specified RST annotations with RSTTool and TreeAnnotator

The number of annotation errors for both tools is virtually the same (see Tab. 5). However, the types of errors differed: annotators working with RSTTool mainly erred with respect to RST tree structures, while annotators using TreeAnnotator mainly produced incorrect text segmentations. As a result, TreeAnnotator’s text segmentation was subsequently altered to make the currently selected segment more obvious for users.

The number of annotation errors for both tools is virtually the same (see Tab. 5). However, the types of errors differed: annotators working with RSTTool mainly erred with respect to RST tree structures, while annotators using TreeAnnotator mainly produced incorrect text segmentations. As a result, TreeAnnotator’s text segmentation was subsequently altered to make the currently selected segment more obvious for users.

\[
\begin{array}{cccc}
\text{RSTTool} & \text{TreeAnnotator} \\
\hline
\text{Incorrect relation name} & 3 & 1 \\
\text{Incorrect RST structure} & 6 & 2 \\
\text{Incorrect text segmentation} & 0 & 5 \\
\text{Output file missing entirely} & 1 & 1 \\
\hline
\text{total} & 10 & 9 \\
\end{array}
\]

Table 5: Number of errors for each annotation tool (60 pre-specified RST annotations per group in total)

Users of TreeAnnotator (on average) needed 28.5% of their time for segmentation, while the program’s undo function was used very sparsely. One concern illustrated in Tab. 6 is the fact that the time per newly included relation increases with text length, a problem partly due to the hardware used during the study, which did not allow for smooth zooming and moving of larger RST trees.

\[
\begin{array}{cccc}
\text{text length} & \text{time for segmentation} & \text{Undo actions} & \text{Time per new EDU} & \text{Time per new relation} \\
& \text{(% of total time)} & \text{(per text)} & \text{(in sec)} & \text{(in sec)} \\
\hline
\text{short} & 29.6% & 0.30 & 4.4 & 5.1 \\
\text{medium} & 28.6% & 0.35 & 4.1 & 8.2 \\
\text{long} & 27.3% & 0.15 & 4.1 & 12.1 \\
\hline
\text{total} & 28.5% & 0.27 & 4.2 & 8.5 \\
\end{array}
\]

Table 6: Average user behavior for pre-specified RST annotations in TreeAnnotator

3.3. Summary

TreeAnnotator provides a state of the art approach to rhetorical structure annotation. TreeAnnotator outperforms RSTTool in all categories shown above (albeit only minimally in terms of error-proneness). Improvements in the number of needed mouse actions is a particularly remarkable result. Furthermore, we found a reduced annotation time, especially for shorter to medium length texts.

4. Outlook

TreeAnnotator offers a modern and arguably intuitive approach to discourse annotations for RST-like structures, with unique features that set it apart from related tools. Future versions will add support for annotations based on representations from DRT and SDRT and a newly designed module for semantic analysis, utilizing various lexical resources to create a comprehensive model of semantic representation. However, rhetorical structure annotations will not be overloaded by adding, say, semantic or anaphoric annotations. Rather, a multilayer annotation scenario will be implemented, which has already proven to be useful, for instance, in the GUM project (Zeldes, 2017). Text-Annotator’s code will be made available on GitHub in the future. Meanwhile, the tool can be reviewed at http://www.textannotator.hucompute.org (login: textAnnotator, password: textAnnotator).

5. Bibliographical References


6. Language Resource References
