UNIVERSITY OF SÃO PAULO

School of Arts, Sciences and Humanities

Technical Report PPgSI-001/2022

ResDialTools – A Toolset for Text Annotation

Caio T. Cruz, Matheus R. A. Veneziani,
Norton T. Roman, Alexandre R. Alvares
Tiago E. I. Missão, Daniel V. da Silva

March - 2022

Technical Report Series

PPgSI-EACH-USP
Arlindo Béttio St. 1000 - Ermelino Matarazzo - 03828-000.
São Paulo, SP. Brazil.
TEL: 55 (11) 3091-8197 http://www.each.usp.br/ppgsi

ResDialTools – A Toolset for Text Annotation

Caio T. Cruz[1] **, Matheus R. A. Veneziani**[1] **, Norton T. Roman**[1]
Alexandre R. Alvares[1] **, Tiago E. I. Missão**[1] **, Daniel V. da Silva**[1]

[1]Escola de Artes, Ciências e Humanidades – Universidade de São Paulo
São Paulo – SP, Brazil

Abstract. *This article describes a toolset designed for general-purpose text annotation tasks. The toolset comprises a pipeline of three independent but interconnected tools, covering all steps throughout the annotation process, from data segmentation to data annotation to annotation evaluation. These tools were primarily built to address three main issues found in current general-purpose annotation tools: (i) the potential confounding of variables, as a result of having annotators do both segmentation and annotation in a single step; (ii) the cognitive load imposed to annotators; and (iii) the difficulty in comparing one study to others when different agreement indexes are reported.*

Within this toolset, these issues are dealt with by (i) having different tools perform data segmentation and annotation separately; (ii) giving researchers a tool where they can define and generate ad hoc tools, tailored to specific annotation schemes, to be distributed amongst annotators; and (iii) furnishing a way to calculate inter-annotator agreement according to six different indexes. Given its modularity, the toolset can be used both as a pipeline, whereby the output of one tool can be input to the next one, and for specific subtasks. It then lends itself, among other things, to the quick prototyping and testing of annotation schemes, to the execution of full annotation experiments and efforts, and to the study of how different agreement indexes behave on the same data.

1. Introduction

Within the field of Computational Linguistics, *corpora* have been historically used for a variety of tasks, from the construction of tutoring systems (*e.g.*, Callaway et al. [2005]) to automatic summarisation (*e.g.*, Radev et al. [2004], Hasler [2007], de Loupy et al. [2010], Atkinson and Munoz [2013]), also serving as the basis for the observation and proposal of hypotheses, along with their optimisation and assessment [Mitkov et al. 1999], or even for the modelling of linguistic phenomena through the use of machine learning techniques [Pustejovsky and Stubbs 2012]. Whatever the end, the decision about relying on some *corpus* to produce results comes at the price that, in order to make them useful, they usually must receive some sort of pre-processing, in the form of meta-data holding information about the phenomenon of interest, *i.e.* they must be annotated [Pustejovsky and Stubbs 2012].

Annotating a *corpus*, however, is no simple task, specially if the procedure is to be carried out by humans. The problem, in this case, is that annotators usually produce different results when compared to each other, given that this task is, in fact, an interpretation process by its executors [Grouin et al. 2011]. These differences are usually accounted for through the use of some inter-annotator agreement measure, that is a measure of the extent to which different people assign the same label to the same portion of the annotated material, thereby determining the consistency of the annotation across annotators [Grouin et al. 2011].

In fact, high levels of inter-annotator agreement are very much welcome in any annotation effort, since the higher the number of people agreeing in their classification, the higher our confidence in data quality [Craggs and Wood 2005, Bayerl and Paul 2011, Grouin et al. 2011]. Moreover, high agreement scores also allow us to infer that annotators clearly understood the annotation scheme they used [Artstein and Poesio 2008], thereby increasing our confidence in the appropriateness of the annotation scheme itself. This agreement, when taken as a measure of our confidence in the quality of both annotated data and annotation scheme, becomes a necessary condition for assessing any scheme's validity [Artstein and Poesio 2008].

There are nevertheless some hurdles in the path to high agreement scores. Apart from the natural subjectivity of the task at hand, which in an on itself constitutes a source of disagreement (*e.g.* Gut and Bayerl [2004], Alm et al. [2005]), some experimental choices may actually pose further threats to this measure. One such choice is, for example, allowing for a confusion of variables, whereby disagreement cannot be associated to a single experimental condition. This confusion might come up as a result of having the same set of annotators define the basic unit of annotation along with its label (*e.g.* [Stenetorp et al. 2012, Pérez-Pérez et al. 2015]), or of allowing them to modify specific units (*e.g.* [Lenzi et al. 2012]). Even though there are situations that cannot be helped, when analysing the data researchers would not be able to determine which combination of basic unit definition, annotation and annotator was responsible for the observed results, making it harder to redesign the procedure should something go wrong.

Another important aspect of designing an annotation experiment is the cognitive load imposed to annotators. In that sense, one should limit difficulties as much as possible to those naturally found in the data interpretation task, since the reliability of the annotated material has been found to correlate with the complexity of the annotation process (*cf.* Gut and Bayerl [2004]). Within this setup, the use of annotation tools may be of great value, providing a speed-up to one of the most time-consuming and financially costly parts of the research [Stenetorp et al. 2012]. Such tools, however, should not increase the annotation burden, by demanding annotators to learn their idiosyncrasies. They should, then, be simple to learn and provide intuitive and user-friendly interfaces, so as to allow for less specialised non-technical annotators, such as domain experts, to use them [Orăsan 2003, Stenetorp et al. 2012, Bontcheva et al. 2013].

Once data are annotated yet another difficulty arises, which lies in the heart of applying inter-annotator agreement measures. The problem in this case is that, despite the importance of measuring agreement, there is no single standard way of calculating it. This lack of a proper measure, in turn, opens room for the use of a number of methods, ranging from the plain reckoning of the proportion of annotations upon which annotators agree (*e.g.* Petasis [2012]), to more elaborated coefficients, such as Krippendorff's α [Krippendorff 2004] and Cohen's κ [Cohen 1960]. As for this last one, κ has been at some point announced as the *de facto* standard coefficient for Computational Linguistics (*cf.* [Carletta 1996]), only to be debunked later on, due to its undesirable behaviour with some data distributions (*cf.* Eugenio and Glass [2004], Artstein and Poesio [2008]), and so leading us back to the starting point.

This uncertainty about how to measure inter-annotator agreement has already raised a debate about how appropriate current indexes are (*e.g.* Geertzen and Bunt [2006], Art-

stein and Poesio [2008], Bayerl and Paul [2011]), with experiments being carried out on a number of different proposals (*e.g.* Grouin et al. [2011], Mathet et al. [2012], Fort et al. [2012]). However important as a debate, the current situation leaves Computational Linguistics experimenters with no other choice but to stick to the method they believe to better suit their needs. As such, different researchers report their results using different indexes, sometimes making their comparison across independent studies virtually impossible.

An indicative of this problem's magnitude can be found in a literature review comprising publications on prosodic and phonetic transcriptions, along with word-sense disambiguation [Bayerl and Paul 2011], where it was found a total of 972 agreement indexes reported in 326 studies, leading to a mean value of almost three different indexes per study. As it seems, researchers in Computational Linguistics have already adopted the strategy of reporting multiple agreement indexes in their publications (*e.g.* Petasis [2012], Fort et al. [2012]), a choice that comes at the price of having to calculate agreement according to each of the reported indexes.

To help them in this task, some annotation tools already come with a set of preprogrammed indexes (*e.g.* Apostolova et al. [2010], Verhagen [2010], Petasis [2012]). Still, it might be the case that (i) researchers are not capable of adapting the annotation tool to their needs; or (ii) the desired agreement measure is not implemented in the tool at hand, since most of the existing tools seem to provide no more than two indexes (*e.g.* Müller and Strube [2006], Petasis [2012], Bontcheva et al. [2013]), thereby reducing the odds that one's research can be compared to others'.

In this article, we describe a toolkit comprising three independent but interconnected tools designed to approach each of the above mentioned issues separately, but also allowing for the final integration of their results, and so filling all the steps throughout the annotation process, from data segmentation to data annotation to annotation evaluation. As one of its main features, the toolkit allows for the independent execution of each of these stages, with the integration of partial results so as to form an overall output. Even though beta versions of each tool have already been presented to the public (mostly in Portuguese), here we introduce their stable versions, with more features added and having undergone a set of usability tests, which resulted in their (sometimes deep) modification, in order to reduce further the so desired low cognitive burden imposed both to researchers and annotators.

Modifications made to the toolset include (but are not limited to): the addition of new agreement coefficients, going from four to six; the modification of the Graphical User Interface of some tools; the possibility of carrying out projects both in Portuguese and English (originally the toolset was available in Portuguese only); and the standardisation of their internal codification (originally, the data codification scheme output by the segmentation tool differed from the standards required by the remaining tools). All tools are currently distributed under GPL[1], being available for download at `https://github.com/NortonTR/ResDialTools`. We hope the community will find them useful.

The rest of this article is organised as follows. Section 2 gives an overview to the toolset, along with some implementation and testing details. Next, in Section 3, we

[1]GNU General Public License: `www.gnu.org/licenses/gpl-3.0.html`

present the results of the usability tests undergone by each tool in the toolset. In this section, we also present a detailed description of each tool, as a result of the correction of the usability problems, also describing all incorporated features since their beta versions. Section 4, in turn, compares our system to some other existing annotation tools, pointing out the main differences between these projects and ours. These differences are further discussed in Section 5, where we try to determine when they can be taken as advantages and in which situations they would not be desirable features. Finally, in Section 6 we present a conclusion to this work, along with directions for future improvement.

2. Toolset Overview

Figure 1 illustrates a typical workflow within our toolset. It is worth observing, however, that even though the toolset was originally designed to work as a pipeline, each of its stages can be run separately, provided that input files are coded following the standards defined by Roman [2013]. In this figure, the process begins with a *corpus* of "raw" texts, that is texts with no other information added. These texts are then input to TSeg [Rodrigues et al. 2012], where users can segment them in spans (either automatically, manually, or a combination of both), so as to produce the basic units for annotation. The resulting segmentation may then be inspected by the researcher, modifications can be made and, if necessary, the process can be started over. This is a specially useful feature in case of manual annotations, where researchers might wish to determine if segmentation instructions were well understood by annotators [Pérez-Pérez et al. 2015] before going deeper in the process. For this reason, and similar to Verhagen [2010], only one segmentation scheme is allowed throughout the entire annotation process.

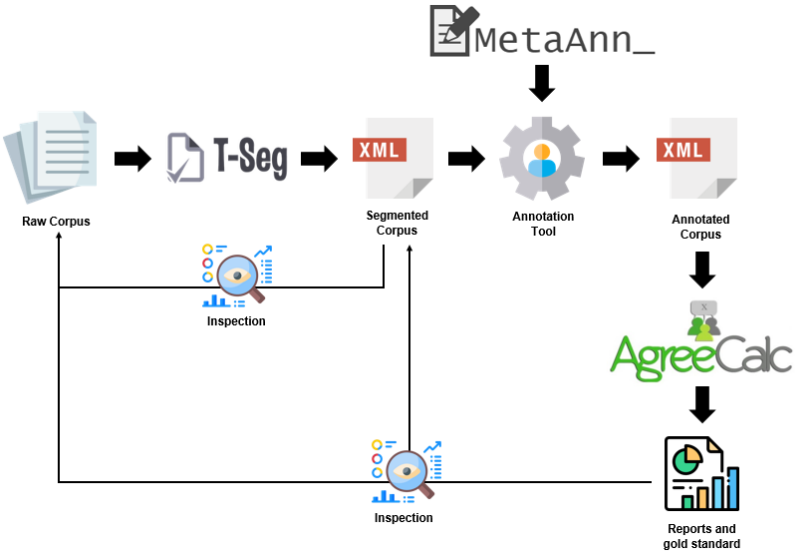


Figure 1. Typical workflow within our toolset.

Once a segmentation is settled, researchers can define an annotation scheme, with categories, values and labels, and use MetaAnn [Missão and Roman 2013] to codify it through its Graphical User Interface. Upon finishing with this codification, they can have MetaAnn build the source code (along with its compiled counterpart) for the annotation tool to be distributed to annotators. The distributed tool will then take as input the segmented corpus (output by TSeg), and add any annotation made by its user. In doing so,

MetaAnn provides researchers with a graphical interface for the creation of *ad-hoc* annotation tools, allowing them to codify their annotation schemes. Throughout this process, and as recommended by Orăsan [2003], MetaAnn not only hides the details of the annotation scheme away from annotators, but also reduces their cognitive load, by building an annotation tool with a simpler interface, and so having annotators focus on the annotation task itself.

This tool can then be distributed to a small number of annotators for testing purposes. Annotated corpora, output by the tool, can be collected and input to AgreeCalc [Alvares and Roman 2013], which presents researchers with a set of six different coefficients of agreement they can apply to the dataset. Besides furnishing this range of possible indexes, AgreeCalc also allows for their application to different subsets of annotation units and annotators, thereby allowing for a more in-depth analysis of the results (*cf.* Gut and Bayerl [2004]), generating reports in HTML, XML and PDF. These reports, in turn, contain not only the overall agreement result of the indicated dataset, but also an analysis of the agreement calculated amongst all possible pairs of annotators, so as to allow for the identification of any deviant behaviour, along with an agreement range (from the lowest to the highest agreement pair).

If agreement turns out to be low in this tentative set, researchers can look at these reports, verifying differences between pairs of annotators. With these results at hand, they can make modifications to the annotation scheme and annotation guidelines, rerunning MetaAnn and so starting the annotation process (or even the entire process, beginning with segmentation) over, until they come up with a more reliable scheme. This final scheme (in fact, the annotation program that applies it to the corpus) can then be distributed to a larger amount of annotators. At the end of the annotation period, annotators may send their results (*i.e.* the files output by MetaAnn's generated tool) back to the researchers. These files correspond to the original (segmented) corpus with annotations added to it in a stand-off manner (*e.g.* [Mueller and Strube 2001, Lenzi et al. 2012, Bontcheva et al. 2013, Pérez-Pérez et al. 2015, Janssen 2016]).

With all annotations at hand, final agreement figures can be calculated and, if they turn out to reach some acceptable amount, researchers can use AgreeCalc to build gold standards for training and/or testing their future models. As illustrated, our toolset provides a way to separate the tasks within the annotation effort, thereby hiding task-specific details away from the remaining annotation steps, and making them simpler and easier to be carried out, also allowing for probing tests to be made alongside the process. Additionally, the toolset also complies with some commonly found principles of design (*cf.* Mueller and Strube [2001], Müller and Strube [2003], Müller and Strube [2006], Geertzen and Bunt [2006]), such as the use of XML to store its data (which forms the base of its codification scheme, presented in Roman [2013]) and the use of stand-off annotation.

Finally, the fact that the toolset was implemented in Java makes it platform independent, as advised in Mueller and Strube [2001]. This is a desirable feature, given the freedom it gives to researchers. But beyond, since the tool generated by MetaAnn is also written in Java, so is this tool platform independent, thereby increasing the amount of potential annotators capable of using it. As an additional feature to help make the toolset available to a broader public, all texts it manipulates can be codified using UTF-8, making it suitable to any supported language (*cf.* Day et al. [2004], Lenzi et al. [2012], Stenetorp

et al. [2012]). In what follows, the usability tests undergone by our toolset will be described in more detail whereas each tool, along with the changes we made from their beta versions to the current ones, are described in Section 3.

2.1. Testing Usability

As a way to verify the above claimed simplicity and ease of use, each tool in our toolset undertook a couple of usability tests, which gave birth to the versions we present here. Amongst the available techniques to test usability, in this work we focused on Heuristic Evaluation, which involves having evaluators examine the interface in light of recognised usability principles – the heuristics [Nielsen 1993], and Cognitive Walkthrough, where the investigator "walks through" the interface in the context of the tasks a typical user would try to accomplish, comparing the actions and feedback by the interface to this user's assumed goals and knowledge [Jeffries et al. 1991, Jordan 1998].

The tests started with the definition of test scenarios, *i.e.* tasks the user must execute within the system[2] (*cf.* Dumas and Redish [1999]), built from a (depth) search through each tool's user manual. Since AgreeCalc had no user manual up to this point, scenarios for this tool were constructed from an analysis of its description in Alvares and Roman [2013] instead. In all tests, users were supposed to be experienced in the computational platform at had, that is to say they were assumed to be familiar with navigating through the basic functions of graphical interfaces (such as closing, opening, minimising or maximising windows) and the use of the mouse (or any other pointing device) and its buttons.

During the Cognitive Walkthrough, each scenario was examined bearing in mind four key points [Lewis and Wharton 1997]: (i) will the users perform the right action for the desired effect?; (ii) will they perceive the right action is available?; (iii) will they associate the right action with the desired effect?; and (iv) if the correct action is executed, will they perceive progress was made towards the desired effect? Answers are then documented in a Walkthrough Form, and a negative answer to any of these questions will deem the interface to have failed with this respect, and so point out a usability problem to be addressed.

For Heuristic Evaluation, we relied on the heuristics proposed by Nielsen [1994], which cover ten key aspects with which interfaces should comply, from aesthetic to error prevention to documentation issues[3]. Analysed scenarios were the same as those used during Cognitive Walkthrough. Each scenario was then assessed according to the whole set of heuristics, with results documented in an Evaluation Form. Once again, should any scenario fail with respect to any of these heuristics, the interface will be regarded as having a usability problem. Interestingly, various problems found by Heuristic Evaluation were not found by Cognitive Walkthrough and vice-versa, indicating that both techniques can be used as a complement to each other.

Scenarios resulting in usability problems (*i.e.* those failing in any of the usability tests) were tabled and sorted according to the severity of the reported problem. In this work, we ranked problems according to the scale presented by Nielsen [1993, p. 103], which classifies them as irrelevant (not a problem at all), cosmetic (fixed only if time allows it), minor (low priority), major (high priority) and catastrophic (imperative fixing). Problems

[2]Not to be confused with scenarios for software testing.

[3]For a complete list we refer the interested user to [Nielsen 1994].

were then addressed following a descending order of severity. Even though our initial goal was to solve only problems from catastrophic to minor, we managed to solve them all, with a single exception, regarding a proper system icon for the toolset.

In what follows, more details on the discovered usability problems will be presented, along with a description of each tool's current form and functionalities. Differences between current versions and previous (beta) ones will also be pointed out, so as to allow for a better comparison between them.

3. Results

During Heuristic Evaluation, TSeg failed in 14 of the 18 tested scenarios (78%), MetaAnn failed in 16 of 22 (73%), its generated tool in 11 of 13 (85%), and AgreeCalc failed in 39 of 41 scenarios (95%). Even though resulting in relatively fewer failures, Cognitive Walkthrough still pointed out 10 failed scenarios in TSeg (out of 18 – 56%), of which nine were also found by Heuristic Evaluation; seven (out of 22 – 32%) for MetaAnn, of which six were also determined by Heuristic Evaluation; six (of 13 – 46%) for MetaAnn's generated tool, all of them also found during Heuristic Evaluation; and 18 (of 41 – 44%) for AgreeCalc, of which 17 were also highlighted during Heuristic Evaluation. Tables 1 and 2 summarise these results.

Table 1. Scenario results for Cognitive Walkthrough and Heuristic Evaluation.

		Cognitive Walkthrough		Heuristic Evaluation	
Tool	*Tested*	*Success*	*Failure*	*Success*	*Failure*
TSeg	18	8	10	4	14
MetaAnn	22	15	7	6	16
MetaAnn's tool	13	7	6	2	11
AgreeCalc	41	23	18	2	39

Table 2. Failed scenarios for Cognitive Walkthrough (CW) and Heuristic Evaluation (HE).

	TSeg	*MetaAnn*	*MetaAnn's Tool*	*AgreeCalc*
Failed in both	9	6	6	17
Failed only in CW	1	1	0	1
Failed only in HE	5	10	5	22
Failed in none	3	5	2	1
Total	18	22	13	41

In these tables, its interesting to notice that, despite Heuristic Evaluation having resulted in the higher amount of identified usability problems, other failures were only identified during Cognitive Walkthrough, indicating the complementarity of both methods. Also, it is noticeable the comparatively small amount of scenarios that succeeded in both tests. This is yet another indicative of the relevance of testing usability in such tools, given their natural complexity and importance to Computational Linguistics. As it will be discussed further in this article, this is not mainstream behaviour in the field.

Perhaps inevitably, during the usability tests some functional problems (*i.e.* behaviour that was not in accordance to the tool's description in its user manual) were also uncovered, along with some places for improvement. As a result, two additional fixes were executed in TSeg, 12 in MetaAnn, five in its generated tool, and five in AgreeCalc. These changes, together with the usability corrections made, constitute real improvements to each tool's previous version, so as to make these tools more suitable for usage by the community. In what follows, we will describe our toolset in more detail, pointing out the main changes made to it.

3.1. TSeg

TSeg was primarily built to break text inputs into basic units of annotation (akin to Mueller and Strube [2001]'s markables and what Lenzi et al. [2012] calls Minimum Markable Unit), either automatically (*e.g.* by splitting the text into words, sentences and paragraphs) or through its manual segmentation into units of arbitrary length. In doing so, TSeg also aims at complementing the various existing tools (*e.g.* [Müller and Strube 2006, Lohr et al. 2019]) which need previously segmented texts to work. Since its introduction to the community in 2012 [Rodrigues et al. 2012], TSeg was internationalised[4] and improved, resulting in its first stable version. This version was then assessed according to its usability, thereby leading to its current version, as described here.

Written in Java, TSeg can be run in all computing platforms and architectures supporting the Java Virtual Machine (JVM) in its 7.0 version or higher, with the Java Runtime Environment (JRE) installed. As an additional feature, TSeg was built as a stand-alone application, thereby having no need for a network connection. This, in turn, makes its use convenient by annotators who do not wish to worry about such technical details as network signal, delays and the like. Nevertheless, TSeg presents the inconvenience of demanding its users to somehow send the annotated data (*i.e.* the files edited through TSeg) back to researchers. This is a trade-off that we are afraid cannot be helped, even though we still believe it pays off to leave annotators' minds out of the aforementioned issues.

3.1.1. Data Modelling: Representing Annotations

Besides the usability corrections and its translation to English, another improvement to TSeg took place in its representation language. In this sense, despite the efforts in the search for a linguistic codification standard (*e.g.* Przepiórkowski and Bański [2009], Verhagen [2010]), no single coding scheme has so far emerged as the *de facto* representation language for the annotation of written corpora. There is, however, some understanding that, amongst other things, this format should preferably be compatible with XML, thereby making it platform independent [Müller and Strube 2006, O'Donnell 2008], and allow for new annotations to be added independently, that is there is a preference for stand-off annotation (*e.g.* Müller and Strube [2006], O'Donnell [2008], Verhagen [2010]).

Originally implementing an annotation format of its own, TSeg was modified so as to work with the XML-like language described in Roman [2013]. Its data model corresponds then to a set of plain text files tagged so as to represent the segmentation defined by its user. Each segmented file begins with a <document> tag, with a corre-

[4]Currently, it presents an interface in Portuguese and English

sponding </document> at the end of the file. Segments are enclosed within <UNIT> and </UNIT> tags (akin to the <word> and </word> tags used in Müller and Strube [2006]), as illustrated in Figure 2, with each UNIT holding an identification number, so it can be connected to other units (be them in the same file or in other files) in a stand-off manner.

Figure 2. XML-like coding in TSeg.

In Figure 2, one can see that some units are embedded into others (*e.g.* unit_1 is fully embedded into unit_0, and unit_3 is partially embedded into unit_2). This is the only feature that takes our coding scheme apart from a pure XML codification, which does not allow for such embeddings. Even though we could have achieved the same effect by adding an extra stand-off layer to the model, we though it would unnecessarily add complexity to the coding scheme. Along with their identification number, some units also carry the "IND" label. This is a special mark used to flag that these units, although embedded into others, are independent of their hosting unit. "IND" should be used when some text spam builds up a segment which bears no relation to the larger segment that contains it, such as when parentheses are added to some text, for example. They represent then discontinuities (or interruptions) in the regular flow of the text, and must be interpreted according to the phenomenon under study.

Through this procedure, our model covers all segmentation properties pointed out by Reidsma et al. [2005], to wit:

- *Superposition*: TSeg allows for overlapped segments, in which two or more segments share the same text span. In this sense, embedded segments are but segments in which one fully superposes the other;
- *Interleaving*: occurs when spans belonging to two or more independent segments are partially ordered in an interleaved way. In TSeg, interleaving can be codified by giving the same identification to multiple (sub-)segments;
- *Discontinuities*: discontinuous segments can be determined either through the definition of a segment and posterior definition of an embedded segment within it

(thereby removing part of the text in the hosting segment), or through the definition of separated segments with the same identification;

- *Multiple Annotations*: by forcing each segment to have a unique identification in the corpus, TSeg allows for their annotation with more than one element in a stand-off fashion. Multiple annotations can then be assigned to the same segment through this identification;
- *Input Coverage*: the segmentation may or may not cover all of TSeg's input, thereby allowing for some text spans to be left out; and
- *Segment Size*: segments may vary in size, ranging from words to arbitrary text spans.

3.1.2. Functionalities and Improvements

As soon as the input (raw) text is loaded into TSeg, it is automatically annotated with some default metadata tags, such as identifiers for the document, annotation scheme, source-text, source corpus and annotator. The user can then edit the document, selecting the desired text span, clicking on it and then choosing the appropriate action (Figure 3). Once defined some units, the user can also remove them, as shown in this figure[5]. As an alternative to visualising XML tags, the user can opt for a representation where segments are assigned colours. Both views are illustrated in Figure 4.

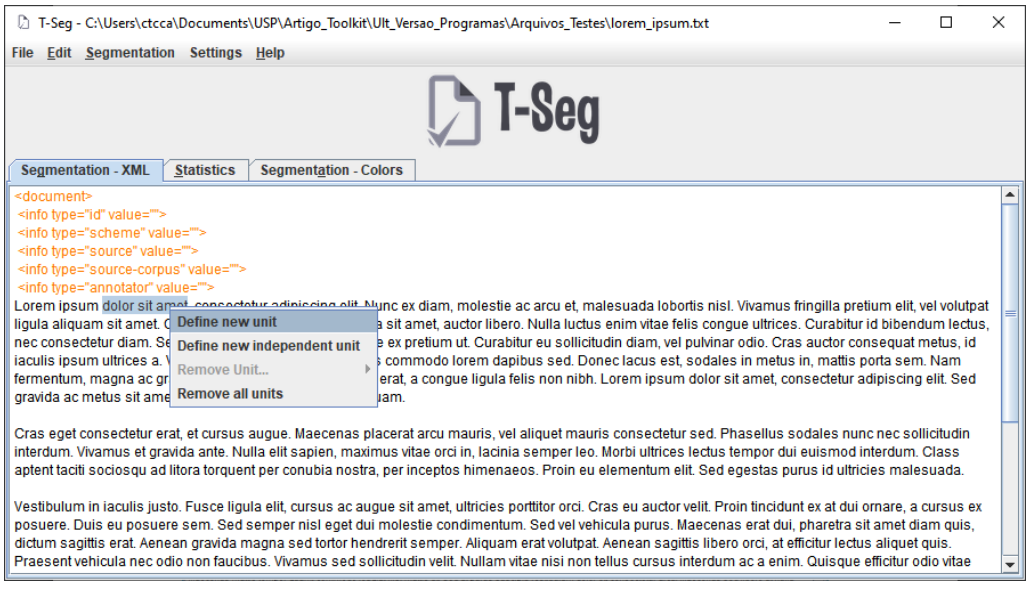


Figure 3. Defining an annotation unit.

Within TSeg, the user can also choose to automatically segment the input text into words, sentences or paragraphs. Sentences are taken to be text spans limited by full stop, exclamation or question marks. Paragraphs, in turn, are collections of sentences ending in a new-line character. Finally, besides segmenting the text, TSeg also gives some basic statistics about it, such as the total amount of segments (*i.e.* words, sentences, paragraphs or user defined units), along with the amount of overlapped units. This information can then be used by the researcher to better understand the properties of the corpus at hand.

[5]Even though, in the figure this option is disabled, since no unit has been defined.

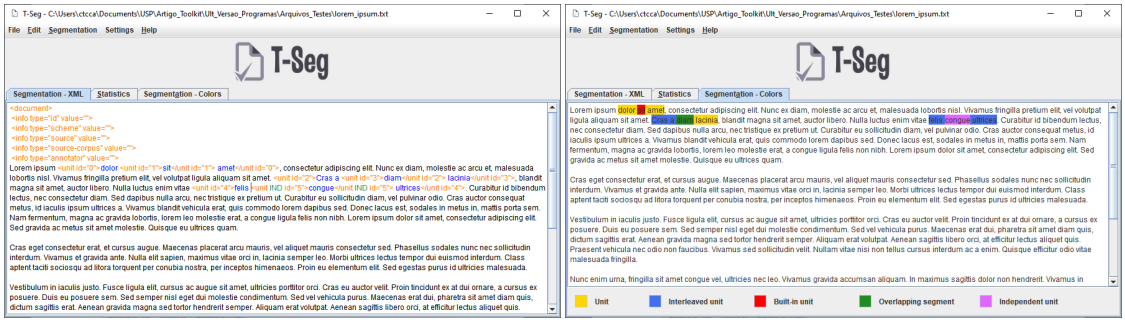


Figure 4. Possible views of TSeg's segmentation.

Since its beta version [Rodrigues et al. 2012], TSeg has undergone some major modifications. These were:

- The changing of its coding format, as described in Section 3.1.1;
- The double view of the annotation, through XML tags or colour (Figure 4); and
- Its internationalisation, with the addition of an English translation of its interface.

Besides these changes, we have also carried out an usability analysis of the system and corrected almost all raised problems (*cf.* Tables 1 and 2). The only exception was the issue related to the absence of a system icon to TSeg. On this regard, even though we have actually created an icon and added it to the program interface, it could not be assigned to the program in the Operating System, since it is distributed as a *.jar* file, which has an icon of its own. Changing this icon would reflect in all other Java applications in the system.

Finally, our usability analysis also highlighted some functional problems, such as bugs in the colour representation of units for example, which were corrected. Additionally, and as a way to sort out the usability problems we found, some new functionalities were added to TSeg, such as:

- The automatic segmentation of all files in a directory (in the previous version this was done on a documentwise basis);
- The inclusion of an "undo" choice for the user;
- The possibility of removing more than one unit in a single go (this was done unit by unit in the previous version); and
- The parametrisation of colour choice, whereby the user can now choose a palette to be used for segment representation.

3.2. MetaAnn

One possible solution to the problem of the high cognitive load demanded by general-use annotation tools, along with the cost, in terms of time and resources, imposed by the development and adaptation of *ad hoc* tools, is to make this development as automatised as possible, through some meta-tool capable of automatically generating the source code for simpler *ad-hoc* annotation tools. Within this framework, all a researcher needs to do to apply a new annotation scheme to some corpus is to define its details at this meta-tool's interface, with no need for programming, as is the case with some applications (*e.g.* Day et al. [2004], Lenzi et al. [2012]). The meta-tool would then be responsible for building

the annotation program that final annotators would use to apply the defined scheme to the target corpus, thereby tailoring it to this scheme and corpus, and so reducing the cognitive load imposed to them.

This reduction, in turn, makes the generated tool suitable for the use by non technically-specialised annotators, *i.e.* people who might contribute to the annotation but that would nevertheless struggle to deal with complicated annotation tools. Motivated by these factors, and designed for non-expert use, MetaAnn fill in these roles by (i) furnishing a Graphical User Interface (GUI), as opposed to relying on XML or tab-separated configuration files (*e.g.* Verhagen [2010], Petasis [2012]), through which researchers can define categories and values to be applied to the target-corpus, with no need for actual programming; and (ii) building an annotation tool, written in Java, with a "cleaner" GUI, so as to not demand from its users (*i.e.* the final annotators) knowledge beyond what is necessary for the application of the defined annotation scheme to the corpus.

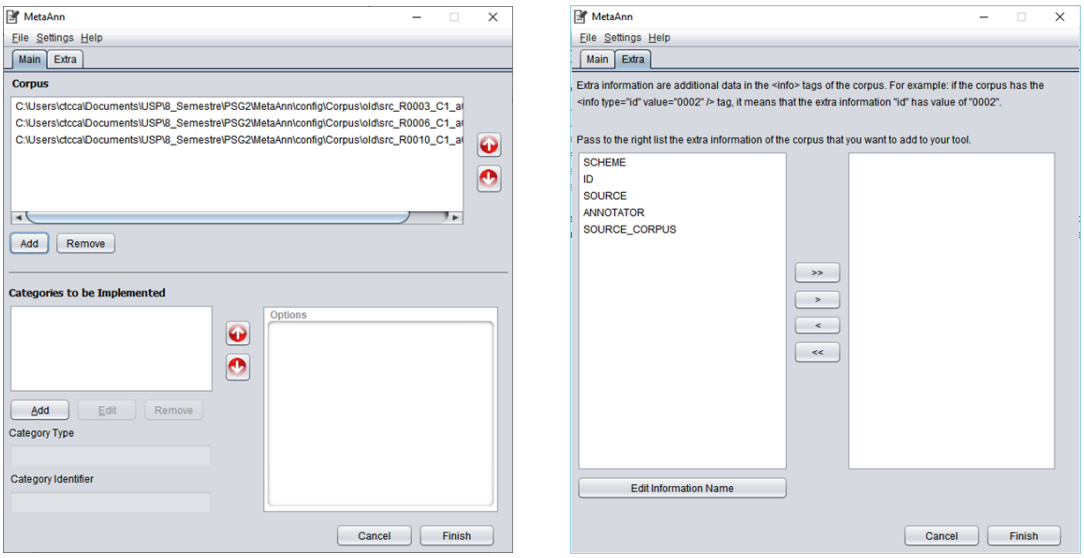
3.2.1. Functionalities

MetaAnn builds from the assumption that the researcher already has a *corpus* of text documents at hand, codified according to the pattern defined in Roman [2013], and already segmented in basic units of annotation (*i.e.* a corpus as output by TSeg). MetaAnn's main objective is then to generate an *ad-hoc* tool that allows final annotators to apply some researcher-defined annotation scheme to this segmented corpus. Within it, these schemes are coded as a series of values organised in categories. This organisation is then hard coded at the GUI of the tool generated by MetaAnn, thereby resulting in an annotation tool tailored to that specific scheme.

When running MetaAnn for the first time, researchers must load some examples of the corpus to be annotated (Figure 5a), so as to allow it to obtain a list of all metadata defined in the corpus. During this first step, MetaAnn also lets the researcher determine which of these metadata are to be visualised in the generated tool' GUI (Figure 5b). It is important to notice that all metadata must be already present in the *corpus* (*i.e.* they must have been added during the segmentation step with TSeg, for example). As such, one cannot create new metadata, since this is an interface feature only. Researchers can nevertheless define how they are to be shown at the tool's interface, by renaming them (by clicking on *Edit information name* in Figure 5b).

Once finished with this initial setting, the researcher can proceed to the definition of the annotation scheme properly, by adding the categories that build up the scheme, as shown in the bottom half of Figure 5a. Within this interface, one can also edit and remove the already defined categories, besides visualising some of their details (such as existing option values, for example). To add a category, researchers are presented a different interface (Figure 6), where they must provide details regarding the added category, such as its name (as it will appear at the generated tool's interface), its identifier (to be used in the source code in all elements related to this category, thereby allowing for its prompt identification, should any modification be necessary), and its type (either mutually exclusive options, multiple selection list, or free text).

For the first two types, option values must be added (by clicking on *Add* in the in-



(a) Corpus characteristics.

(b) Adding metadata to the GUI.

Figure 5. MetaAnn's main interface.

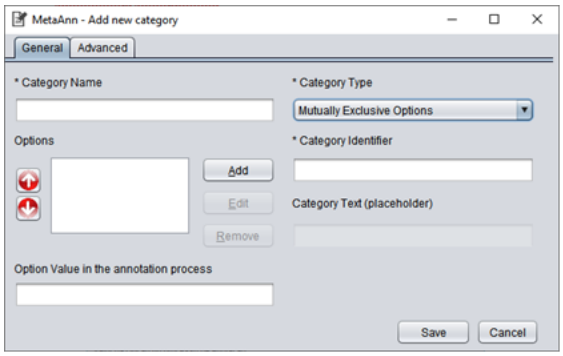


Figure 6. MetaAnn: adding a category.

terface). The main difference between these two types is that while the later allows for more than one value to be chosen, the former does not. For the third type (free text), researchers can also define an optional default text to appear at the annotator's interface (*Category Text*, in the figure), providing some guidance to the annotator. These details will define the graphical elements to be included at the generated tool's interface, along with their behaviour.

3.2.2. MetaAnn's Generated Tool

Once the annotation scheme is codified into MetaAnn, researchers can have it generate the *ad-hoc* tool that applies this scheme to the corpus at hand, by clicking on the *Finish* button in Figure 5a. MetaAnn will then write up the source code for the annotation tool (in Java), compile it and wrap it into a *.jar* file, which can then be used to run the tool at the annotator's computer[6]. It is important to notice that, at this point, researchers are free to edit the source code and recompile it, should any modification be necessary. We understand that, even though MetaAnn was designed so that editing the code is not strictly necessary, there are situations that cannot be helped. The generated tool can then be finally shipped, along with the corpus to be annotated, to final annotators.

Within the generated tool's interface, and similar to Verhagen [2010] and Lenzi et al. [2012], mutually exclusive options and regular lists are implemented with combo boxes, whereas free texts are represented by plain text boxes. Figure 7 shows an example of each possible category type. The tool's interface is split in three sections: context, annotation and browsing. The higher portion of the interface is dedicated to introducing the corpus to the annotator – the context (*cf.* Andreas et al. [2012]). Within it, annotators can see the plain document under analysis, along with the specific unit to be annotated and its identifier. At the bottom of the interface, one finds some buttons to navigate through the annotation units, along with buttons to save the annotation and exit the application.

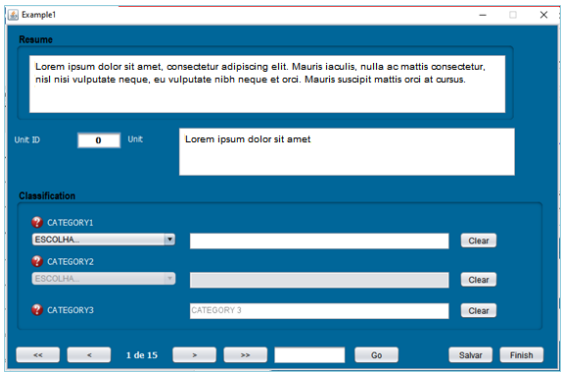


Figure 7. MetaAnn: generated tool's GUI.

Finally, the central part of the interface is reserved to the implementation of the annotation scheme itself (*i.e.* the part defined by the researcher when using MetaAnn). Another important feature of MetaAnn, and which is also shown in Figure 7, is the pos-

[6]Through issuing the command *java -jar jarfile* or by clicking on it, depending on the user's Operating System.

sibility of defining conditional categories, *i.e.* categories that only make sense if the annotator chooses a specific value for another category in the annotation scheme. Such categories initially show up as deactivated elements (*CATEGORY2* in the figure), becoming active (and so allowing annotators to interact with them) only upon the choice of a specific value for some predetermined category (in this example, *CATEGORY2* depends on *CATEGORY1*).

As the annotation proceeds, MetaAnn's generated tool will create (and update, if already present in the tool's corpus directory) a series of XML files holding the classification attributed to each annotation unit defined in the corpus (according to the format described in Roman [2013]). A different file will be created for each annotated document in the corpus, being stored in a folder named *Annotations*. Once finished the annotation, annotators must return this folder to the researcher, so it can be input to AgreeCalc for agreement analysis. At the end of this process, researchers will find themselves with a set of annotation folders, one for each annotator, that they can use for inspection or whatever else purpose.

3.2.3. Improvements to MetaAnn

Since its introduction to the community in 2013 [Missão and Roman 2013], MetaAnn has undergone a set of functional and usability tests, with the objective of making corrections, improvements and to make it accessible to a wider range of researchers, mainly through its internationalisation (originally it was available in Portuguese only). As with TSeg, of all highlighted usability problems (see Tables 1 and 2 for their overall amount), along with functionality issues, both in MetaAnn and its generated tool, the only issue that still remains to be solved is that of a proper system icon, given the already pointed out behaviour by Operating Systems regarding this point. All remaining problems were corrected.

Amongst the main improvements made, we highlight the following:

- The addition of "undo" and "cancel" functionalities, whereby users can step back in case of mistakes or give the current course of action away;
- The possibility of choosing where corpora can be found in the filesystem (originally, the user was forced to place them in a specific folder);
- The addition of feedback messages to the user, and the change of current messages, making them more informative; and
- The solving of the "cyclic dependency" problem, whereby two categories could originally be made dependent of each other.

As it turns out, even though MetaAnn did not present any heavy issue regarding its functionalities, it was found to be really hard to use by the inexperienced user. We hope these problems are settled now. As with TSeg, so was MetaAnn written as a Java standalone application, running in all computing platforms and architectures supporting the Java Virtual Machine (JVM) in its 7.0 version or higher, with the Java Runtime Environment (JRE) installed.

3.3. AgreeCalc

As a way to tackle the issue regarding the multiplicity of agreement indexes and consequent difficulties in carrying out cross-studies comparisons, AgreeCalc was designed so as to present researchers with a number of different options. Beyond simply allowing for multiple indexes to be calculated, amongst a list of six implemented so far, AgreeCalc also allows this calculation to be made with subsets of both data and annotators, thereby giving researchers a point and click way to explore the data at hand.

Since its introduction to the community by Alvares and Roman [2013], AgreeCalc has undergone a set of functional and usability tests, with the objective of correcting existing issues and making it accessible to a wider range of researchers. Of all failed scenarios pointed out in Tables 1 and 2, and following its fellow tools in the toolset, the only point that remains unsolved was that of a proper system icon, for the reasons already mentioned in the descriptions of TSeg and MetaAnn. Still, if users find this important, we have developed such an icon, which is delivered with the system, so they can assign it to AgreeCalc in their own systems, either by creating a wrapper script that calls AgreeCalc, and then assigning this icon to the script, or by assigning it to all *jar* files in their system.

As for the modifications we made, main improvements were:

- The internationalisation of its interface, with its current version available both in Portuguese and English (originally, AgreeCalc was available in Portuguese only);
- The addition of "undo" and "cancel" functionalities, whereby users can step back in case of mistakes or give the current course of action away;
- The addition of two other agreement indexes, building a total of six in AGreeCalc's current version;
- The inclusion of feedback notes, progress bars, and confirmation dialogues (for destructive, time consuming, or irreversible actions);
- The setting up or a user manual, something that was missing in its previous version; and
- Some bug fixes, regarding the reports generated by the tool, data corruption, and allowance of invalid input (such as empty sets of annotators and/or annotations, badly-formatted corpora etc).

Written in Java, AgreeCalc runs in all computing platforms and architectures supporting the Java Virtual Machine (JVM) in its 7.0 version or higher, with the Java Runtime Environment (JRE) installed.

3.3.1. Working with AgreeCalc

Taking as input a set of annotations by different annotators, codified as described in [Roman 2013] and, therefore, in conformity with the annotations output by MetaAnn's *ad-hoc* annotation tool, AgreeCalc presents researchers with a graphical interface to evaluate annotation schemes (Figure 8), with no demands regarding a deep knowledge about the details of the calculations involved in this process. To this end, the system relies on some Java libraries[7] to communicate with the R statistical package, which is then used to

[7]Namely, *irr* (`http://cran.r-project.org/web/packages/irr/irr.pdf`) and *RCaller* (`https://code.google.com/p/rcaller/`)

calculate the agreement indexes. In doing so, AgreeCalc furnishes a graphical interface to these R functions in which details regarding each coefficient are made transparent to its user.

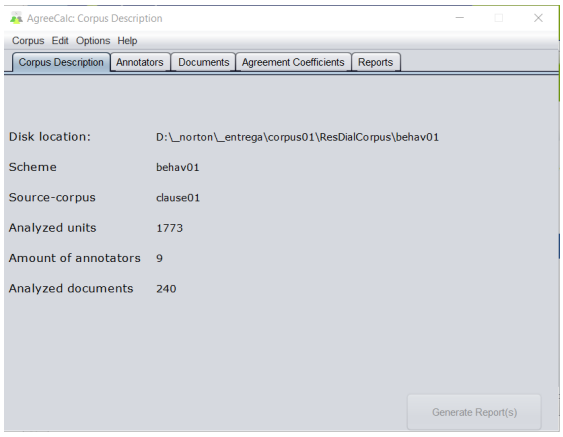


Figure 8. AgreeCalc's main interface.

Upon loading a set of annotations into AgreeCalc, the user is presented with a brief description of the selected corpus, as shown in Figure 8. This description contains some of the metadata present in the annotation files, such as the annotation scheme's name and the name of the corpus upon which the annotation was applied, along with the corpus location at the user's file system. AgreeCalc also calculates some statistics, presenting the user with the total amount of units and documents in the loaded corpus, as well as the number of annotators it detected.

From the main interface, researchers can determine which annotators are to be considered for agreement measures (Figure 9). To do so, the list of available annotators, which was automatically obtained from the metadata in the selected corpus, is shown at the right side of the interface, whereas the left side is reserved to the annotators ruled out from the calculation. Researchers are then free to explore the data, by selecting subsets of annotators and verifying their agreement, so as to determine whether some individual annotator, or even subgroups of annotators, diverge from the remaining ones.

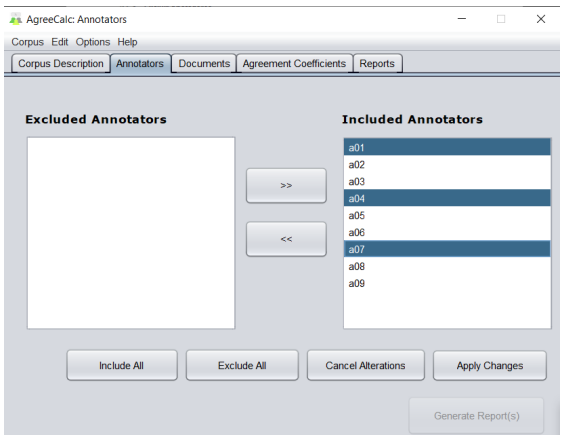
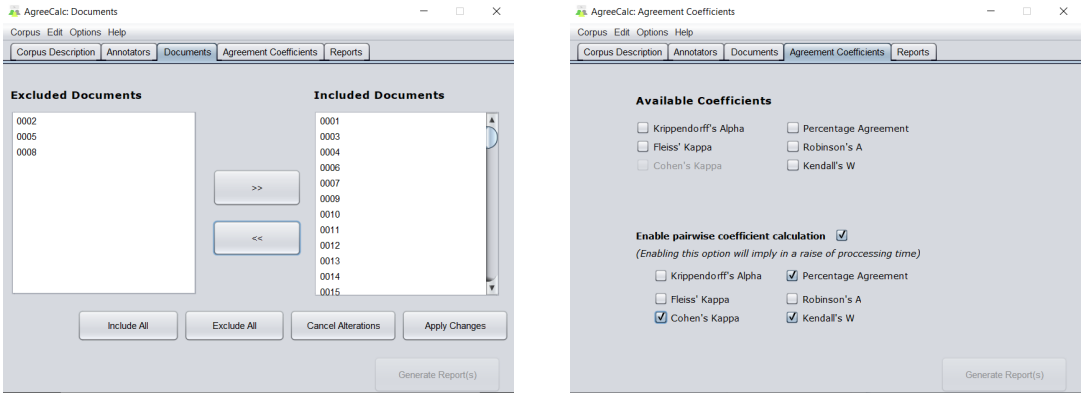


Figure 9. Annotators subsetting.

At the next tab (*Documents*), shown in Figure 10a, the researcher can apply the same

selection to documents, by determining which documents are to be considered for agreement and which will be excluded (the system defaults to calculating agreement for all documents in the corpus). Along with the selection of annotators, document subsetting gives the researcher a greater freedom to explore agreement across the data set, thereby allowing for the identification of highly subjective documents (as indicated by subsets of documents with very low inter-annotator agreement), or of annotators that apparently did not understand the task at hand (should a small set of them astray from the rest).



(a) Document subsetting. **(b) Agreement computation.**

Figure 10. Calculating agreement.

It is important to notice, however, that a document refers to a complete text, instead of some isolated annotation unit (*i.e.* the minimum unit upon which an annotation is applied). In its current version, AgreeCalc allows only for the selection of entire documents, in which case agreement is calculated only for the selected documents (Figure 10a) and annotators (filtered as shown in Figure 9). After selecting the documents and annotators that will be considered for the computation of agreement, researchers can proceed to the *Agreement Coefficients* tab (Figure 10b), where they are presented with a set of six different coefficients of agreement. As shown in the figure, depending on the number of annotators selected for this computation, some coefficients may not be available, as is the case with Cohen's κ, which is to be used with two annotators only.

Another useful feature of AgreeCalc is the possibility of having it calculate pairwise agreement according to any of the implemented coefficients (Figure 10b), in which case the system will calculate the value for the selected indexes for all possible permutations of annotators, taken two at a time. As a result, it will present, for each selected index, the pair of annotators with the lowest and highest agreement, also furnishing the mean agreement across all pairs. As for implemented coefficients, AgreeCalc's current version features Krippendorf's α, Cohen's κ, Fleiss' κ, Percentage of Agreement, Robinson's A and Kendall's W.

In AgreeCalc's last tab (Figure 11), researchers can define the output format of the document containing the calculated agreement results. Currently, available options are HTML, XML and PDF. Besides generating the final agreement report, researchers can also build two extra data sets, which are but annotation documents holding, for each unit in the corpus, the most popular classification across annotators, *i.e.* the category most frequently associated to each unit (its mode), regardless of the number of annotators

associating it to that unit; or the classification attributed by the majority of annotators (*i.e.* above 50% of the annotators), in which case units where majority could not be reached are left unclassified.

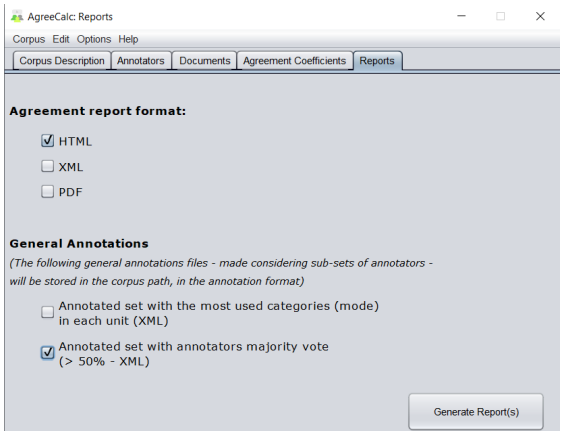


Figure 11. Gold standard generation.

These gold standards are codified as their source corpus, according to the pattern described in Roman [2013]. In following this procedure, researchers are not only presented with the numerical results of inter-annotator agreement, but also with a gold standard, built according to one of these metrics (mode or majority), which can then be used for future analyses, such as the identification of patterns and correlations, or even as a subsidy for future research, such as the training of Machine Learning models, for example.

4. Related Work

One of the first decisions one must make when designing a general-purpose text annotation tool, *i.e.* a tool capable of applying different user-defined annotation schemes to textual data, is whether it will be web-based or stand-alone. Web-based tools (*e.g.* Verhagen [2010], Lenzi et al. [2012], Andreas et al. [2012], Stenetorp et al. [2012], Bontcheva et al. [2013], de Castilho et al. [2014], Pérez-Pérez et al. [2015], Janssen [2016], Lohr et al. [2019]) have the straightforward advantage of allowing for a greater interaction amongst annotators, and even between annotators and researchers. So if this is a requirement to one's project, the decision is clear. Also, they have the additional advantage of imposing fewer requirements concerning memory usage, storage, and processing power on the annotator's side, being suitable for tools that should run in mobile phones for example.

On the other hand, they obviously require a full time live internet connection [Verhagen 2010, Petasis 2012], which sometimes must not be taken for granted, specially if annotators are doing their job while in movement (such as when travelling in trains, aeroplanes and the like), or if the population to be reached lives in more isolated areas. In an experimental design where some subtasks must be performed as a unit, thereby not allowing for interruptions during their execution, depending on such a connection will bring an extra threat to the validity of the experiment, to the extent that it increases the odds of annotators dropping out for technical reasons (*cf.* Roman et al. [2006]). But even when internet connection can be assumed, there are some issues regarding browser compatibility [Verhagen 2010, Petasis 2012], along with the setting up of a server with enough processing power, memory and storage space to handle requests by the annotators, let

alone the need for some maintenance service to keep it up and running (*cf.* Bontcheva et al. [2013]). These requirements translate into costs at the researcher's side.

Stand-alone tools (*e.g.* Mueller and Strube [2001], Day et al. [2004], Müller and Strube [2006], Ogren [2006]), on the other hand, do not suffer from these problems. Nevertheless, they have disadvantages of their own, such as the absence of network communication and cooperation amongst annotators and between annotators and researchers. If these are requirements for the annotation effort, such tools should not be considered. Although reducing the costs related to servers and services on the researcher's side, these are actually transferred to the user's side, that is the annotator. These costs, translated into processing and memory requirements, are nevertheless distributed amongst annotators, which might result in a burden they can afford. In this research, we opted for delivering a stand alone tool, so as to maximise the odds annotators would work independently of one another, besides reducing the drop-out risk when carrying out continuous subtasks that should not be interrupted.

Given the advantages and disadvantages of each approach, some tools (*e.g.* Petasis [2012]) try to get the best out of both worlds (with some researchers explicitly stating their intent to follow this same path, such as Lenzi et al. [2012]), by presenting themselves as stand-alone applications with network capabilities for storing data in centralised servers, along with instant messaging facilities for communicating events to all annotators. This is an interesting feature we do not possess, and which we believe might be added to future versions of our system. Nevertheless, we still believe this is a feature that must hold only between the researcher and the annotator, thereby not allowing for inter-annotator communication. Such a communication would certainly break the requirement of independence between annotators, making it hard to estimate the chance-related agreement – a necessary feature to the majority of existing agreement indices.

Whatever the tool, there seems to exist some common sense interpretation whereby being general-use implies being "all-in-one", that is it implies delivering everything both researcher and annotator might need for setting up the experiment, carrying it out, and analysing its results. This is a common feature to virtually all system cited above, even though many separate the roles through session passwords and the like (*e.g.* Verhagen [2010], Bontcheva et al. [2013], de Castilho et al. [2014], Janssen [2016]). Exceptions to this unstated rule are SYNC3 [Petasis 2012], which is based on the Ellogon platform [Petasis et al. 2002], thereby outsourcing project administration to this platform, and some stand-alone tools (*e.g.* Day et al. [2004]), which are pre-packed and then sent to final annotators. These, however, depend either on manually pre-built configuration files or additional programming to generate the ad-hoc tools to be sent to final annotators. Within our toolset, this task is made more user-friendly, through MetaAnn.

In fact, our toolkit seems to fit an apparently contradictory class: that of general-use-ad-hoc tools. It is general-use because researchers can use it for many different text annotation tasks and schemes, and it is ad-hoc because the final tool, which will be used by annotators, is tailored to the specific task at hand. In a sense, the toolkit might also be classified as a collaborative toolset, that is a set of simpler tools that integrate and work together for a common annotation purpose. In doing so, the toolset presents the advantages, at the annotator side, of delivering an annotation tool focused at the annotation task proper, with no other external interference (as provided by interfaces allowing for differ-

ent roles in the annotation process), and clearly separating all steps (and associated roles) in the annotation process. At the downside, the existence of different tools might bring an extra burden at the researcher's side, who has to deal with these tools as opposed to a single system with all information at the reach of their hands. Still, we believe this is a price worth paying.

Regarding the analysis of annotation quality, most of the above cited systems present some way to calculate inter-annotator agreement (*e.g.* Ogren [2006], Verhagen [2010], Andreas et al. [2012], Petasis [2012], Lenzi et al. [2012], Bontcheva et al. [2013], de Castilho et al. [2014], Pérez-Pérez et al. [2015], Lohr et al. [2019]). The problem, however, is that they hardly agree on which indexes should be presented, with Cohen's κ (*e.g.* Petasis [2012], Lenzi et al. [2012], Andreas et al. [2012], Bontcheva et al. [2013]) and f-measure (*e.g.* Pérez-Pérez et al. [2015], Bontcheva et al. [2013], de Castilho et al. [2014]) figuring as the preferred ones. Krippendorff's α (*e.g.* Lohr et al. [2019]), Fleiss' κ (*e.g.* Lenzi et al. [2012]), Dice's coefficient (*e.g.* Lenzi et al. [2012]) and percent agreement (*e.g.* Petasis [2012]) are sometimes also calculated.

Still, none of the analysed tools seems to deliver more than three different indexes, with presenting a single alternative to the user being the apparent rule (*e.g.* Andreas et al. [2012], de Castilho et al. [2014], Pérez-Pérez et al. [2015], Lohr et al. [2019]). As already argued, this reduces the degree to which different studies can be compared regarding inter-annotator agreement, for different indexes have different assumptions and, as such, do not necessarily behave the same way in face of the same dataset (see Artstein and Poesio [2008] for a detailed analysis of these indexes). Within our toolset, this problem is overcome by presenting researchers with a broader range of choices to calculate inter-annotator agreement (to wit, Krippendorf's α, Cohen's κ, Fleiss' κ, Percentage of Agreement, Robinson's A and Kendall's W), thereby increasing the odds that research data analysed with our tool can be compared to others.

Finally, although usually considered an important feature to be taken into account when choosing an annotation tool (*cf.* Dipper et al. [2004]), usability analysis does not seem to be of much concern by current research. In fact, even among the systems that claim it to be one of their upside features (*e.g.* Geertzen and Bunt [2006], Bertran et al. [2008], Lenzi et al. [2012]), we could find no detailed account of how it was measured (*i.e.* which approach was taken to analyse usability), and what it ultimately measured (*i.e.* what features were considered when analysing the tool at hand). In contrast, here we give details of the procedure we followed when analysing our toolset's usability, such as the adopted approach and the analysed variables, thereby allowing for a better comprehension, by the reader, of the toolset's strengths and weaknesses.

5. Discussion

As discussed in the previous section, our toolset stands out from current research in that it splits the main annotation steps (*i.e.* text segmentation, annotation and assessment) into a pipeline of different tools. In doing so, we not only allow for each of these steps to be tried and assessed independently of the others, but also make it easier for researchers to apply only a subset of these tools to their own research, provided that input data is tagged following the same pattern accepted by the toolset. This could be an interesting feature for existing tools that take previously segmented data as input (*e.g.* Lohr et al. [2019]),

or researchers that intend to compare their results to others through different agreement metrics (*e.g.* de Arruda et al. [2015]). Even though both cases require a mapping between our toolset data representation format and that accepted by the tools used in the research, this mapping will most certainly require much less effort than building tools specifically to this end.

In building annotation tools that can be independently distributed to annotators, we allow the toolset to be used not only in annotation efforts, but also in annotation experiments where independence between subjects is a requirement. Moreover, the separation between annotation and segmentation, as opposed to performing both tasks in a single annotation step, allows for different annotation schemes to be applied to the same segmentation. This is a useful feature if one has already settled on a segmentation (or is bound to an existing one) and wishes to test different schemes (perhaps unifying categories or splitting existing categories into subcategories, for example) in the search for a better inter-annotator agreement. This is an issue that has already been raised in the related literature (*e.g.* Craggs and Wood [2004], Roman and Carvalho [2010]).

Another small but sometimes held important feature of our system is the fact that it can automatically segment text according to different granularities (to wit words, sentences and paragraphs), treating them as basic units of annotation. These are then dealt with indiscriminately by the annotation tool, thereby not introducing any extra effort if one is to deal with long text stretches, as pointed out by Lohr et al. [2019]. Also, the possibility of defining conditional categories makes the system suitable for research that has hierarchically structured labels as a major requirement (*e.g.* Lohr et al. [2019]). In this case, however, instead of showing some drop-down menu with labels and sub-labels (as done in Lohr et al. [2019]), whereby the user must follow through the hierarchy down to the desired label, we require a choice to be made at each level of the hierarchy. Although this may represent extra work to the annotator it also allows the researcher to identify where in the hierarchy disagreement occurred.

At the downside, and as pointed out by Grover et al. [2006], the fact that annotation is separated from segmentation makes it harder to recover from segmentation mistakes during annotation. This, however, was a design choice we made, for we believe it to be a small price given the benefits of not introducing any confounding variables. To reduce the risk of this kind of problem, it is advised to researchers to test segmentation schemes by having a small set of annotators annotate the data, and then analysing the source of disagreement, so as to determine whether it comes from segmentation or annotation itself. Still, we understand that there are situations that cannot be helped, to which our tool might not be the best choice. Finally, the lack of a proper communication between researchers and annotators makes it harder to make interventions to the annotation experiment, or even to raise a consensus discussion between researchers and annotators. These are features left for future improvements.

6. Conclusion

In this article, we described a toolset designed for general-purpose text annotation tasks. The toolset comprises a pipeline of three independent but interconnected tools, covering all steps throughout the annotation process, from data segmentation to data annotation to annotation evaluation. These tools were primarily built to address three main

issues found in current general-purpose annotation tools, to wit (i) the potential confounding of variables, as a result of having annotators do both segmentation and annotation in a single step, and so making it hard to determine which variable might be responsible for disagreement amongst annotators; (ii) the cognitive load imposed to annotators, by presenting them a tool with different roles and tasks altogether in a single interface (sometimes even allowing them access to information beyond the task at hand), potentially limiting the use of the tool to annotators who are more familiar with such systems; and (iii) the difficulty in comparing one study to others, when different agreement indexes are reported.

Within our toolset, these issues are approached by (i) having different tools perform data segmentation and annotation separately; (ii) giving researchers a tool where they can define and generate *ad hoc* tools, tailored to specific annotation schemes, to be distributed amongst annotators; and (iii) furnishing a way to calculate inter-annotator agreement according to six different indexes. Given its modularity, the toolset can be used both as a pipeline, whereby the output of one tool can be input to the next one, or for specific subtasks, such as segmenting texts into basic units of annotation (either manually or automatically), automatically generating annotation tools to specific schemes, or calculating agreement, according to different indexes, for subsets of annotations and annotators. The toolset lends itself then, among other things, to the quick prototyping and testing of annotation schemes, with no extra effort needed to the development of different tools to this end, also allowing for the study of the behaviour of different agreement indexes in the data set.

Although beta versions of each tool have already been introduced to the community in their Portuguese versions, the final integration of these tools has never been showed. More importantly, all tools in the set have undergone some major upgrades, being also evaluated according to their usability. This is something we found considerably rare in the current literature, raising some concerns, specially in face of the high rate of scenario failures showed by our toolset, as reported in Section 3. Main modifications to the toolset comprise the addition of new inter-annotator agreement indexes, the modification of their Graphical User Interfaces and corpus codification scheme, their internationalization through UTF-8 and their presentation both in English and Portuguese, whereby the user may choose the language adopted at the interface. Finally, all tools are currently distributed under GPL, being available for download at `https://github.com/NortonTR/ResDialTools`.

As for the toolset's limitations, one has to bear in mind that it was primarily designed for annotation experiments and, as such, it is not suited for annotation efforts where annotators must communicate, so as to settle on some annotation. The fact we have split segmentation and annotation apart makes the tool not suitable to tasks where these need to be made in a single step, such as in some prosody annotation tasks for example. Another drawback lies in the way annotators send their final results back to researchers. In its current version, the toolset does not allow for the automatic delivering of these results over the internet, so annotators must be instructed to find the annotated files and send them through e-mail, for example. Even though our original intention was not to force researchers into a specific set of steps that might add some prohibitive costs (such as setting up a centralised server to receive all annotations), we understand an upload facility could

be offered as an option to them, and leave it to future improvement.

Along similar lines, and as suggested by Lohr et al. [2019], it might be desirable to have some kind of automated monitoring mechanism, so researchers could check both on the overall stage of the annotation and annotator-specific progress. Such a mechanism might, for instance, record some meta information about the annotator's environment and behaviour, such as time spent on each annotation, operating system, devices etc. Following Bontcheva et al. [2013], another place for improvement might be to measure agreement between each annotator and the gold standard produced by the system, so as to have some way to determine how individual annotations are spread around this standard. Even though there are currently workarounds to achieve this, it might be interesting having this information at the produced report. Last, but not least, the toolset must provide a way to measure agreement at the segmentation phase (perhaps through Dice's coefficient, for example, as reported in Lenzi et al. [2012]). This is definitely an improvement to be made.

Acknowledgements

The authors would like to thank early adopters of this toolset, who have provided us with valuable feedback and suggestions for new features.

References

Alm, C. O., Roth, D., and Sproat, R. (2005). Emotions from text: Machine learning for text-based emotion prediction. In *Proceedings of HLT/EMNLP 2005*, Vancouver, Canada.

Alvares, A. R. and Roman, N. T. (2013). Agreecalc: Uma ferramenta para análise da concordância entre múltiplos anotadores. In *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology (STIL 2013)*, pages 1–10, Fortaleza, CE – Brazil.

Andreas, J., Rosenthal, S., and McKeown, K. (2012). Annotating agreement and disagreement in threaded discussion. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, pages 818–822, Istanbul, Turkey.

Apostolova, E., Neilan, S., An, G., Tomuro, N., and Lytinen, S. (2010). Djangology: A light-weight web-based tool for distributed collaborative text annotation. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Artstein, R. and Poesio, M. (2008). Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596.

Atkinson, J. and Munoz, R. (2013). Rhetorics-based multi-document summarization. *Expert Systems with Applications*, 40:4346–4352.

Bayerl, P. S. and Paul, K. I. (2011). What determines inter-coder agreement in manual annotations? a meta-analytic investigation. *Computational Linguistics*, 37(4):699–725.

Bertran, M., Borrega, O., Recasens, M., and Soriano, B. (2008). Ancorapipe: A tool for multilevel annotation. *Procesamiento del Lenguaje Natural*, 41:291–292.

Bontcheva, K., Cunningham, H., Roberts, I., Roberts, A., Tablan, V., Aswani, N., and Gorrell, G. (2013). Gate teamware: a web-based, collaborative text annotation framework. *Language Resources and Evaluation*, 47(4):1007–1029.

Callaway, C., Dzikovska, M. O., Moore, J. D., Reitter, D., and Zinn, C. (2005). D11: Corpus collection and specification. Technical report, The LeActiveMath Consortium.

Carletta, J. (1996). Assessing agreement on classification tasks: The kappa statistic. *Computational Linguistics*, 22(2):249–254.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Education and Psychological Measurement*, 20(1):37–46.

Craggs, R. and Wood, M. M. (2004). A two dimensional annotation scheme for emotion in dialogue. In *AAAI Spring Symposium: Exploring Attitude and Affect in Text: Theories and Applications*, Stanford, USA. Technical Report SS-04-07.

Craggs, R. and Wood, M. M. (2005). Evaluating discourse and dialogue coding schemes. *Computational Linguistics*, 31(3):289–296.

Day, D., McHenry, C., Kozierok, R., and Riek, L. (2004). Callisto: A configurable annotation workbench. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation. (LREC 2004)*, pages 2073–2076, Lisboa, Portugal.

de Arruda, G. D., Roman, N. T., and Monteiro, A. M. (2015). An annotated corpus for sentiment analysis in political news. In *Proceedings of the 10th Brazilian Symposium in Information and Human Language Technology (STIL 2015)*, pages 101–110, Natal, RN – Brazil.

de Castilho, R. E., Biemann, C., Gurevych, I., and Yimam, S. M. (2014). Webanno: a flexible, web-based annotation tool for clarin. In *Proceedings of the 3rd CLARIN ERIC Annual Conference (CAC2014)*, Soesterberg, The Netherlands.

de Loupy, C., Guégan, M., Ayache, C., Seng, S., and Moreno, J.-M. T. (2010). A french human reference corpus for multi-document summarization and sentence compression. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta.

Dipper, S., Götze, M., and Stede, M. (2004). Simple annotation tools for complex annotation tasks: an evaluation. In *Proceedings of the LREC Post-Conference Workshop on XML-Based Richly Annotated Corpora*, Lisbon, Portugal.

Dumas, J. S. and Redish, J. C. (1999). *A Practical Guide to Usability Testing*. Intellect.

Eugenio, B. D. and Glass, M. (2004). The kappa statistic: A second look. *Computational linguistics*, 30(1):95–101.

Fort, K., François, C., Galibert, O., and Ghribi, M. (2012). Analyzing the impact of prevalence on the evaluation of a manual annotation campaign. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 224–230, Istanbul, Turkey. European Language Resources Association (ELRA). ACL Anthology Identifier: L12-1233.

Geertzen, J. and Bunt, H. (2006). Measuring annotator agreement in a complex hierarchical dialogue act annotation scheme. In *Proceedings of the 7th SIGdial Workshop on Discourse and Dialogue*, pages 126–133, Sydney, Australia. Association for Computational Linguistics.

Grouin, C., Rosset, S., Zweigenbaum, P., Fort, K., Galibert, O., and Quintard, L. (2011). Proposal for an extension of traditional named entities: From guidelines to evaluation, an overview. In *Proceedings of the Fifth Linguistic Annotation Workshop (LAW V)*, pages 92—100, Portland, Oregon, USA.

Grover, C., Matthews, M., and Tobin, R. (2006). Tools to address the interdependence between tokenisation and standoff annotation. In *Proceedings of the 5th Workshop on NLP and XML (NLPXML-2006): Multi-Dimensional Markup in Natural Language Processing*, Trento, Italy.

Gut, U. and Bayerl, P. S. (2004). Measuring the reliability of manual annotations of speech corpora. In *Proceedings of Speech Prosody 2004*, pages 565–568, Nara, Japan.

Hasler, L. (2007). From extracts to abstracts: Human summary production operations for computer-aided summarisation. In *Proceedings of the RANLP 2007 Workshop on Computer-Aided Language Processing (CALP)*, pages 11–18, Borovets, Bulgaria.

Janssen, M. (2016). Teitok: Text-faithful annotated corpora. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, Portorož, Slovenia.

Jeffries, R., Miller, J. R., Wharton, C., and Uyeda, K. M. (1991). User interface evaluation in the real world: A comparison of four techniques. In *Proceedings of the ACM CHI 91 Human Factors in Computing Systems Conference*, pages 119–124, New Orleans, USA.

Jordan, P. W. (1998). *An Introduction To Usability*. Taylor and Francis.

Krippendorff, K. (2004). *Content Analysis: An Introduction to its Methodology*. SAGE, 2nd edition .

Lenzi, V. B., Moretti, G., and Sprugnoli, R. (2012). Cat: the celct annotation tool. In *Proceedings of the eighth international conference on Language Resources and Evaluation (LREC 2012)*, Istanbul, Turkey.

Lewis, C. and Wharton, C. (1997). *Hndbook of Human-Computer Interaction*, chapter Cognitive Walkthroughs, pages 717–732. Elsevier.

Lohr, C., Kiesel, J., Luther, S., Hellrich, J., Stein, B., and Hahn, U. (2019). Continuous annotation quality control, support for hierarchically structured label sets and long-segment annotation with w at -s l 2.0. In *Proceedings of the 13th Linguistic Annotation Workshop*, pages 215–219, Florence, Italy.

Mathet, Y., Widlöcher, A., Fort, K., François, C., Galibert, O., Grouin, C., Kahn, J., Rosset, S., Zweigenbaum, P., and Zweigenbaum, P. (2012). Manual corpus annotation: Giving meaning to the evaluation metrics. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING 2012): Posters*, pages 809–818, Mumbai, India. The COLING 2012 Organizing Committee.

Missão, T. E. I. and Roman, N. T. (2013). Metaann: Um gerador de ferramentas para anotação de textos. In *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology (STIL 2013)*, pages 11–20, Fortaleza, CE – Brazil.

Mitkov, R., Orasan, C., and Evans, R. (1999). The importance of annotated corpora for nlp: the cases of anaphora resolution and clause splitting. In *Proceeding of "Corpora and NLP: Reflecting on Methodology Workshop", TALN'99*, pages 60 – 69, Cargese, Corse.

Mueller, C. and Strube, M. (2001). Mmax: A tool for the annotation of multi-modal corpora. In *Proceedings of the 2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, Seattle, USA.

Müller, C. and Strube, M. (2006). Multi-level annotation of linguistic data with MMAX2. In Braun, S., Kohn, K., and Mukherjee, J., editors , *Corpus Technology and Language Pedagogy: New Resources, New Tools, New Methods*, pages 197–214. Peter Lang, Frankfurt a.M., Germany.

Müller, C. and Strube, M. (2003). Multi-level annotation in mmax. In *Proceedings of the Fourth SIGdial Workshop of Discourse and Dialogue (SIGDIAL)*, pages 198–207, Sapporo, Japan.

Nielsen, J. (1993). *Usability Engineering*. Academic Press.

Nielsen, J. (1994). *Usability Inspection Methods*, chapter Heuristic evaluation. John Wiley & Sons.

O'Donnell, M. (2008). The UAM corpustool: software for corpus annotation and exploration. In *Proceedings of the XXVI Congreso de AESLA*, Almeria, Spain.

Ogren, P. V. (2006). Knowtator: A protégé plug-in for annotated corpus construction. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Demonstrations*, pages 273–275, New York City, USA.

Orăsan, C. (2003). Palinka: A highly customisable tool for discourse annotation. In *Proceedings of the Fourth SIGdial Workshop of Discourse and Dialogue (SIGdial2003)*, page 39–43, Sapporo, Japan.

Petasis, G. (2012). The sync3 collaborative annotation tool. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 363–370, Istanbul, Turkey. European Language Resources Association (ELRA). ISBN 978-2-9517408-7-7.

Petasis, G., Karkaletsis, V., Paliouras, G., Androutsopoulos, I., and Spyropoulos, C. D. (2002). Ellogon: A new text engineering platform. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2002)*, pages 72–78, Las Palmas, Canary Islands, Spain.

Przepiórkowski, A. and Bański, P. (2009). Which xml standards for multilevel corpus annotation? In *Proceedings of the 4th Language and Technology Conference, LTC 2009, Poznan*, pages 400–411, Poznan, Poland.

Pustejovsky, J. and Stubbs, A. (2012). *Natural Language Annotation for Machine Learning*. O'Reilly Media, 1 edition . ISBN: 978-1-4493-0666-3.

Pérez-Pérez, M., Glez-Peña, D., Fdez-Riverola, F., and Lourenço, A. (2015). Marky: A tool supporting annotation consistency in multi-user and iterative document annotation projects. *Computer Methods and Programs in Biomedicine*, 118:242–251.

Radev, D., Allison, T., Blair-Goldensohn, S., Blitzer, J., Çelebi, A., Dimitrov, S., Drabek, E., Hakim, A., Lam, W., Liu, D., Otterbacher, J., Qi, H., Saggion, H., Teufel, S., Topper, M., Winkel, A., and Zhang, Z. (2004). Mead — a platform for multidocument multilingual text summarization. In *Proceedings of the 4th International conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal.

Reidsma, D., Jovanovic, N., and Hofs, D. H. W. (2005). Designing annotation tools based on properties of annotation problems. In *Proceedings of the 5th International Conference on Methods and Techniques in Behavioral Research*, Wageningen, The Netherlands.

Rodrigues, F., Semolini, R., Roman, N. T., and Monteiro, A. M. (2012). Tseg – a text segmenter for corpus annotation. In *Proceedings of the VIII Brazilian Symposium on Information Systems (SBSI 2012)*, volume 1, pages 700–709, São Paulo, SP – Brasil. ISSN: 2177-885X.

Roman, N. T. (2013). Resdial – coding description (v.1.0). Technical Report PPgSI-003/2013, EACH-USP, São Paulo, SP – Brazil.

Roman, N. T. and Carvalho, A. M. B. R. (2010). A multi-dimensional annotation scheme for behaviour in dialogues. In Kuri-Morales, A. and Simari, G. R., editors , *Proceedings of the 12th Ibero-American Conference on Artificial Intelligence (IBERAMIA 2010)*, volume 6433 of *Lecture Notes in Computer Science*, pages 386–395, Bahía Blanca, Argentina. Springer Berlin Heidelberg. Print ISBN: 978-3-642-16951-9 Online ISBN: 978-3-642-16952-6.

Roman, N. T., Piwek, P., and Carvalho, A. M. B. R. (2006). A web-experiment on dialogue classification. In Rezende, S. O. and da Silva Filho, A. C. R., editors , *Pro-*

ceedings of the Fourth Workshop in Information and Human Language Technology (TIL'2006), Ribeirão Preto, SP – Brazil. ICMC-USP.

Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S., and Tsujii, J. (2012). Brat: a web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107, Avignon, France.

Verhagen, M. (2010). The brandeis annotation tool. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, pages 3638–3643, Valletta, Malta.