Engineering Annotation Usability
Toward Usability Patterns for Linguistic Annotation Tools

Inaugural-Dissertation zur Erlangung der Doktorwürde der
Fakultät für Sprach-, Literatur- und Kulturwissenschaften der
Universität Regensburg

Vorgelegt von
Manuel Burghardt
aus
Regensburg
2014

Erstgutachter: Prof. Dr. phil. Christian Wolff
Zweitgutachter: Prof. Dr. rer. soc. Rainer Hammwöhner

Dedicated to the memory of Johann Burghardt

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Dr. Christian Wolff, who encouraged me to start this dissertation in the first place, and who constantly offered me his support, advice and encouragement throughout the entire process. I have learned so much in the past years, and for this opportunity I am truly grateful. I also would like to thank Prof. Dr. Rainer Hammwöhner, for his helpful comments and stimulating discussions, which I always greatly appreciate.

I would like to thank my fellow doctoral students from our doctoral seminar, for their support and feedback. I also would like to thank all my friends and colleagues at the University of Regensburg – I cannot think of a more inspiring and fun place to work at. In particular, I would like to thank Tim and Patricia for interesting discussions about patterns and usability evaluation methods. I also would like to thank Alex and Kaiser for their technical support with web server and JavaScript issues, and Bella, Saskia and Flo for their help with my empirical studies. Thanks also to the fearless proofreaders Johannes, Patricia, and Julia.

I would like to thank my mother, who always supported me and who never questioned any choices I have made in my life.

Finally, I would like to thank Julia, who has provided me with an endless supply of fresh coffee, and who always was understanding and cheering for this project. I am grateful for your love and support, thank you <:3~

Abstract

This dissertation engages with the improvement of user interfaces for manual annotation tasks in a linguistic context. Eleven linguistic annotation tools are evaluated in order to derive hints for good and bad designs in this specific domain of application. The evaluation design is influenced by related work from the field of human handwritten annotation theory and linguistic annotation standards and practices. At the same time it builds upon established usability engineering concepts and usability testing methods. The results of the annotation tool evaluation are documented as usability patterns. While the identification of patterns is generally considered a rather vague and unstructured process, this dissertation presents a systematic approach for the integration of qualitative results from a series of heuristic walkthroughs and a usability pattern format. The twenty-six usability patterns identified in this work help tool developers to prevent common mistakes and pitfalls in the design of annotation tool interfaces. They also guide actual tool users to decide which tool to choose with regard to usability requirements.

Zusammenfassung

Diese Dissertation beschäftigt sich mit der Verbesserung von Benutzeroberflächen für manuelle Annotationsaufgaben in linguistischen Anwendungskontexten. In einer Evaluationsstudie mit insgesamt elf linguistischen Annotationswerkzeugen wurden wichtige Hinweise für gutes und schlechtes Interface-Design in dieser spezifischen Anwendungsdomäne identifiziert. Das Evaluationsdesign ist beeinflusst durch verwandte Arbeiten aus dem Feld der handschriftlichen Annotationstheorie (*human handwritten annotation theory*) sowie durch Standards und Praktiken der linguistischen Textannotation. Gleichzeitig baut die Studie auf etablierten Konzepten und Methoden des *Usability Engineering* und der Usability-Evaluation auf. Die Ergebnisse der Studie werden in Form von Usability-Patterns dokumentiert. Um den häufig als vage und unstrukturiert wahrgenommenen Prozess der Pattern-Identifikation besser nachvollziehbar zu machen, präsentiert diese Dissertation einen systematischen Ansatz für die Integration qualitativer *Heuristic Walkthrough*-Daten in ein bestehendes Usability-Pattern-Format. Die so identifizierten 26 Usability-Patterns helfen Tool-Entwicklern dabei typische Fehler beim Design von Annotationstools zu vermeiden. Gleichzeitig erlauben es die Patterns den Anwendern solcher Annotationstools geeignete Werkzeuge in Hinblick auf Usability-Anforderungen für spezifische Projekte auszuwählen.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Problem context

Digital annotations

Since the advent of the home computer, desktop publishing and the Internet, information is ubiquitously and abundantly available in digital form. Coping with this flood of information can be very challenging, or as Shillingsburg (2006, p. 11) sums it up: "It is easy to get lost or discouraged in the field of electronic texts". Digital annotations are an important means to make the daily flood of information manageable, as they allow us to add "invisible intelligence" (Ruecker et al., 2011, p. 27) to a resource (e.g. a text or an image), thus making implicit information explicitly available in machine-readable form. This digitally annotated data may then be accessed and processed by automatic information retrieval systems.

Manual annotation

Although there are approaches for the automatic annotation of digital data, manual annotation is still a crucial task[1], as humans are exceptionally good at identifying implicit information, using their previous knowledge to resolve complex semantics. While rule-based and statistical approaches have had significant success in some areas of text annotation, manual annotation and human judgment are still indispensable when it comes to semantic and pragmatic ambiguities. As manual annotation typically is a laborious task, computer-based annotation tools need to provide an interface that makes the annotation process as convenient and efficient as possible.

Linguistic annotation

In this work, the problem context will be narrowed down to the special case of *linguistic annotation*, which is a common task in corpus linguistics and other disciplines that rely on the quantitative analysis of text. While trying to algorithmically implement knowledge about language, the usability of tools for manual annotation is typically neglected by software developers in this field. As a result, we find a plethora of existing tools that aim to support and facilitate manual annotation, but often struggle to do so, because of poor interface design and an unawareness or disregard of basic usability principles. The design of user-friendly linguistic annotation tools is even more challenging, as the user group can be characterized by a low degree of computer literacy.

[1] This assumption is backed up by recent crowdsourcing marketplaces, like e.g. *Amazon mechanical turk* (available at `https://www.mturk.com`), where numerous manual annotation tasks are offered to human annotators. Note: *All* web pages referenced in this dissertation were last accessed on June 06, 2014.

1.2. Research agenda

The recurring theme of this dissertation is the improvement of user interfaces for manual annotation tasks in a linguistic context. This goal will be achieved by evaluating a number of available annotation tools, in order to derive hints for good and bad designs in this specific domain of application. These hints will be documented as usability patterns. Such patterns can help future tool developers to prevent common mistakes and pitfalls in tool design. They also guide actual tool users to decide which tool to choose with regard to usability requirements.

1.2.1. Objectives

To achieve this higher-level goal, a number of subordinate objectives will have to be met. The following research objectives will be addressed in this dissertation:

- **Annotation theory and linguistic annotation**: Capture the state of the art of annotation theory and derive implications for the design of user-friendly, linguistic annotation tools (cf. chapter 2); describe typical standards and methods for linguistic annotation and further illustrate the motivation of the overall research goal of this work (cf. chapter 3).
- **Evaluation design**: Describe the domain of linguistic annotation in more detail (typical users, requirements, tasks, tools) and find an appropriate method for improving the user interface of annotation tools (cf. chapters 3 and 4); design and conduct an evaluation study that reveals positive and negative aspects of annotation tool interfaces (cf. chapter 4).
- **Pattern identification**: Discuss the appropriateness of usability patterns as a means for documenting design knowledge about annotation tool interfaces; integrate the results obtained from the evaluation study with a transparent pattern identification process (cf. chapter 5).
- **Usability patterns**: Create a collection of generic usability patterns for the design of user-friendly, linguistic annotation tools (cf. chapter 6).

1.2.2. Scope and limitations

Although chapter 2 approaches annotation on a rather generic level, the focus of this work is quite specific. Figure 1.1 shows the context, scope and limitations of this dissertation: While it is possible to linguistically annotate images, videos or audio files, this research focuses on the annotation of *text documents*. Text documents and their annotation have a long history, and despite the rise of new multimedia formats in the web context, we can also observe an increased availability of text documents, as the web still is an interconnected hypertext at

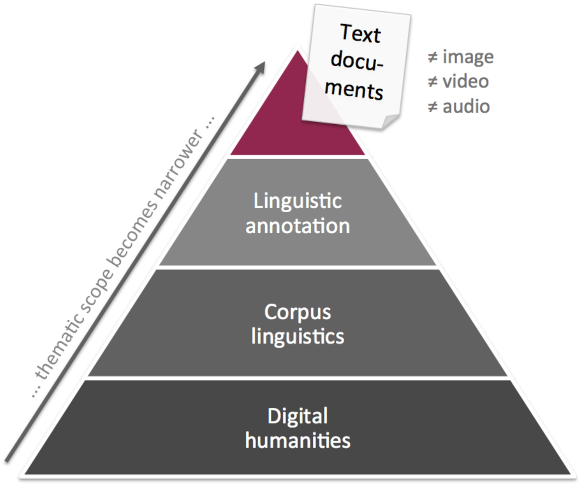


Figure 1.1.: Wider context, specific scope ("linguistic annotation of text documents") and basic limitations (no image, video or audio annotation") of the dissertation.

its core. Social media services (e.g. *Facebook* and *Twitter*) in particular churn out textual data in vast amounts, sometimes also referred to as *big data*.

Linguistic annotation is a typical task in the field of *corpus linguistics*, a branch of linguistics that makes use of empirical data and quantitative methods. Corpus linguistics in turn can be seen as a prime example of the *digital humanities*, which is a rather broad term that describes the use of digital tools and resources in the humanities (cf. Schreibman et al., 2004). The implications of this research for other tools and applications from the field of digital humanities will be addressed in the outlook section at the end of this dissertation.

1.3. Related publications

The following articles relate to work described in this dissertation and have been published in the course of the last years. It will be made clear throughout the dissertation whenever contents from one of these articles are reproduced.

- Burghardt, M. (2012). Usability Recommendations for Annotation Tools. In *Proceedings of the ACL 2012, 6th Linguistic Annotation Workshop, LAW '12* (pp. 104–112). Stroudsburg, PA, USA: Association for Computational Linguistics.
- Burghardt, M. (2012). Annotationsergonomie: Design-Empfehlungen für linguistische Annotationswerkzeuge. *Information, Wissenschaft & Praxis, 63(5)*, 300–304.

  Both these articles describe the design and results of a pilot study for the evaluation of annotation tools. The pilot study is important preliminary work for the *exhaustive series of heuristic walkthrough evaluations that is described in chapter 4.5.*

- Burghardt, M., & Wolff, C. (2009). Werkzeuge zur Annotation diachroner Korpora. In W. Höppner (Ed.), *Proceedings of the GSCL-Symposium "Sprachtechnologie und eHumanities"* (pp. 21–31). Duisburg: Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen.

  The work described in this article provides a first systematic study of annotation tool interfaces, indicating that most existing annotation tools are suffering from severe usability problems. The contents of this work are referenced in chapter 4.4 (relevant aspects: tool requirements, annotation tasks, annotation tool classes).

1.4. Scholarly context

This dissertation is an interdisciplinary[2] research project that solves a practical problem from the field of *linguistics* with tools and methods from the domain of *usability engineering*. Usability research is associated with many different scholarly disciplines, most notably *psychology* (*cognitive studies*; *human factors*) and *computer science* (*human-computer interaction*; *interface design*). By analyzing digital annotation tools and practices, the dissertation also shows many characteristics of the *digital humanities* field.

At its core, however, this work is a typical *information science* (IS) project, as IS has many connecting factors to linguistics and digital humanities as well as to usability evaluation and user interface design. A concise summary of the relationship between IS and linguistics is provided by Montgomery (1972, p. 195):

> Information science is concerned with all aspects of the communication of information, language is the primary medium for the communication of information, and linguistics is the study of language as a system for communication information.

While Engerer (2012) presents a comprehensive, historical overview of the relationship between IS and linguistics, there are also more recent examples of linguistically motivated IS research, e.g. a study on the relation of interlingual aspects and the information quality of articles in Wikipedia (Hammwöhner, 2007). IS's interdisciplinary relationships are, however, not restricted to the field of linguistics, but can rather be expanded to a wider range of humanities[3], such as *literary studies* (IS aspect: *digital libraries*), *art history* (IS aspect: *image retrieval*) and *musicology* (IS aspect: *music information retrieval*). The connecting factors between IS and *digital humanities* are also reflected by recent activities in the community, such as the "5. Potsdamer I-Science-Tag" (organized by the the IS group

[2] Also cf. the interdisciplinary perspective on annotation in chapter 2.2.

[3] Also cf. Womser-Hacker (2010, p. 335), who notes that German IS has its roots in the humanities and social sciences as well as in mathematics and computer science.

at the Fachhochschule Potsdam, in early 2014), which had the motto "Digital Humanities meets Information Science"[4]. Finally, *usability evaluations* of information systems can be seen as traditional IS research topics (cf. e.g. Ferreira & Pithan, 2005).

This section has illustrated the wider scholarly context of this work as well as the many connecting factors to the field of information science.

1.5. Outline of the dissertation

Figure 1.2 shows the overall structure of this dissertation and the main functions of each chapter at a glance:

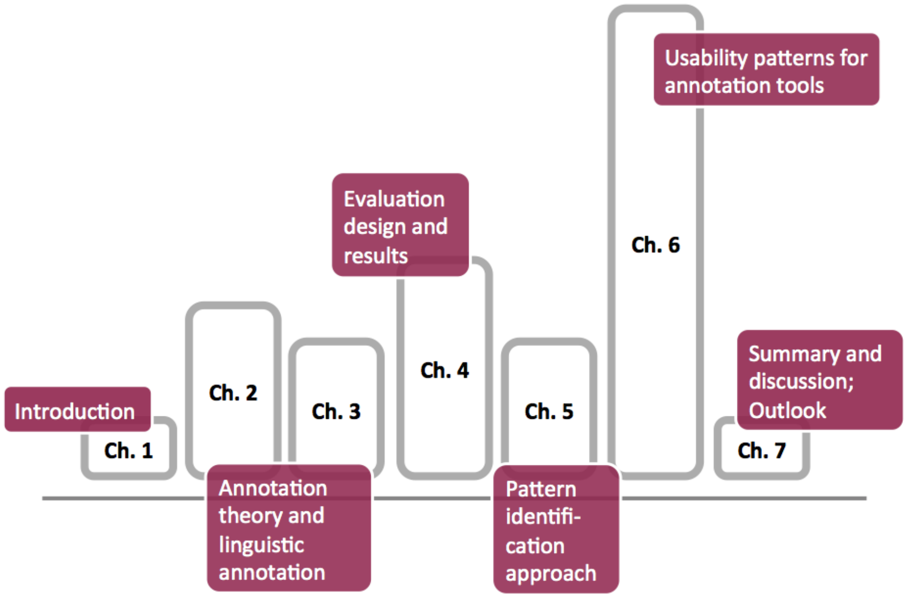

Usability patterns for annotation tools
Evaluation design and results
Ch. 6
Ch. 2
Ch. 3
Ch. 4
Ch. 5
Summary and discussion; Outlook
Introduction
Ch. 1
Ch. 7
Annotation theory and linguistic annotation
Pattern identifi-cation approach

Figure 1.2.: Main function of the chapters in this dissertation at a glance. The size of the chapters roughly indicates its extent (i.e. number of pages dedicated to the particular chapters).

Ch. 1: Introduction The introductory chapter gives an overview of the context of this work and also presents the research objectives of the dissertation.

Ch. 2: Annotation theory This chapter describes the state of the art of annotation theory, including basic definitions and terminology as well as functions and characteristics of annotations. The engagement with existing annotation theory and the adaption to a linguistic context may be seen as impor-

[4] Cf. DHd (*Digital Humanities im deutschsprachigen Raum*) blog, `http://dhd-blog.org/?p=3050`

tant preliminary work for the subsequent chapters. The chapter concludes with a case study on linguistic annotation in a pen and paper context.

Ch. 3: Linguistic annotation The third chapter introduces common annotation practices and standards from the area of linguistic annotation. These practices make clear that there is a necessity for manual annotation despite the existence of automatic and semi-automatic annotation tools.

Ch. 4: Usability and the case for annotation tools This chapter starts with a description of the fundamentals of usability engineering, introducing basic terminology and methods. It also characterizes the domain of linguistic annotation tools in more detail by presenting user classes, tool requirements, typical tasks and a taxonomy of annotation tools. The main part of this chapter is dedicated to the design of an evaluation study for the identification of the strengths and weaknesses of the user interface of existing annotation tools. The quantitative results of this study are summed up at the end of the chapter, whereas the qualitative results are discussed in more detail in chapter 6.

Ch. 5: Patterns and pattern identification This chapter introduces *patterns* as a possible means to document the results from the usability evaluation study that was conducted in the previous chapter. At the beginning of this chapter, the general idea of the pattern concept is introduced, followed by a more specific account of the characteristics of patterns in the context of usability engineering and HCI. It becomes obvious that the identification of patterns is a rather vague and unstructured process. This chapter presents a systematic approach that integrates an expert-based usability inspection method and the process of pattern identification. The approach is illustrated by the explanation of the generation of an exemplary pattern.

Ch. 6: Usability patterns for annotation tools This chapter describes the main results of this work: a collection of twenty-six usability patterns for linguistic annotation tools, that are based on the results of a series of heuristic walkthrough evaluations. The patterns are organized according to six categories (*Installation*, *General UI*, *Primary data*, *Annotation scheme*, *Annotation process*, and *Annotation visualization*). Evaluation results that could not be refined into a usability pattern are also discussed individually in this chapter.

Ch. 7: Summary and outlook The last chapter concludes with a summary of the main contributions of this work. Moreover, it presents a discussion of the identified design patterns and critically reflects on the overall approach, i.e. the integration of heuristic walkthrough and usability pattern structure. Finally, an outlook on future work beyond the scope of this dissertation is presented.

2. Annotation theory

2.1. Introduction

This chapter gives an overview of the large body of annotation theory and research, which is vital for understanding the domain of linguistic annotation and its cognitive and functional implications for the human annotator. It will become clear that annotation is a generic concept that reaches from semantically annotated websites to handwritten comments in university course books. With the rise of digital annotation and the steady transition of annotations from paper to screen, numerous researchers have engaged with this topic (cf. Marshall, 1997, 1998; Schilit et al., 1998; Ovsiannikov et al., 1999; Bottoni et al., 2003; Fogli et al., 2004; Agosti et al., 2005; Agosti & Ferro, 2007; Agosti, Bonfiglio-Dosio, & Ferro, 2007; Marshall, 2010, and many more).

Research questions

In this chapter, the most prominent approaches to analyze and model annotations are introduced. A review of relevant literature will help to answer the following questions, as raised by Agosti, Bonfiglio-Dosio, & Ferro (2007, p. 1):

Q1 What is an annotation?

Q2 What are the features of annotations?

Q3 What are the ways of using annotations?

Although most of the theoretic work on annotations describes *human handwritten annotation* (cf. Ovsiannikov et al., 1999), there are many connecting factors to linguistic annotation, which is much more formalized. The implications of human handwritten annotation theory for linguistic annotation tools will be pointed out and discussed whenever appropriate. Understanding the theoretic background of annotations in general is vital to be able to suggest solutions for the user-friendly design of linguistic annotation tools. It is also helpful to have existing terminology at hand when it comes to discussing specific characteristics of linguistic annotation.

Chapter structure

The chapter is structured as follows: Section 2.2 describes annotations from the perspective of various scholarly disciplines; section 2.3 gives an overview of the history and etymology of the term *annotation* and illustrates how annotation has changed in the course of time. Section 2.4 provides a detailed description and definition of annotation and related concepts. The basic constituents of an annotation have been modeled by various researchers, with slightly different terminology and varying degrees of formalization; section 2.5 presents an overview of these different annotation models. While section 2.6 describes the characteristics of annotations, section 2.7 takes into account the different functions of annotations. Section 2.8 provides the results of a an evaluation study on handwritten annotations in a linguistic context; section 2.9 presents a brief summary of the whole chapter and leads over to the next chapter that addresses "linguistic annotation" in more detail.

2.2. An interdisciplinary perspective on annotation

Ambiguity of the term

Up to this point, annotation has been used as an umbrella term that needs to be further defined and disambiguated. The ambiguity of the term is also underlined by the final report on the "Summit on Digital Tools for the Humanities" (Frischer et al., 2005, p. 7), where several "sub-components of annotation . . . as an interpretation-building process" have been identified:

- Identify the environment (discipline, media)
- Encounter a resource (search, retrieval)
- Explore a resource
- Vary the scope / context of attention
- Tokenize, segment the resource (automatically or manually)
- and many more

Although annotations vary in form and function (cf. Agosti, Bonfiglio-Dosio, & Ferro, 2007), there is one basic motivation behind any annotation task:

Basic motivation Annotations are always used to make some implicitly available information explicit, either for oneself, for other humans or for computer programs.

Information retrieval

From an *information retrieval* (IR) perspective, annotation is very similar to the process of document indexing. Such indices are used to create a meaningful representation of a text which in turn can be accessed efficiently by a retrieval algorithm. Accordingly, IR systems (e.g. a web search engine) can only be used to search for specific document content if that content has a systematic address, or as Gugerli (2009, p. 15) puts it: "That which is searched for has to be labeled in order to be available and accessible" (translated from German). At the same time, the labels have to be correct and meaningful, as "incorrect annotations inevitably lead to unsuccessful applications" (Wilcock, 2009, p. 1).

Digital libraries

In the context of *digital libraries* (DL), annotations are understood to not only make intrinsically hidden information explicit, but also to add new information to existing contents, thus enriching a given text with new thoughts and references to other relevant texts (Agosti, Coppotelli, et al., 2007, p. 2). Along these lines, Marshall (1998) conducted a study to analyze communicative aspects of annotations, and to evaluate the added value of annotations in university text books, which are passed from student to student via a store that is selling used course books, describing the *ecology* of such annotated texts.

Digital humanities

Scholars from the digital humanities have discovered the need for annotation in the context of computer-supported work scenarios more than 30 years ago: Stone (1982, p. 300) argues that although the computer does support scholarly work by means of large-scale data storage and manipulation, humanists need to know about existing data and how it is represented and described, i.e. scholars need to normalize and annotate their data before it can be processed by computer tools. Schreibman et al. (2004) introduce the concept of digital *re-representation* of physical artifacts, which describes the need for annotation very well: While the mere *representation* of physical artifacts only creates a *surrogate* (cf. Unsworth, 2004) that is trying to imitate the original artifact, it still is only an attempt to preserve the original as a digital representation. In order to be able to "reveal properties and traits not evident when the artifact was in its native form" (Schreibman et al., 2004, p. xxiv), it has to be re-represented by means of adding annotations.

In the same context Unsworth (2000) defines seven *scholarly primitives*, which cover the basic activities and scholarly practices of humanists, including *discovering*, *annotating*, *comparing*, *referring*, *sampling*, *illustrating*, and *representing*. Other models that try to capture scholarly workflows include annotation as a basic task, too (for an overview cf. Bélanger, 2010a). McLoughlin (2008) stresses the importance of annotations beyond the scope of digital humanities, arguing that it is indispensable for any researcher to explicitly encode those aspects of data that will be investigated in more detail during the course of later research: "Whatever aspects of the text are not encoded will not be available to the user" (McLoughlin, 2008, p. 9). Of course this also holds true for researchers from areas who are not dealing with text, but who are annotating more complex objects such as video files, images or even genomes.

Corpus linguistics

This dissertation is focused on the linguistic annotation of text documents, which can be seen as a popular task in the field of *corpus linguistics* (for a more detailed discussion of corpus linguistics as a discipline cf. chapter 3.2). Linguistic annotation is concerned with the explicit labeling of implicit information about words (e.g. parts of speech (POS)), sentences (e.g. syntactic information) and the document as a whole (e.g. genre), in order to make it accessible for other readers – especially for readers who might have difficulties in making up such implicit information on their own, most notably computer programs (cf. chapter 3 for a more detailed description of linguistic annotation).

2.3. A short history of annotation

Etymology

In order to shed some light on the rather generic concept of annotation, Agosti, Bonfiglio-Dosio, & Ferro (2007, p. 2) investigated the word *annotation* and related terms by looking at their historical usage. Etymologically speaking, the term can be traced back to its Latin origin *annotare*, which means "to annotate" or "to observe in writing" (Agosti, Bonfiglio-Dosio, & Ferro, 2007, p. 3). More precisely, they describe that *annotare* comes from Latin *ad + nota*, where *nota*, means "to note or to mark", and the intensifying suffix *ad* in compound words means "to approach / tend /add", i.e. "*ad + nota*" can be translated as "add a note".

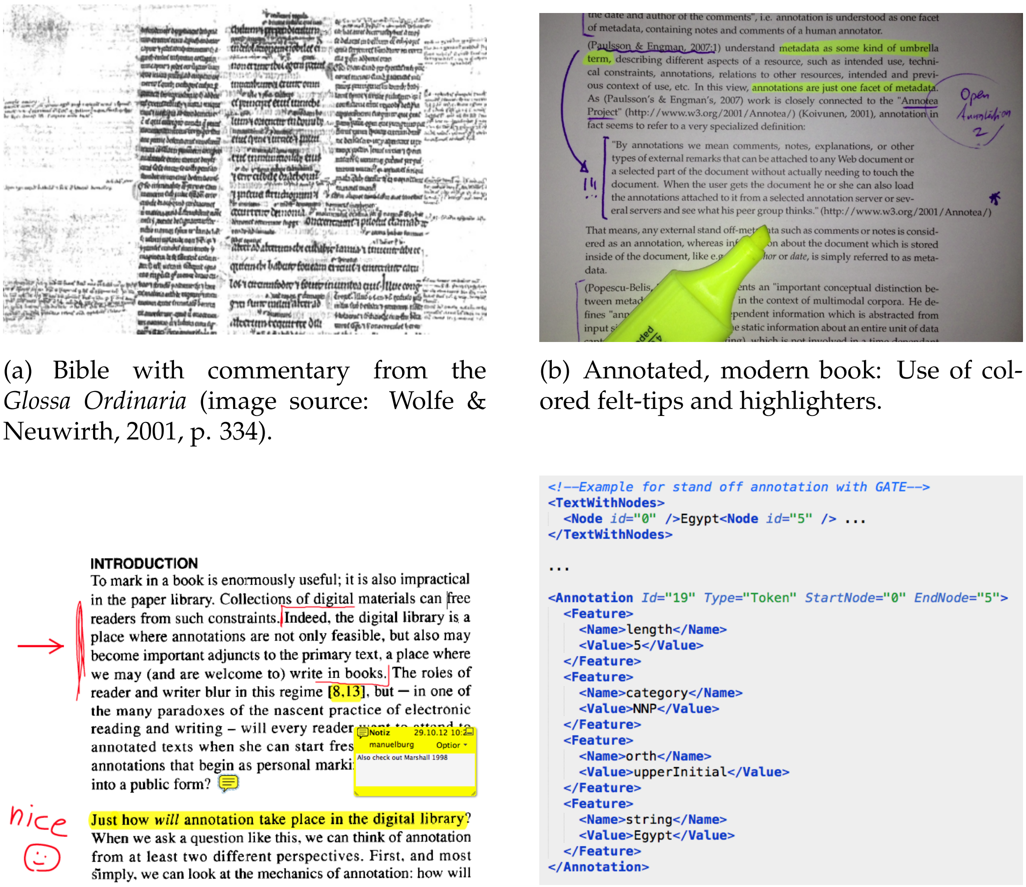
Synonyms and related terms

Apparently, the basic function of an annotation, which is to add some additional information to an object like e.g. a text or an image, is expressed through the very term *annotation*. Agosti, Bonfiglio-Dosio, & Ferro (2007, p. 4) also provide historical viewpoints on the concept of annotation by systematically looking up synonyms and related terms in a dictionary of the English language. Among the terms they identify are *comment*, *elucidation*, *explanation*, *footnote*, *gloss*, *jotting*, *note*, *postil*, and many more. While each formulation is to be found in a specific context, and each comprises some specific information about the actual annotation, all of them share the basic function mentioned above: to add some kind of additional information to an object.

Evolution of annotation

Figure 2.1 illustrates that textual annotations have evolved, as has the medium for text. With the steady change from paper to screen, the functions of annotations have changed, too. Picture (a) shows an annotated bible, with rich annotations all around the original text (in larger print). Agosti, Bonfiglio-Dosio, & Ferro (2007, p. 4) emphasize that the act of annotating entails an "autonomous intellectual work". The value of historic annotations as some kind of self-contained product of intellectual effort is also stressed by Wolfe & Neuwirth (2001, p. 333), as they describe annotations as "central to knowledge sharing in medieval literary cultures", where annotations were used to discuss and communicate with other annotators. As text reproduction in medieval times was accomplished by laborious, manual copying and transcribing, oftentimes annotations were copied along the way of transcription, thus becoming part of the primary text. The communicative and cooperative functions of annotations will be examined in more detail later on. It is, however, noteworthy that the main function of annotations as a means of communication between different *human* readers of a text has expanded to include computers in the communication as well, thus demanding more formal and abstract forms of annotation which in turn are not easy to understand by humans. It is due to this change of annotations that we find a gap between human- and computer-readable annotations.

Wolfe & Neuwirth (2001, p. 333ff.) also note that in medieval times, i.e. before the age of printed books, annotations were produced more collaboratively, as many people had to share the same book (and the same annotations). As a result



the date and author of the comments", i.e. annotation is understood as one facet of metadata, containing notes and comments of a human annotator.

(Paulsson & Engman, 2007:1) understand metadata as some kind of umbrella term, describing different aspects of a resource, such as intended use, technical constraints, annotations, relations to other resources, intended and previous context of use, etc. In this view, annotations are just one facet of metadata. As (Paulsson's & Engman's, 2007) work is closely connected to the "Annotea Project" (http://www.w3.org/2001/Annotea/) (Koivunen, 2001), annotation in fact seems to refer to a very specialized definition:

> "By annotations we mean comments, notes, explanations, or other types of external remarks that can be attached to any Web document or a selected part of the document without actually needing to touch the document. When the user gets the document he or she can also load the annotations attached to it from a selected annotation server or several servers and see what his peer group thinks." (http://www.w3.org/2001/Annotea/)

That means, any external stand off-met ata such as comments or notes is considered as an annotation, whereas in on about the document which is stored inside of the document, like e.g. hor or *date*, is simply referred to as metadata.

(Popescu-Belis, ents an "important conceptual distinction between metad in the context of multimodal corpora. He defines "ann pendent information which is abstracted from input s e static information about an entire unit of data

Open Annotation 2

INTRODUCTION

To mark in a book is enormously useful; it is also impractical in the paper library. Collections of digital materials can free readers from such constraints. Indeed, the digital library is a place where annotations are not only feasible, but also may become important adjuncts to the primary text, a place where we may (and are welcome to) write in books. The roles of reader and writer blur in this regime [8,13], but — in one of the many paradoxes of the nascent practice of electronic reading and writing – will every reader annotated texts when she can start fres annotations that begin as personal marki into a public form?

Also check out Marshall 1998

Just how *will* annotation take place in the digital library? When we ask a question like this, we can think of annotation from at least two different perspectives. First, and most simply, we can look at the mechanics of annotation: how will

nice

```
<!--Example for stand off annotation with GATE-->
<TextWithNodes>
  <Node id="0" />Egypt<Node id="5" /> ...
</TextWithNodes>

...

<Annotation Id="19" Type="Token" StartNode="0" EndNode="5">
  <Feature>
    <Name>length</Name>
    <Value>5</Value>
  </Feature>
  <Feature>
    <Name>category</Name>
    <Value>NNP</Value>
  </Feature>
  <Feature>
    <Name>orth</Name>
    <Value>upperInitial</Value>
  </Feature>
  <Feature>
    <Name>string</Name>
    <Value>Egypt</Value>
  </Feature>
</Annotation>
```

(a) Bible with commentary from the *Glossa Ordinaria* (image source: Wolfe & Neuwirth, 2001, p. 334).

(b) Annotated, modern book: Use of colored felt-tips and highlighters.

(c) Annotated PDF.

(d) XML markup & rendering.

Figure 2.1.: Examples for different kinds of annotation throughout history.

of the rise of the letterpress, people tend to have their private copy of a book, and are therefore more inclined to make personal annotations (cf. Picture (b); also cf. Marshall & Brush, 2004).

Digital books (cf. Picture (c)) technically provide both annotation scopes that have been described so far: the annotator can keep his copy and his annotations private, or he can share his annotations (read + write) with others[5]. Beyond that, digital texts in the form of PDFs or some other e-book formats play an important role in bridging the gap between human handwritten annotations and more formalized, machine-readable annotations. A PDF for instance may be annotated with some freehand content, i.e. it is very similar to handwritten annotation on paper, but it cannot be interpreted by the computer. If a note is applied to the PDF via keyboard, it is in a machine readable format but does not feel quite as natural as freehand annotation[6].

[5] Many recent ebook readers, like for instance *Amazon's Kindle*, offer to peek at the shared annotations of other readers of the same digital book. The actual annotators are, however, anonymous.

[6] It must be noted that most freehand annotation modes do not feel too natural at all. Only few

Picture (d) shows an example of digital annotation, which is highly formalized by using XML markup. These annotations are optimized for machine processing, but at the same time are not ideal for human processing. Scholars from the humanities in particular have a hard time reading and writing XML and related formats, as markup languages typically are not part of their genuine, domain-specific skill set. Vanhoutte (2011, sec. 2.2, para. 3) mentioned the issue of "angle brackets fear" in his keynote at the *2011 Annual Conference and Members' Meeting of the TEI Consortium* , kicking off a lively debate (which was picked up several times throughout the conference) between those who think it is just a natural fear, as XML is none of a humanities scholar's business, and those who think it is just a psychological condition that can be easily overcome by any humanist who is willing to learn formal markup languages.

Implications for linguistic annotation: The main challenge for linguistic annotation tools is to bridge the gap between human handwritten annotations and formalized, machine-readable markup. Flynn (2006, 2009) has noted the same gap in the context of authoring tools for structured documents, e.g. for XML or TeX editors.

2.4. Describing *annotation* and related concepts

Annotation: activity vs. result

Before trying to provide a working definition for the term annotation, a basic distinction has to be made when talking of annotations: the first meaning of annotation denotes an *activity*, i.e. "the act of annotating", the second meaning denotes the *result* of such an activity, the actual annotation, i.e. "a note added as an explanation" (Agosti, Bonfiglio-Dosio, & Ferro, 2007, p. 2). In the course of this dissertation both meanings will be used frequently, as an enhanced usability of annotation tools is concerned with both, the act of annotation, i.e. some kind of user activity, as well as the actual result, the annotation itself. If the distinction of process vs. result is not possible by mere context, annotation as an activity will be paraphrased as *annotation task*, *annotation process*, etc.

Formal vs. informal annotation

The most common way to distinguish annotations is by their degree of formality and structure (Marshall, 1998, p. 41). XML markup can be seen as a good example for more formal annotations, whereas freehand scribbles and comments on printed or electronic books are less structured and less formal. Traditionally, annotations have been informal, but with the rise of electronic texts and the main *recipient* of annotations increasingly becoming machines rather than humans, a more formal approach to annotation had to be established. Agosti et al. (2005, p. 4) illustrate the difference between formal and informal annotations in the context of digital libraries (DLs): While the formal tagging of electronic documents

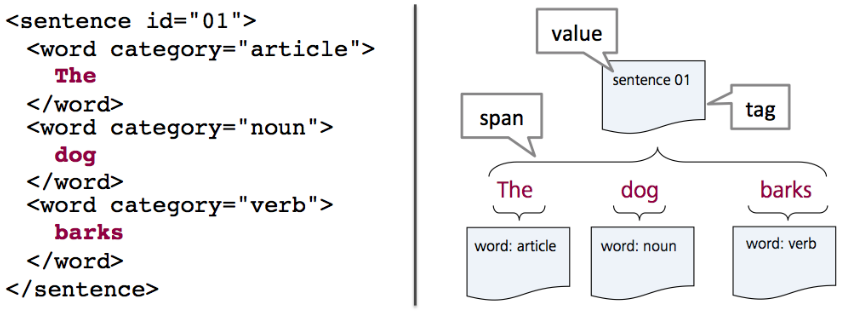
systems use pens and highly sensitive touch-screens to provide a nearly natural annotation experience.

is always based on some predefined annotation scheme, informal annotations such as personal user comments, which are by nature less structured, are often formulated *ad hoc* and are intended mainly for social navigation through the text.

TEI

As corpus linguistics has been so successful mostly due to the availability of large amounts of electronic data and machines to process that data, formal markup languages are closely connected to state of the art linguistic annotation. While the different existing standards and languages in this area will be addressed in chapter 3.3, for now some basic terminology will be discussed as it is used by the *Text Encoding Initiative* (TEI[7]), which is one of the most renowned XML based markup languages[8] for the academic annotation of text documents. The basic purpose of the TEI is to provide a suitable way of representing the features of textual resources in order to enable and facilitate their processing by computer programs (TEI, 2014a, para. 2). More concretely, TEI suggests a set of textual markers, so called *tags*, which allow computers to distinguish between structural/semantic markup and the original text. Tags function like machine-readable labels that can be placed on a piece of text. Such labels have two main functions: (1) To provide additional information about the labeled text, e.g. how it should be presented, and (2) to indicate which span of text is labeled, i.e. where one label stops and some other label or even entirely unlabeled pieces of text start (cf. Figure 2.2).

Tags



```
<sentence id="01">
  <word category="article">
    The
  </word>
  <word category="noun">
    dog
  </word>
  <word category="verb">
    barks
  </word>
</sentence>
```

Figure 2.2.: XML tags function as labels, adding information to a delimited piece of text.

Markup, encoding, tagging

A specific collection of tags is called a *markup vocabulary*, which is defined by a more generic *markup language*. According to the TEI guidelines, the process of adding such tags to a textual resource is called *markup*, *encoding* or just *tagging*. These terms can be seen as synonyms for annotation, where each of them describes rather formal types of annotation. Bearing in mind that the term annotation is ambiguous in describing informal as well as formal additions to a text, it may at first seem strange to speak of *linguistic annotation* rather than linguistic

Linguistic annotation

[7] `http://www.tei-c.org`

[8] XML can be used to define markup languages for specific purposes (e.g. XHTML for the markup of websites) by defining the features and characteristics of a document type in a *Document Type Definition* (DTD).

markup or encoding. Nagao (2003, p. 61) observes a trend to use the term annotation in the area of language engineering and corpus linguistics[9]. A Google lookup of the frequencies of phrases like *linguistic annotation* or *linguistic markup* seems to confirm the predominance of *annotation* when combined with *linguistic* (cf. Table 2.1).

Term	**Total hits on Google**	**Hits for phrase** ("*linguistic* + term")	**Percentage**
Annotation	51 200 000	47 900	0.094%
Encoding	113 000 000	20 000	0.018%
Markup / mark-up	45 650 000	4 060	0.009%
Tagging	66 300 000	4 060	0.006%
Labeling / labelling	75 100 000	2 715	0.004%

Table 2.1.: Frequencies for single terms and phrases ("linguistic" + term) as well as percentage of phrases in relation to the total hits (source: Google lookup, August 29, 2012).

Bradley & Vetch (2007, p. 1) explain their understanding of annotation as some kind of umbrella term, differentiating a "specialized meaning", such as e.g. linguistic annotation, and a "more mundane" meaning of annotation, which is very closely related to the area of unstructured, freehand annotations, for instance on the margins of a book. In the course of this dissertation, the term *annotation* will be used synonymously for markup, encoding or tagging[10], and the term *annotation scheme* for markup vocabulary. One of the most common ways to describe such an annotation scheme is by creating a *Document Type Definition* (DTD). A DTD not only contains all possible or allowed tags of an annotation scheme, but also basic rules on how the different tags can be structured and combined. A more sophisticated way to describe an annotation scheme would be by means of XML Schema.

Basic types of annotation

Annotations can be categorized as (1) *structural* annotations, which are needed to describe the structure of a text, and (2) *descriptive* (or also: positional) annotation, which describe the actual text in a semantic dimension (Paulsson & Engman, 2007, p. 2). The second type can have different levels of granularity, i.e. a descriptive annotation can relate to one word, a sentence or the whole document. When it comes to descriptive annotations, another term frequently occurs: *Metadata* is defined and used very heterogeneously in respective literature. Some authors define metadata as a subcategory of annotation (cf. Paulsson & Engman, 2007; NISO Press, 2004), and vice versa (cf. Agosti & Ferro, 2007). Popescu-Belis (2010, p. 189) notes that depending on the respective field of study, some disciplines seem to be more fond of the term annotation (e.g. speech and language

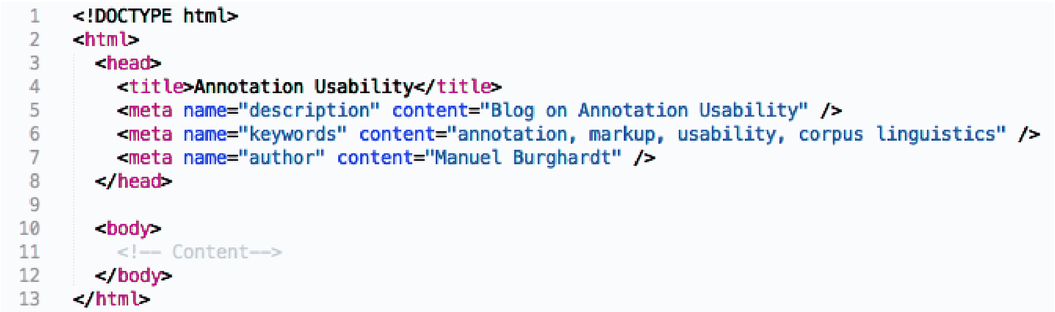
Annotation vs. metadata

[9] For more examples of the use of *annotation* in a linguistic context cf. Leech (1997), Ide & Romary (2004), Wilcock (2009), and many others. The term is also used throughout international standards like the *Linguistic Annotation Framework* (ISO 24612, 2012) .

[10] The term *tagging* will be picked up as a specific form of linguistic annotation in chapter 3.6, where tagging refers to the annotation of word classes (*part of speech tagging*).

studies), while others prefer metadata (e.g. video and image processing). Ruvane (2007, p. 1), however, observes that similar to the use of markup and encoding for more formal types of annotation, there seems to be an inclination to use the term metadata for formal markup of digital documents, and annotation for more informal, unstructured notes or comments scribbled on paper. This interpretation sounds plausible, as traditionally the term metadata has been used in libraries, in order to describe books with the help of a set of formalized, predefined metadata categories such as *title*, *author*, *year*, *genre* etc. With the rise of digital resources, metadata plays an important role in the digital world and especially in the semantic web, spawning many different standards and frameworks like e.g. the *Dublin Core Metadata Initiative* (DCMI), the *Resource Description Framework* (RDF), and the *Multimedia Content Description Interface* (MPEG-7). Figure 2.3 shows a typical example for the use of metadata in the context of websites. HTML (*Hypertext Markup Language*) provides a set of meta-tags, including e.g. a *description* of the website, *keywords*, which are very similar to the descriptors used in libraries, and an *author* of the page content.

Metadata frameworks



```
<!DOCTYPE html>
<html>
  <head>
    <title>Annotation Usability</title>
    <meta name="description" content="Blog on Annotation Usability" />
    <meta name="keywords" content="annotation, markup, usability, corpus linguistics" />
    <meta name="author" content="Manuel Burghardt" />
  </head>

  <body>
    <!-- Content-->
  </body>
</html>
```

Figure 2.3.: HTML metadata for websites.

Popescu-Belis (2010) discusses the conceptual distinction between metadata and annotation in the context of multimodal corpora, by introducing the dimension of *time* as a parameter of differentiation. He defines "annotations as the time-dependent information which is abstracted from input signals" and "metadata as the static information about an entire unit of data capture (e.g. a session or a meeting), which is not involved in a time dependent relation to its content" (Popescu-Belis, 2010, p. 187-188). Although the dimension of time is of little relevance for the annotation of static texts, the timeline is an important metaphor in multimodal annotation of video and audio data, as it indicates the sequential processing of a multimedia file. Time-dependency could be translated into the field of static text annotation as the selection of single spans of text while time-independent metadata applies to the whole document.

Agosti & Ferro (2007, p. 4ff.) identify two main views on annotation[11] (also cf. Agosti, Coppotelli, et al. (2007, p. 2) and Marshall (1998, p. 41)):

Annotations: *metadata* vs. *content*

[11] Agosti, Bonfiglio-Dosio, & Ferro (2007, p. 7-8) also suggest further viewpoints on anno-

- *Annotation as content* (cf. Marshall's concept of *informal annotations*)
- *Annotation as metadata* (cf. Marshall's concept of *formal annotations*)

Annotation as *content* is to be understood as new, additional content that is added to some previously existing content. It is less structured and formalized and thus primarily intended for human recipients. Annotations as content can be further differentiated as "annotation as content enrichment" and "annotation as stand-alone document", which indicates that annotations can become actual documents themselves (Agosti, Bonfiglio-Dosio, & Ferro, 2007, p. 7). However, annotation as *metadata* is characterized by extending some existing content in order to clarify its meaning. Typically this kind of annotation is more formal and to some degree restricted and limited, as it has to adhere to a predefined annotation scheme. At the same time, metadata annotations can be processed more easily by computer programs. An overview of the differences between annotation as *content* and annotation as *metadata* is given in Table 2.2.

Parameter	**Annotations as content**	**Annotations as metadata**
Description	New content added to existing content	Data added to existing content
Recipients	Humans (make document readable for humans)	Computers (make document machine-readable)
Form	Informal / implicit annotation, not easily processable by a machine	More formal / explicit annotation, has to adhere to a predefined scheme
Function	Enhance existing content by additional elucidations and explanations	Clarify properties and semantics of the annotated content

Table 2.2.: Overview of annotations as *content* vs. annotations as *metadata* (cf. Agosti & Ferro, 2007; Agosti, Coppotelli, et al., 2007; Agosti, Bonfiglio-Dosio, & Ferro, 2007).

A typical example for metadata is the annotation of websites with metadata, which is also called semantic annotation (cf. the respective metadata frameworks mentioned above). Agosti & Ferro (2007, p. 5) describe another example, which is vital for the view on annotations in this work:

> Similar uses of annotations can be found in the natural language processing field; for example, part of speech tagging consists of annotating each word in a sentence with a tag that describes its appropriate part of speech so as to decide whether a word is a noun, a verb, an adjective.

Implications for linguistic annotation: Although Agosti & Ferro (2007, p. 5) categorize linguistic annotation as metadata, the predominant term for adding

tations, such as "annotations as hyperlinks", stressing the possibility to create relationships between contents, "annotations as a kind of context", i.e. annotations introducing a new layer of explanation of contents, and "annotations as dialog acts", describing the collaborative and communicative nature of annotations.

linguistic information to a text document is *annotation* (cf. the previous discussion on the usage and interpretation of *annotation* in different disciplines). For the rest of this dissertation, *annotation* will refer to adding linguistic information to a text in the form of tags.

2.5. The elements of annotation

Until now, the concept of annotation has been discussed on a rather generic level, assuming that an annotation is anything that is added to the original text. It is, however, necessary to identify and describe the single constituents of an annotation to be able to systematically improve the process of linguistic annotation. These basic constituents have been discussed from many different perspectives (Agosti et al., 2004; Agosti & Ferro, 2007; Bargeron et al., 2001; Bottoni et al., 2003; Fogli et al., 2004; Marshall, 2010; Ovsiannikov et al., 1999), some more formal than others (cf. Figure 2.4).

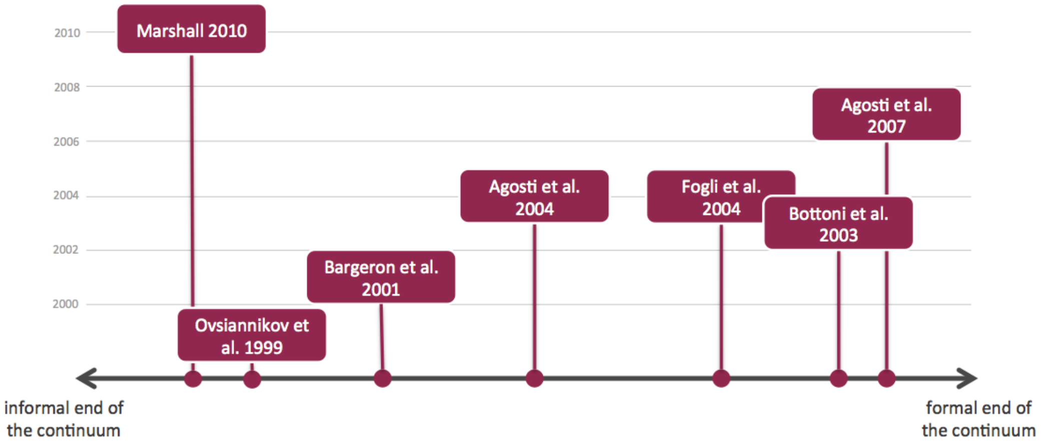
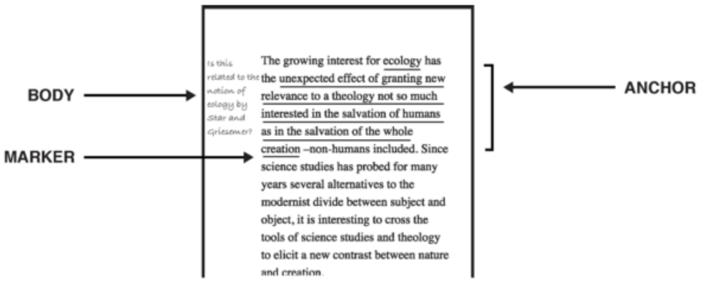


Figure 2.4.: The elements of annotation are defined and discussed with different degrees of formality throughout literature (visualization as a continuum).

The goal of this section is to introduce some of the most prevalent definitions for the elements of annotations and to discuss and adapt them for the context of linguistic annotation.

2.5.1. Anatomy of an annotation

Catherine C. Marshall was one of the first to systematically describe the practices of handwritten annotation. Her basic idea was to analyze students' annotations in printed university textbooks, to examine implications for digital material and to derive clues on how to design annotation systems in the digital library context (cf. Marshall, 1997, 1998). Since Marshall's studies toward a theory of annotation, several other scholars have tried to identify and formalize the elements of annotation (cf. Figure 2.4). In her most recent book Marshall (2010, p. 42) stresses the need for consistent terminology for annotations in order to enable interoperable annotation services. To this end, Marshall (2010, p. 42ff.) suggests three basic elements, which constitute the basic anatomy of any annotation: *body*, *anchor* and *marker* of an annotation (cf. Figure 2.5)[12].


BODY
MARKER
ANCHOR
Is this related to the notion of ecology by Star and Griesemer?

The growing interest for ecology has the unexpected effect of granting new relevance to a theology not so much interested in the salvation of humans as in the salvation of the whole creation –non-humans included. Since science studies has probed for many years several alternatives to the modernist divide between subject and object, it is interesting to cross the tools of science studies and theology to elicit a new contrast between nature and creation.

Figure 2.5.: Marshall's anatomy of an annotation (image source: Bélanger, 2010b, p. 13).

Body

The body of an annotation is the actual content that is added to a text. This content may reach from cryptic signs, like an asterisk or a smilie (example for rather implicit content), to elaborated comments (more explicit content)[13]. The annotation body is connected to an anchor that denotes the scope of a portion of text an annotation does relate to. Marshall (1998, p. 43ff.) suggests four scopes of anchors:

Anchor

Link to a collection e.g. "Chapter 7" (widest scope)

Node-to-annotation links are "annotations that don't visibly refer to any particular document element, but are localized within a document page"

Standard hypertext associations "from an anchored portion of the text to a note or commentary"

Word-to-word associations "particularly common in foreign language texts, in which the student translates a word into his or her native language, usually writing between lines of text" (narrowest scope)

According to Marshall (1998, p. 43ff.), mechanisms to establish connections (links) between an anchored portion of text and the actual annotation content can be arrows, brackets, braces, some custom marks or just proximity. Like the body of an annotation, an anchor may be explicit (e.g. an underlined portion of text) or implicit (only spatial relation between anchor and body). Fogli et al.

[12] Note: In an earlier study, Marshall & Brush (2004, p. 3) named the same three constituents *content* (= body), *anchor*, and *type* (= marker).

[13] The characteristics of annotations are described in more detail in chapter 2.6.

(2004, p. 1) introduce the *base of the annotation* as another useful term that allows us to differentiate between the actual annotation (= body) and the object that is being annotated (= anchor). As there can be many different annotations, the base of the annotation has to be marked with some kind of *visual identifier* that makes clear where a *base* starts and where it ends, and a *visual link* that points to the actual annotation (Fogli et al., 2004, p. 1). Marshall calls such visual identifiers *markers*: The marker is the actual visualization of the anchor and can be characterized by its *shape*, its *position*, and its *materialization* (e.g. color). In a study on handwritten annotation behavior, Marshall & Brush (2004, p. 3) analyzed more than 1 500 annotations to identify *underlines*, *highlights*, *circles* and *margin bars* as the most common types of markers. Furthermore, Marshall (1997, p. 134) observed the following characteristics for the *form* a marker can take: Marker

- In-text annotation vs. annotation on margins or on other blank spaces
- Telegraphic vs. explicit (meaning of) annotation
- Removable (post-its, dog-ears, bookmarks) vs. part of the text

Implications for linguistic annotation: Marshall's simple model can be applied to the area of linguistic annotation with a few limitations: Marshall does not address the issue that the bodies of several annotations might be identical, or in other words: one annotation content (e.g. the value "noun") can be related to several anchors. It is also important to note that a computer only needs the body of annotation and its anchor to interpret an annotated document. The visual marker is an appliance for human readers of the annotations. One common problem of linguistic annotation tools is that the XML-formalized anchors, which are primarily intended for machines, are oftentimes also used as visual markers for human readers. A user-friendly annotation tool should address the issue of alternative, human-readable annotation markers. This becomes even more challenging when we consider the different types of linguistic annotation (cf. section 3.4.3) and the common practice to add multiple, parallel annotations to one anchor (multi-level annotation).

2.5.2. A theory of human handwritten annotation

HHA vs. HEAD

Only a few years after Marshall's studies on the features and characteristics of annotations, Ilia A. Ovsiannikov (2002) finished her dissertation on "Annotation Databases for Distributed Documents"[14], introducing a *human handwritten annotation theory* (HHAT) in order to provide a framework for all kinds of *human handwritten annotations* (HHA), which she sees as the basis for the design of better electronic annotation systems, an area of research she refers to as *human electronic annotation of documents* (HEAD). In order to build a fundamental

[14] Note: The dissertation covers many issues from the frequently cited article "Annotation Technology" (Ovsiannikov et al., 1999).

HHAT, Ovsiannikov (2002, p. 6ff.) conducted an extensive case study, examining a large collection of manually annotated documents (HHA) from university students and staff. Similar to Marshall, her basic hypothesis is that electronic annotation can be enhanced and optimized by looking at the paper annotation process:

> A typical Tablet PC-based HEAD application will feature natural, paper-like annotation of electronic documents, provide advanced support for active reading, collaboration, knowledge lifecycle and ensure tolerance to electronic document changes. (Ovsiannikov, 2002, p. 4)

Similar to Marshall's body-anchor-marker troika, Ovsiannikov et al. (1999, p. 340) present some basic elements concerning the structure and appearance of an annotation, namely: *atoms*, *clumps* (a set of semantically related atoms) and *annotations*, which are linked to atoms or clumps.

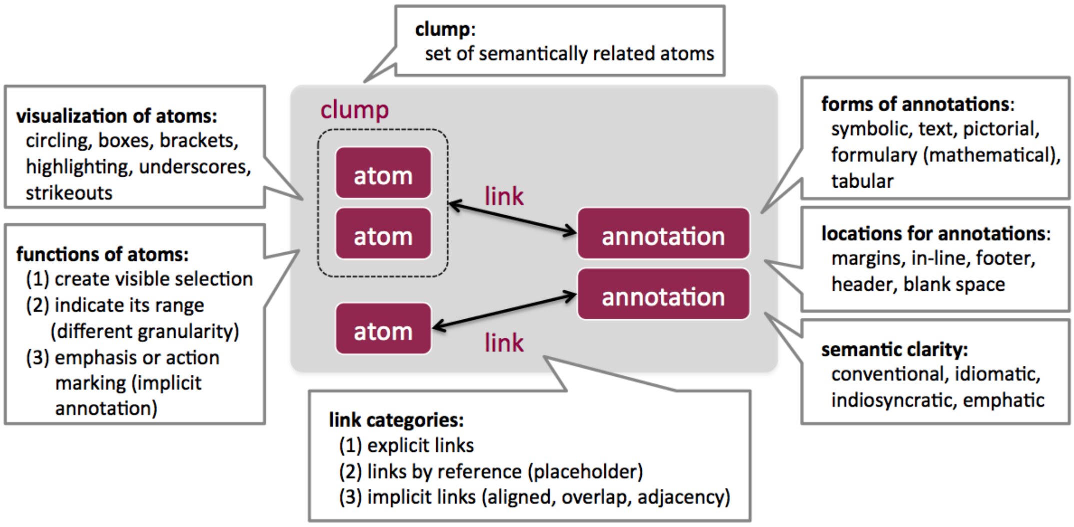

clump:
set of semantically related atoms
visualization of atoms:
circling, boxes, brackets, highlighting, underscores, strikeouts
functions of atoms:
(1) create visible selection
(2) indicate its range (different granularity)
(3) emphasis or action marking (implicit annotation)
clump
atom
atom
atom
link
link
annotation
annotation
forms of annotations:
symbolic, text, pictorial, formulary (mathematical), tabular
locations for annotations:
margins, in-line, footer, header, blank space
semantic clarity:
conventional, idiomatic, indiosyncratic, emphatic
link categories:
(1) explicit links
(2) links by reference (placeholder)
(3) implicit links (aligned, overlap, adjacency)

Figure 2.6.: Visualization of the basic concepts of Ovsiannikov's human handwritten annotation theory (HHA) (cf. Ovsiannikov et al., 1999; Ovsiannikov, 2002).

Atoms and clumps

An *atom* (cf. Marshall's *anchor* concept) is "the largest annotated indivisible unit of data of some format" (Ovsiannikov et al., 1999, p. 340), e.g. a piece of text, a sector of an image or a snippet of an audio file. The clump concept can be used to group a "set of semantically related atoms" (Ovsiannikov et al., 1999, p. 340). While Marshall defines the presentation of the anchor as an element of its own (the marker), Ovsiannikov et al. (1999) differentiate between the visualization of atoms and the functions they have. In her case study Ovsiannikov (2002, p. 11ff.) found that in HHA there are typical practices to visualize anchors (= atoms), or in other words to make a selection of text which is about to be annotated. Among the most common techniques are *circles*, enclosing text in *boxes* or *brackets*, *highlights* as well as the use of *underscores* and sometimes even *strikeouts*.

Visualization

Functions

Another important characteristic of such visualization techniques for anchors is the scope, which can have different granularity, ranging from the selection of single characters to the selection of multiple lines or even pages. It also seems interesting that practices for selecting an atom are not merely a means to create visual demarcations with a certain range, but also a means to emphasize certain parts of a document by using color or thicker lines (Ovsiannikov, 2002, p. 15ff.). Ovsiannikov (2002, p. 19) also observes that "atom markings" can be used as action instructions, like e.g. the deletion of a passage of text by crossing it out. Although emphasis is often used as a means to re-find important passages in HHA, it has not been considered in linguistic annotation. However, there may be some cases where emphasis can support linguistic annotation scenarios: (1) Emphasis can help to solve complex annotation tasks, which include multiple atoms that may be scattered over several pages. (2) The emphasis of interesting linguistic phenomena could also be visualized later, when the annotated data is queried and analyzed (e.g. optionally highlighted phrases in the KWIC-view of a corpus tool).

Annotation

An annotation (cf. Marshall's *body* concept) is "a datum created and added by a third party to the original document" (Ovsiannikov et al., 1999, p. 340), which may be anything from written, interlinear notes to personal drawings or embedded video-clips, placed on the margins of the document. Each atom is connected to one or more annotations. It is also possible to link the same annotation to a *clump*. By treating links as a means to connect atoms and annotations Ovsiannikov (2002, p. 22ff.) suggests three main categories of links:

Linking

Proper links have some explicit, visual linking symbol that works good for short distances (i.e. on the same page)

Links by reference are realized by means of a placeholder element that is referencing the actual annotation

Implicit relations are realized by methods like *aligned* (same axis) and *direct overlap* (on top of the text) or *adjacency*.

Forms

Ovsiannikov (2002, p. 19ff.) also offers a basic classification for the forms a handwritten annotation can take, and where it is likely to be positioned. Among the most common forms are *symbolic*, *text* and *pictorial* annotations, but also *formulary* (mathematical) and *tabular* annotations. Possible locations for annotations are: *on margins*, *in-line*, *in footer*, *in header* and *in blank* space. If the atom has not been marked explicitly, "the annotation is placed in such a way as to implicitly select it" (Ovsiannikov, 2002, p. 20). Annotations may, however, differ with regard to their *semantic clarity* (Ovsiannikov, 2002, p. 20-21): *Conventional* annotations are rather explicit, and can be understood by readers other than the actual annotator, as they make use of a commonly known vocabulary. Annotations can also be *idiomatic*, i.e. they are not as explicit as conventional annotations, but still understandable in the right context (e.g. an exclamation mark on the margin

Locations

Semantic clarity

next to a text passage). *Idiosyncratic* annotations are very hard to understand by readers other than the original annotator, as they use unconventional symbols. A fourth dimension for the categorization of semantic clarity of annotations is to be found with *emphatic* annotations, which are most commonly realized by using exclamation marks or stars.

Implications for linguistic annotation: Ovsiannikov's (2002) distinction between human handwritten annotation (HHA) and human electronic annotation of documents (HEAD) is important for this dissertation. While the approach to use insights from HHA for the enhancement of HEAD-tools seems plausible, both, Marshall's and Ovsiannikov's research leave behind some open questions for the case of linguistic annotation:

- What can be learned from HHA and HEAD for more formal annotation tasks like for instance linguistic annotation, and how does linguistic annotation relate to HHA and HEAD?
- Can annotation tools be improved by offering a special view that hides formal XML markup and instead offers elements which are known from HHA?
- Is it reasonable to offer two different views on linguistic markup, a formal view that contains the actual XML annotations, and a personal view that helps the annotator to read and understand the text he is about to annotate?

Another interesting aspect is the notion of *clumps*, which imply that one annotation may relate to a collection of atoms. As opposed to HHA, this seems to be a likely scenario in linguistic annotation, where technically the same annotation content can be related to several atoms. This issue is closely connected to the types of annotations that occur in HHA/HEAD and in linguistic annotation scenarios: Typically, HHA/HEAD scenarios use *ad hoc* annotations, i.e. the annotators do not use static, predefined annotations schemes. It is, however, possible that certain colors or symbols, like for instance an asterisk, may have a predefined meaning for the annotator. In the case of linguistic annotation, predefined annotation schemes are the norm. This is due to the need for formalized annotations, which demand to define and encode the possible annotation values in advance. The consecutive questions[15] would be:

- What kinds of different tasks do we have in linguistic annotation (POS-annotation vs. co-reference annotation)?
- Are there annotation tasks in HHA that are similar to linguistic annotation tasks?
- Can certain linguistic annotation tasks be implemented in a way that feels

[15] Some of these questions will be addressed in a case study on handwritten linguistic annotation (cf. chapter 2.8).

as natural and intuitive as it would in HHA (e.g. annotate POS like you would use a highlighter, etc.)?

Along the same lines, Marshall (1997, p. 135) addresses the problem of *cognitive overhead*, which is generated by using multiple annotation tools to accomplish the realization of the actual annotation:

> Using annotation tools (such as pens) demands a certain amount of attentional resource. As Thorngate suggests, making choices expends attention. . . . Students who use highlighters write fewer marginal notes than students who underline passages with pens. . . . In one of the copies . . . the student used a color-coded highlighting scheme that seems to have required a great deal of attention.

These observations illustrate that the need for easy-to-use and intuitive annotation tools and the requirements to realize more complex annotations seem to be conflicting. At the same time, linguistic annotation tools have to face the challenge of annotating a huge number of different annotation values systematically, i.e. according to a predefined scheme (so the computer can understand the annotations). Such multi-value annotation is cognitively very demanding, as the annotator must switch the *tool* for every annotation category. The annotator also needs to know the whole scheme in advance, i.e. he needs to know which categories are existent for annotation.

2.5.3. A formal model for digital annotation

Digital objects

While Marshall and Ovsiannikov are ranged more at the informal end of the continuum described in Figure 2.4, Agosti & Ferro (2007) can be seen as a prominent example for the more formal extreme. In their *formal model on digital annotations*, Agosti & Ferro (2007) build on existing concepts such as *digital objects*, which were first introduced by Bottoni et al. (2003)[16]. Accordingly, a digital object *o* is defined as a tuple of attribute-value pairs. The *type name* of a digital object indicates that it belongs to a certain category, e.g. a text file or an annotation. The attribute-value pairs describe the digital object and may include items like e.g. *author*, *title* or *creationDate*.

$$o = typeName((attr_1; val_1), (attr_2; val_2), ...(attr_n; val_n))$$

Mandatory attributes

Bottoni et al. (2003, p. 2) suggest two attributes as mandatory for any digital object: a *unique identifier* and some actual *content*. With digital annotations being one possible type category of digital objects, two additional attributes are suggested as mandatory: a *location*, describing "a reference to the annotated portion(s) of the object(s)", and a *placeHolder* for the "rendering of indications of the

[16] Bottoni et al. (2003) may be seen as founders of an *Italian school of formal annotation theory*, as their original idea of a *digital object* has been adopted by a number of fellow countrymen (Agosti et al., 2004; Fogli et al., 2004; Agosti & Ferro, 2007).

presence of annotations" (Bottoni et al., 2003, p. 3). These two attributes correspond to Marshall's anchor (= location) and marker (= placeHolder). Further attributes of digital annotations are its *role* (e.g. explanation or emphasis) and its *accessibility* to other users.

While Marshall's *body-marker-anchor* formula is easy to understand, it does not cover all annotation scenarios satisfactorily, as it is focused on handwritten annotations that support the human reading process. Although (much like Marshall) primarily interested in the use of annotation in the context of digital libraries, Agosti et al. (2004) argue for a more generic definition of the elements and contextual variables of annotations that could be applied to any scenario involving some form of digital annotation. Building on the aforementioned concept of digital objects, Agosti et al. (2004, p. 250) offer a comprehensive model of annotations (cf. Figure 2.7) in the form of an *entity-relationship model* (cf. Chen, 1976), extending the basic constituents of Marshall and Ovsiannikov by adding further entities to describe the concept of (digital) annotation extensively.

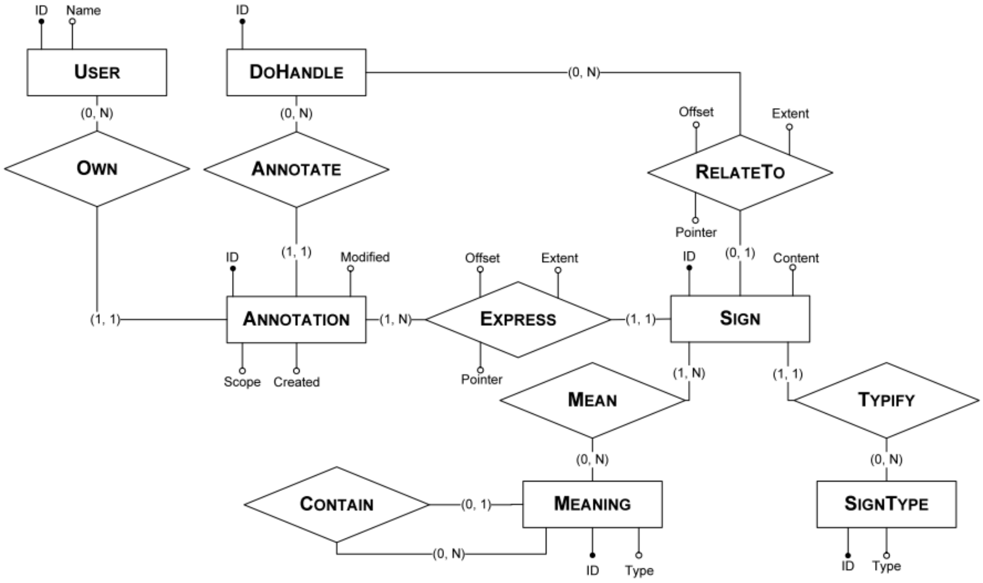


Figure 2.7.: *Entity-relationship model* of digital annotation (image source: Agosti et al., 2004, p. 250).

Modelling annotations: Most notably, Agosti et al. (2004, p. 249ff.) suggest to model annotations by differentiating its actual content, the *meaning*, and its materialization, the *sign* of the annotation. This idea is very closely related to Saussure's semiotic concept of *signifier*, the materialization of an object (e.g. as a word or an image), and *signified*, the mental representation of that object. A sign expresses an annotation and has a meaning which can recursively contain other meanings. The differentiation between *meaning* and *sign* seems plausible, considering more implicit forms of annotations, e.g. an asterisk or an exclamation mark, which can have quite different meanings for different readers. Meanings can be broad or narrow: more complex, broader meanings are simply expressed as a semantic hierarchy, i.e. a meaning can contain several other meanings. The semantics of an annotation are closely connected to its function, which can reach from cooperation and revision to personal comprehension and interpretation (Agosti & Ferro, 2007, p. 35). The semantic categories of annotations will be discussed in more detail in chapter 2.7. The sign entity can be described by a *sign type* entity. Signs describe the shape of an annotation and can be seen as its actual materialization. Agosti & Ferro (2007, p. 33) identify the following basic sign types for annotations which can be combined to more complex signs of annotations (compound signs):

Textual sign: semantics of the annotations are expressed by a piece of text

Graphic sign: graphic mark

Video sign: video fragment

Auditive sign: audio fragment

To address the cooperative aspects of annotations, Agosti et al. (2004) suggest a *user* entity that can own an annotation. It is vital to know the actual author of an annotation in order to discuss specific characteristics of an annotation as described by Marshall (1998) (cf. chapter 2.6). A few years later, Agosti & Ferro (2007, p. 21ff.) also suggest other roles that are involved in cooperative annotation scenarios: Besides the author of an annotation there can also be *single users* and *groups of users* for which issues concerning authentication, authorization and permission can be defined. According to Agosti & Ferro (2007, p. 23), an annotation can have the following access permissions: it is *denied*, i.e. no access whatsoever is allowed, it has *read only* access, or it has access for *read and write* operations. The scope of access may reach from *private* (can be accessed only by its own author) to *shared* (can be accessed only by a designated set of groups of users) or even to *public* (can be accessed by all users) (Marshall, 1998; Agosti & Ferro, 2007).

Permission

Scope

Connecting annotations to information resources: According to Bottoni et al. (2003, p. 2), a digital object is a generic concept that captures all kinds of *source files*, e.g. a piece of text or an image, but also secondary data like digital annotations. This is the formal cornerstone for a recursive annotation mechanism that can be utilized to annotate annotations, which in turn are digital objects. Annotations and annotated documents, which are both classified as digital objects, can, however, be distinguished by their temporal relationship, i.e. before an annotation can be created, the existence of an object that will be the target of the annotation is implied, or in other words: you cannot annotate in the void, because an annotation always relates to some primary data. Once an annotation is successfully created, it may become a target for other annotations. Agosti et al. (2004) introduce the *DOHandle* entity, which stands for a generic digital object that can be referenced and identified via a handle. Examples for existing mechanisms to realize such handles are the *Uniform Resource Identifier* (URIs), *Digital Object Identifier* (DOIs), *OpenURL*, *Persistent URL* (PURL), etc. (Agosti & Ferro, 2007, p. 20). The *DOHandle* is useful to express the circumstance that not only a piece of text can be annotated, but an existing annotation might be annotated as a digital object that can be referenced by its handle, as well. Handles basically serve as a linking mechanism to connect annotations and annotation targets (Agosti & Ferro, 2007, p. 24ff.) in one of the following two ways:

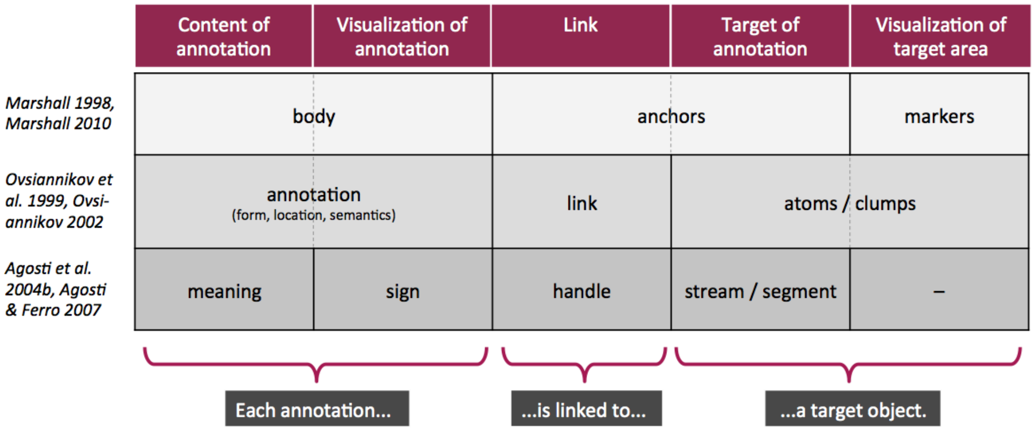
Annotate links describe intra-digital object relationships, i.e. they link within the same document

Relate-to links describe inter-digital object relationships, i.e. they link to another document

Eventually, Agosti & Ferro (2007, p. 26-31) introduce the *stream* concept to describe and represent the actual content of a digital object as an "ordered sequence of symbols", and *segments* (Agosti & Ferro, 2007, p. 31-33) as a mechanism for selecting specific parts of such a *stream* of symbols (e.g. single words of an HTML document).

Implications for linguistic annotation: Agosti et al. provide by far the most comprehensive view on the elements and determining factors of digital annotations. While some of the basic concepts they describe are also covered by Marshall's and Ovsiannikov's more informal models, a number of new ideas is introduced, which can be applied to the domain of linguistic annotation. The notion of *recursive annotations* (annotation of annotations) can be easily transferred to linguistic multi-level annotation. The *sign type* concept can be used to cover the issue of annotation markers with different scopes in a systematic and consistent manner (e.g. *sign type = word*, *phrase*, etc.). While the previous models did not explicitly address the issue of *cooperative aspects*, they are considered by Agosti et al. The notion of *users* and *user groups*, *access permissions* and *access scopes* (*private/shared/public*) will be relevant for the collection of requirements for linguistic annotation tools (cf. chapter 4.4.2)

Summary of the different models: Up to now, several approaches to describe the basic constituents of an annotation have been discussed – some more formal and more extensive than others. Figure 2.8 sums up the most important elements of each theory and relates them to each other. It shows that an annotation consists of the actual content (and its visualization) and an anchor (and its visualization) that selects what is to be annotated. Both elements are connected by some kind of link, to make clear which annotation belongs to which anchor. In a nutshell, the existing theories can be simplified to the following formula: *Each annotation is linked to a target object*.



	Content of annotation	Visualization of annotation	Link	Target of annotation	Visualization of target area
Marshall 1998, Marshall 2010	body		anchors		markers
Ovsiannikov et al. 1999, Ovsiannikov 2002	annotation (form, location, semantics)		link	atoms / clumps	
Agosti et al. 2004b, Agosti & Ferro 2007	meaning	sign	handle	stream / segment	–

Figure 2.8.: Overview of elements of annotation throughout literature.

2.6. Characterizing annotations

As a result of her studies on the annotation behavior of university students in textbooks, Marshall (1998) identified altogether seven annotation characteristics that can be used to categorize most annotation contents[17]. Although Marshall describes exemplary values for each of the *seven dimensions of annotation*, each dimension is intended as a continuum (cf. Table 2.3).

Implications for linguistic annotation: The following list is an attempt to characterize linguistic annotation according to Marshall's seven dimensions:

Formality Linguistic annotation is clearly at the more formal end of the continuum, as it is mainly intended for machine processing and thus needs a high degree of structuring and standardization.

Intelligibility This dimension is closely related to the degree of formality. The highly formalized linguistic annotations must nonetheless be discussed ambivalently, as they are without any doubt highly legible for machines, but rather cryptic for human readers, especially to readers who are not familiar with the notation and syntax of XML-based markup languages.

Function Typically, the function of linguistic annotation is *writing*, i.e. adding new information to a text that explains some linguistic aspect in detail. It seems promising to add a second layer of annotation which is intended for *annotation as reading*, aiding the human annotators to comprehend and structure the primary text, which could lead to better, more correct annotations.

[17] Several consecutive studies (Ruvane, 2006, 2007; Rowe, 2011) have shown that Marshall's framework can be used successfully to characterize annotations.

Dimension	Description of possible values
(1) Formality	*Formal*: Structured, standardized metadata; intended to ensure interoperability *Informal*: Less structured notes; intended for personal use
(2) Intelligibility	*Explicit*: High legibility; could be read and understood by others than the annotator *Tacit*: Low legibility, more cryptic, incomplete notes, which have meaning for the annotator alone
(3) Function	*Annotation as writing*: Wordy, lengthy explanations; participatory aspect of text *Annotation as reading*: Main goal is structuring and organizing content
(4) Focus	*Hyperextensive / Extensive*: References and links to other texts (hypertext features) *Intensive*: Occupation with a single text, no external references etc.
(5) Value	*Permanent*: Annotation has value beyond its first creation and use *Transient*: Annotation has limited value and usefulness
(6) Intent / Audience	*Published*: More authorial annotations, intended for public *Private*: More personal annotations, not intended for other readers
(7) Scope	*Global / Institutional / Workgroup / Personal*: Scope of the users/readers of the annotations

Table 2.3.: Marshall's seven dimensions of annotation. The column with possible values for each dimension is based on a description by Bélanger (2010b, p. 21).

Focus The focus of most cases of linguistic annotation can be described as *intensive*, i.e. they are concerned with only one primary text. Intertextual annotations, e.g. anaphors scattered across multiple texts, are nonetheless imaginable.

Value The value of linguistic annotation is an important dimension: As corpus creation is a very laborious and time-consuming task, annotated corpora should be designed with regard to re-usability.

Intent The intent for most linguistic annotations is to be published as an accessible corpus, i.e. the annotations are intended for a wider public. It might, however, be helpful for the annotators to have a private layer for personal notes and comments.

Scope Most linguistic annotations are applied in a greater project context, thus having an innately wider, global scope.

While Marshall's dimensions can be used to describe the most important characteristics of linguistic annotations, we still have to address the different functions an annotation can have. The next section reviews appropriate literature to identify basic functions of an annotation.

2.7. Functions of annotations

The main function of annotations has been introduced at the beginning of this chapter: annotations are added to a piece of text to make intrinsic, hidden information explicitly available. This section shows that there are many different ramifications when it comes to describing the exact function and aim of an annotation. It also presents an overview[18] of some of the most influential definitions on the functions of annotations that can be found throughout literature.

Ovsiannikov et al. (1999)

Ovsiannikov et al. (1999, p. 336) conducted a survey to investigate the main applications and uses of annotations. The results of their study suggest four main functions of an annotation: People annotate to (1) *remember* the contents of a text for future use, and to facilitate the "recollection of the main ideas" at a later point in time. Annotations are also used to (2) *think*, i.e. the annotator writes down personal associations, reflections and thoughts on the subject. Another function identified by Ovsiannikov et al. (1999, p. 336) is to (3) *clarify* the contents of a text, which means to "rephrase", "reshape" and eventually "personify" complicated issues so that they match the mental maps and metaphors that are already existing in the reader's head. The last function describes a collaborative aspect of annotation. (4) *Sharing* annotations is a means to communicate with other annotators, and also occurs frequently in collaborative editing and reviewing processes. The last function clearly differs from the other three, as it is not a personal annotation practice[19].

Marshall (1997)

Marshall (1997, p. 135ff.) identifies some finer-grained functions of annotations: Annotations can be used as (1) *procedural signals*:, i.e. annotations are utilized to prepare the text for re-reading and future processing, e.g. by crossing out irrelevant sections. Annotations are also used as (2) *placemarks and memory aids*, which means annotations mark parts that need to be accessed at a later point in time. Some annotations function as (3) *in situ locations for problem-working*, where the annotator adds ideas or notes that help to serve a certain problem with the text. Oftentimes annotations are used as a (4) *record of interpretative activity*. According to Marshall, such annotations include the interpretation of unfamiliar language as well as the interpretation of a complex document structure, or the interpretations of the actual content of a text document. Another function of annotations is to create a (5) *visible trace of the reader's attention*, which helps to document the reading process as well as to make a text more accessible for future use. Finally, annotations may serve as (6) *incidental reflections of the material circumstances*, including some thematically unrelated doodles, or maybe a telephone number that, in absence of some other piece of paper, was hastily jotted down on the margins of the text.

[18] This section builds upon an overview provided by Bélanger (2010b, p. 22ff.).

[19] Marshall & Brush (2004) report on the differences between personal (cf. Functions 1-3) and public (cf. Function 4) annotation behavior in more detail.

Wolfe & Neuwirth (2001)

Wolfe & Neuwirth (2001, p. 336ff.) discuss the "future of annotation" in the context of emerging digital technologies. The authors identify four main functions of annotations: Annotations are actively generated and used to (1) *facilitate reading and later writing tasks* (personal context). At the same time they can be used passively by reading annotations of previous annotators[20] (public context), or as Wolfe & Neuwirth put it, by (2) *eavesdropping on the insights of other readers*. Another function from the public context is to (3) *provide feedback to writers or promote communication with collaborators*. Finally, annotations have the function to (4) *call attention to topics and important passages*.

Agosti & Ferro (2007)

Agosti & Ferro (2007, p. 35) mention the following categories, to classify the different types of meaning an annotation can take, which strongly resemble the functions that have been introduced so far: Annotations as a means for (1) *comprehension and study* of a text, annotation as a means for (2) *interpretation and elucidation*, i.e. annotate to make a text more comprehensible, and annotation in a function to support (3) *cooperation and revision*.

After this short review of appropriate literature it becomes obvious that many of the formulations from different authors are quite similar. Table 2.4 shows an attempt to unify different functional descriptions of annotations into more generic categories. Besides the incidental and oftentimes unintentional, off-topic annotations in the form of doodles or unrelated scribbles, there are five basic functions of handwritten annotations that can be observed in traditional pen and paper scenarios as well as in a digital context (e.g. annotation on a tablet computer).

Active reading

Implications for linguistic annotation: If we want to discuss the above functions in the context of linguistic annotation, we have to keep in mind that these functions were primarily defined for a context of *active reading* (Schilit et al., 1998, p. 249). An active reader critically reflects on the text while he is processing it and is prone to take notes and add annotations. Ovsiannikov (2002, p. 27) stresses the "intimate and inseparable" relation between annotations and the reading process, which is a point worth to consider in the context of linguistic annotation: It can be argued that the reading process of texts that are annotated with linguistic information is fundamentally different from the active reading scenarios described above. Linguistic annotators are more interested in single words or phrases rather than in the content of the document as a whole. As a consequence they typically scan the text for a word or groups of words they want to annotate rather than reading the whole text to understand its contents. This assumption is backed up by Tomanek et al. (2010), who measured the error rate as well as the time to annotate *named entities* (NE) with tags like "person" or "location". The study showed that contextual information beyond the sentence scope did not influence the annotation speed and the number of annotation errors significantly. Additional eye-tracking data in the same study indicates that

[20] Cf. Marshall's idea of "an ecology of annotation" (Marshall, 1998).

Category of function	Ovsiannikov et al. (1999)	Marshall (1997)	Wolfe & Neuwirth (2001)	Agosti & Ferro (2007)
Comprehension	Think	In situ way of working problems	–	Comprehension and study
Interpretation and reflection	Clarify	Record of interpretative activity	–	Interpretation and elucidation
Memory aid	Remember	Placemarking and memory aids	Facilitate reading and later writing tasks; call attention to topics and important passages	–
Restructuring and customization for facilitated reading	–	Procedural signals (for future processing); visible trace of reader's attention	Facilitate reading and later writing tasks; call attention to topics and important passages	–
Communication and collaboration	Share	–	Eavesdrop on the insights of others; provide feedback and promote communication	Cooperation and revision
Incidental annotation	–	Incidental annotations and doodles	–	–

Table 2.4.: Overview of the different functions of an annotation.

annotators mainly look at the phrase that needs to be annotated, indicating that they do not need much contextual information to achieve their annotation tasks.

Scope

Another difference between paper annotation and linguistic annotation is the scope (private vs. public) of the annotation. Linguistic annotators do not primarily annotate for themselves, but rather for other readers. These "readers" are mainly computer programs (that are used to analyze the annotated language data), implying a higher degree of formality, which, compared to handwritten paper annotations, might feel less natural and intuitive for the annotators.

Comprehension

Although the reading behavior in a linguistic context may be different from active reading processes during the actual annotation process (scan rather than read), it is vital for the annotator to read the text before the linguistic annotations are applied. While contextual information may be less relevant for trivial annotation tasks such as POS or NE annotation, more complex tasks, e.g. the annotation of co-reference, calls for a more intense studying of the whole text. Therefore, it might be helpful to provide a layer of personal, less formalized annotations for the human annotator, which can be displayed optionally during the annotation process. Such a personal layer could realize all functions identified for handwritten annotation.

Interpretation, reflection

Interpretation could be the most important function of linguistic annotation, as it is all about making implicit linguistic phenomena explicitly available and machine-processable. The interpretation, however, has to be formulated in a much more formalized way than it would occur in handwritten contexts. The challenge for linguistic annotation tools is to visualize the formalized interpretative annotations in a user-friendly, human-readable format. This becomes even more challenging when we consider that linguistic annotation is much more abundant than handwritten annotation. If, for instance, we annotate the POS of a text, we will have an annotation for every single word, not to mention the possibility to annotate a digital text on several, parallel levels. Here the aforementioned gap becomes obvious again: humans like to annotate in an HHA fashion and have difficulties annotating and reading formalized XML markup, while computers struggle to read HHA and need standardized / conventionalized annotations to perform analyses of the data.

Memory aid

Memory aids in linguistic annotation could be realized on a personal annotation layer, to facilitate complex annotation tasks that are carried out through a longer period of time. It might also be helpful to have personal memory aid annotations available when analyzing the data at a later point in time. It is likely that while annotating the data, interesting research questions will emerge.

Restructuring

This function could be very helpful for linguistic annotation tools. While the tool can keep the original version of the text in some kind of internal representation, it would be possible to restructure the document for the human annotator, e.g. by splitting a longer text into smaller units of text.

Collaboration

Collaborative linguistic annotation is an important issue, as most annotation tasks are very complex, and are thus carried out by several annotators. The biggest challenges here are quality assurance and consistency of annotation. Linguistic annotation tools should provide features that support asynchronous communication in the form of notes and comments as well as a basic version control mechanism. Ideally, the tool provides quick and easy access to the annotation scheme and the gold standard that was defined for the annotation project for all annotators who work on that project.

Incidental annotations

As incidental annotations are more of a side-effect than a function of an actual annotation task, they will not be considered as an explicit function for linguistic annotation tools. It is, however, possible that human annotators will use certain functions of a linguistic annotation tool to create incidental annotations that are similar to those known from paper annotations.

2.8. Case study: Handwritten linguistic annotation

So far, this chapter has shown several attempts to translate concepts and behaviors from the area of *human handwritten annotation* to applications in the digital annotation realm, which may be subsumed as *human electronic annotation* (cf. Ovsiannikov, 2002). However, the preceding section has illustrated that linguistic annotation functions in rather unique ways as compared to active reading annotation contexts. At the same time, many human annotators struggle to annotate linguistic features on the code level (cf. Flynn, 2006, 2009), as the formalized XML markup is primarily intended to be read and understood by machines. Attempts to provide annotation tools that abstract or even hide the technical complexity of formal markup languages are often characterized by insufficient usability of the interfaces (cf. Burghardt, 2012). The latter point will be addressed by getting insights for enhanced linguistic annotation interfaces from human handwritten annotation behaviors. While related work by Ovsiannikov (2002) and Marshall (1997) describes annotation studies in an active reading context, only few findings can be transferred to the area of linguistic annotation. Keeping in mind that human handwritten annotation still is the most natural and intuitive way of annotating text, I designed a study that observes annotation behaviors for linguistic tasks in a pen and paper scenario. The rest of this section describes the design of the study[21] as well as the main findings, which will also be discussed to derive implications for the design of linguistic annotation tools.

2.8.1. Research questions

The case study is motivated by the following overall research question: How do humans annotate linguistic phenomena in a pen and paper context, and what can be learned for the design of digital, linguistic annotation tools? This generic research question will be addressed by answering the following, more concrete questions with the study:

Q1 How are different linguistic annotation tasks (with regard to the anchor scope) realized by means of form and color?

Q2 How do users annotate relations for different anchors beyond the sentence level (e.g. for co-reference annotation tasks)?

Q3 How do users put parallel annotations (i.e. multiple annotations that belong to one anchor) into practice?

[21] The study was designed and conducted as part of a seminar on "Digital Humanities" in the winter semester 2012/2013 at the University of Regensburg. Saskia Gerstmeier and Isabella Hastreiter, both students in the information science master′s program, were involved in the initial study design and carried out most of the user testing and data analysis. For this reason, I will describe the design of the study from a *we* perspective.

2.8.2. Study description

Test participants

A total of fifteen participants (seven male, eight female) was recruited for the study of handwritten annotation behavior in a linguistic context. All participants were Bachelor, Master or PhD students at the University of Regensburg, with a background in humanities. The average age was twenty-four years, with a range of seven years.

Previous knowledge

Besides such demographic data we collected information about previous knowledge with linguistic annotation and digital tools, but also personal behavior and practices for annotation in general. Thirteen participants reported that basic linguistics were part of their studies a the university. The participants of the study were also asked about their digital annotation behavior: Five participants read and annotate text digitally, i.e. in PDF format. Only one participant has experience with linguistic annotation tools.

Selection of texts

As all participants were German native speakers, we chose a column from Silke Burmester, who writes for the German news site Spiegel-Online.de[22]. The text was chosen because it uses simple language (reduced distraction), but at the same time describes a rather unusual topic (increased interest/attention). The text is 505 words long and printed with double line spacing on two DIN A4 pages, to provide enough space for annotations. The whole text was printed in the same font-family and font-size, no segments (e.g. the headline) were visually emphasized.

Task design

We designed a total of six different annotation tasks on the basis of an anchor scope taxonomy as suggested by Burnage & Dunlop (1992, p. 152). The authors describe a classification of different markup in the *British National Corpus* (BNC) which uses the scope of the anchors as basic categories (also cf. chapter 3.4.3): character scope, word scope, phrase scope, sentence scope, structural scope (paragraphs etc.) and text scope (e.g. bibliographic information)[23]. Tasks 1-5 were designed to elicit annotation behavior for these different scopes:

Task 1 Structural level: Mark all paragraphs in the text.

Task 2 Character level: Mark all umlaute (ä, ö, ü) in the first paragraph.

Task 3 Word level: Mark all nouns in the first two paragraphs.

Task 4 Phrase level: Mark all noun phrases on the first page.

Task 5 Sentence level: Mark all relative clauses on the first page.

Task 6 was designed to analyze annotation behaviors beyond the sentence level, by means of relations between two or more anchors.

[22] Title of the article: "Schnauzbart hilft gegen Männerkrebs", November 4, 2012, available at http://www.spiegel.de/kultur/gesellschaft/silke-burmester-ueber-den-prostatakrebs-aktionsmonat-movember-a-865037.html

[23] Metadata that describes the text as a whole will be treated as a special type of linguistic annotation. For this reason it was not included as a task in this study (cf. section 2.4).

Task 6 Mark the relations between personal pronouns and the nouns they are related to / they stand for.

Test setting

The study was carried out over a period of two weeks. At the beginning, the participants were asked to fill in a short questionnaire that shed light on demographic aspects, previous knowledge in the area of linguistic annotation, and general behaviors with regard to digital reading and annotation. The participants were able to choose from a wide variety of pens in various colors. They also had access to other tools, such as a ruler or a rubber.

Moderator

Each test was conducted by a moderator and an observer. The moderator basically guided the participants through the test by reading out the six tasks one after the other, and also provided examples and short explanations for the more challenging tasks (Tasks 4-6). She also stressed that we were not interested in the linguistic knowledge of the test persons, but rather in their annotation behaviors. Any questions or uncertainties related to the tasks were answered by the moderator. In order to motivate the tasks, the participants were told to annotate the texts in such a way that they would be able to answer basic questions about the text in a few weeks[24].

Observer

The observer remained in the background and collected notes about the test process, especially about peculiar annotation behaviors. The test participants were asked to *think aloud* and comment on their behaviors before the test. Additionally, the annotation behaviors were recorded by means of a small camera (cf. Figure 2.9). At the end of the study, the observer used her notes to ask some final questions about the respective annotation behaviors of the participants.

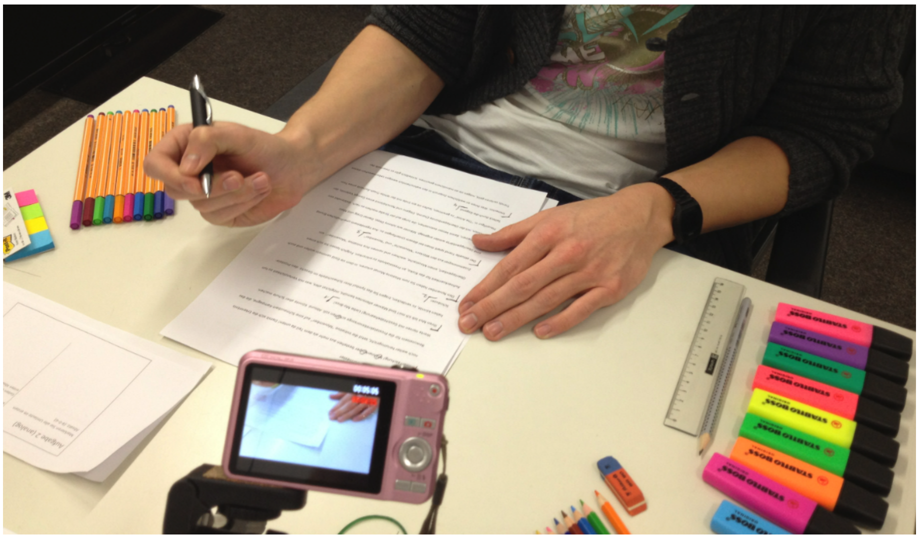


Figure 2.9.: Photograph of a pretest session, showing the test setting that was used throughout the actual evaluation.

[24] Exemplary question for Task 3: "How many nouns appear in the first two paragraphs?"

In the following sections, the main findings will be structured in a way to answer the previously formulated research questions (Q1-Q3). The implications for the design of digital annotation tools will be subsumed at the end of each section.

2.8.3. Findings on annotation forms and color usage

As Tasks 1-5 each addressed different anchor scopes, we expected to observe the use of different annotation types. Most of the identified types are very similar to those described by Marshall (1997, Table 2, p. 135). Depending on whether the scope of the anchor was wider or narrower, the annotations were placed within the text (or in few cases between the lines) or on the left margins. Table 2.5 shows the frequencies of the different annotation types for each of the five different anchor scopes.

Annotation type	**Character scope**	**Word scope**	**Phrase scope**	**Clause scope**	**Paragraph scope**	**Total**
Circle	5	13	5	3	–	26
Underline	2	11	6	5	–	24
Highlight	7	2	3	2	–	14
Margin marker (start marker)	–	–	–	–	6	6
Pair of brackets (within text)	–	–	2	3	–	5
Vertical span (on margin)	–	–	–	–	5	5
Box	–	2	2	–	–	4
Label	–	–	2	2	–	4
Inline marker (end marker)	–	–	–	–	4	4
Little arrow (above character)	1	–	–	–	–	1
Total	15	28	20	15	15	93

Table 2.5.: Annotation types categorized by anchor scopes.

Only the presence or absence of a specific marker (binary choice) were counted, not the actual number of single usage of a marker. Such counts would have been problematic due to the fact that the task success was quite different between the participants, which may be explained by different levels of pre-existing linguistic knowledge as well as the degree of motivation. As we told every participant right from the start of the evaluation that we were not primarily interested in how good or thorough they would achieve the annotation tasks, but rather in their annotation behavior, some were more ambitious in finding all annotation targets in the text than others. Accordingly, Table 2.5 usually shows a total of fifteen annotation types for each scope, one type for each participant. This is not the case for the word and phrase scope, as Task 6 (co-reference annotation) required annotations on the word and phrase level, too. Some participants re-used their previous annotations to achieve Task 6, others made new annotations on the same anchor (parallel annotations).

Scopes

All but one participant used highlights (7), circles (5) and underlines (2) as markers to annotate single characters. One participant used little arrows to point at the respective characters. For the annotation of single words, many participants used circles (13) and underlines (11) as well as boxes (2) and highlights (2). Phrase annotation, i.e. the annotation of groups of words, is carried out rather heterogeneously. Again, we find underlines (6), circles (5), highlights (3) and boxes (2), but also the use of pairs of brackets (2), which mark the beginning and the end of the phrase anchor. Two participants used concrete labels (e.g. "NP" for noun phrase, or just "N") in combination with a scope marker (e.g. a brace or a circle; the scope marker was not counted, only the label). The annotations for the clause level reveal a similar mix of annotation types: underlines (5), circles (3), pairs of brackets (3), highlights (2), and again two cases of explicit labeling. Interestingly, two participants did not mark the whole clause, but only the first word of the relative clause, to indicate where it begins. The paragraph scope annotations stand out from the rest of the annotations. Participants used annotation styles that were not used for the other tasks. Also, the left margins of the page were used very often to place the annotations. Among the most common annotations are margin markers (6) that indicate the beginning of a new paragraph. These markers are used very heterogeneously, for instance a rotated "T" or "L" letter, or a simple dash. Some participants also used markers inside the text (4), placing them at the end of a paragraph. One of them even used a combination of start and end markers. The last category of paragraph annotation styles can be best described as vertical span markers (5) which are placed vertically on the left margin and allow to embrace multiple lines of text by means of a large bracket or just a line. Two participants added running numbers to their vertical brackets.

Color use

The participants were able to choose different colors from a huge palette of available colors. The basic choice of color was a black pen or a grey pencil (used by ten participants). Only one participant worked without any colors, but rather used his black pen for all annotation tasks. Color use varied from 1 to 7 different colors, with an average of about five different colors per annotator (cf. Table 2.6).

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
6	6	5	4	6	5	7	5	1	5	6	4	6	5	6

Table 2.6.: Overview of the use of different colors by the fifteen participants (P1–P15).

Synopsis

Circles (26), underlines (24) and highlights (14) seem to be the most used annotation types for linguistic annotation with anchor scopes that range from a single character to a whole sentence. In this regard our case study on "human handwritten annotation in a linguistic task context" is in accordance with the findings of Marshall & Brush (2004), who conducted a study with eleven graduate students to analyze the relationship between personal and public annotations.

They found that for a total of 1 535 single annotations, 54.9% are underlines, 16.3% are highlights and 9.1% are circles. Color is used as an obvious means to indicate different annotations. However, colors are not used ubiquitously (the average participant used five different colors), but rather in combination with different annotation styles. Another key finding is that annotation styles for anchor scopes beyond single sentences are used very heterogeneously and look quite different from the other annotation styles. Very often, they are placed on the left margin of the page. Apart from the annotation type *labels*, all other annotations were *anchor only* annotations, i.e. anchor marker and annotation content are the same (Marshall & Brush, 2004, p. 351). Obviously, in handwritten annotation contexts it is commonplace to use preferably anchor only annotations for means of efficiency and annotation speed. Marshall & Brush (2004, p. 353) report a frequency of 82.2% anchor only annotations in their study. In this aspect, typical handwritten annotations are clearly different from linguistic annotation tasks, which can quickly become very complex and are hardly understood without explicitly documenting the annotation content. In this study, only two participants added a legend on the top margin of the first page that described the meaning of different colors. The label annotations stand out, as they are the only *compound* annotation types in this study that consist of an anchor and an some kind of annotation content. We will see that annotation types look quite different for task 6 (co-reference annotation).

Implications for linguistic annotation:

- Most annotations in the case study were carried out as *anchor-only* annotations (prevailing forms: circles, underlines and highlights). While the visualization of annotations by means of their anchor in the primary data seems legitimate for a digital annotation tool, it is mandatory to also provide explicit annotation content that is stored in the annotation file. It also seems unrealistic that a human annotator will be able to differentiate multiple annotations only by means of different anchor visualizations, as a linguistic annotation project typically comprises a great number of different annotation values, which are applied to large primary data documents. Therefore, a linguistic annotation tool will have to provide a mechanism that allows the user to (optionally) display the explicit *content* of an annotation.

- Apart from the case of multi-line annotations for paragraphs, there seem to be no preferences for using specific anchor forms for different scopes (e.g. single letters vs. phrase). However, a slight tendency toward using forms that do not cram the primary data (e.g. mark the beginning and the end of an annotation or utilize margin space) could be observed for larger anchor scopes. Accordingly, annotation tools should support showing and hiding different types of annotation interactively, i.e. the user can choose which annotations to display, and which to hide.

- Color seems to be an essential means to realize different annotation values. Although the participants had access to more than twenty different colors, they used only five colors on average. This implies that elaborate color schemes may be hard to differentiate, which means that annotation tools should not rely on color only to unambiguously distinguish different annotation values in the interface.

2.8.4. Findings on the creation of relations

Up to now, we have seen how the participants of the study annotated single anchors of varying scopes. Task 6 was designed to elicit annotations that reach beyond the sentence level, as they relate different anchors from different sentences (and even pages) to each other. The participants annotated anaphoric relations between noun phrases and personal pronouns. An anaphor is a coreference relation between two or more constituents, usually between a noun phrase and a personal pronoun. The first constituent (cf. example: *ante_1* and *ante_2*) is called the antecedent, the successive constituents (cf. example: *ref_1* and *ref_2*) refer to the antecedent backward, thus creating an anaphoric relation between the constituents.

Example: [*Mr. Smith*]$_{\text{ante\_1}}$ bought [*himself*]$_{\text{ref\_1}}$ [*a fancy new hat*]$_{\text{ante\_2}}$. [*He*]$_{\text{ref\_1}}$ was really looking forward to wearing [*it*]$_{\text{ref\_2}}$ at the big party on Saturday.

In order to annotate an anaphoric relation between two or more constituents, the anchors of antecedents and pronouns had to be annotated first. Eleven participants drew circles, two drew a box and one underlined the pronoun. One participant did not annotate the pronoun anchors at all, but rather indicated their role by placing them at one end of a relation arrow. If the anchor of an antecedent had been annotated before (for another task), the existing anchors were used by seven participants. Five participants underlined the anchor, three drew a circle, two drew a box and one annotator drew a little cloud around the antecedent. Two participants did not explicitly annotate the antecedents, but indicated its role by means of relation arrows[25].

Establishing relations

The main challenge for this task clearly is to establish a relation between the anaphoric constituents (cf. Figure 2.10). All but one participant used unidirectional arrows (9) or just lines (5) to establish such connections. The direction of the arrows was mostly pointing toward the antecedent (one participant chose the opposite direction). The lines and arrows either reached directly from the pronouns to the antecedent, thus creating a tree-like structure (a), or they were connected in some sort of chain, were only the first pronoun pointed to the antecedent, and the other pointed to the preceding pronoun (b). Two annotators

[25] Note: Some participants used more than one style for annotating the antecedent anchor. Therefore the number of all different annotation styles (20) here is bigger than the number of different participants (15).

also made use of short arrows[26], i.e. the arrows did not reach from pronoun to antecedent, but rather were used as deictic devices that indicate the direction and position of the antecedent (c). One participant made no explicit relation, but rather relied on a unique color and form scheme to indicate relations between antecedent and pronouns[27].

Position of relations

Ten participants chose to draw their arrows and lines directly through the text (a), while two tried to interrupt the lines so they would not obscure the text (d). One participant even tried to draw the lines around the text using the margins of the page (e), but realized after two arrows that this was not easily achieved, especially when the constituents are not close to a margin. The participant then switched to a mode where he labeled the pronouns with "PP". Two other participants made use of an indexing system (f)[28], mainly to establish a relation from constituents on the second page to the antecedent on the first page.

Relations across two pages

Another big challenge for the annotators was to indicate that a relation exists between constituents that are located on different pages. Seven participants did not annotate any second page relations at all (partial task fail), three annotators used arrows that pointed from the constituents to the left outer margin together with short comments like "see other page". Two participants drew short, unlabeled arrows that pointed to the top or toward the left margin, to indicate that the antecedent was on the other page. Two participants used indices (asterisk and running numbers from 1-3) to indicate the relation to a constituent that is located on another page. One participant duplicated the antecedent by writing it on the second page, and used it as if it was on the same page.

Color use

A final note on color use: All fifteen participants consistently used one color for the annotation of relations, including the actual relation as well as the antecedent and pronoun anchors.

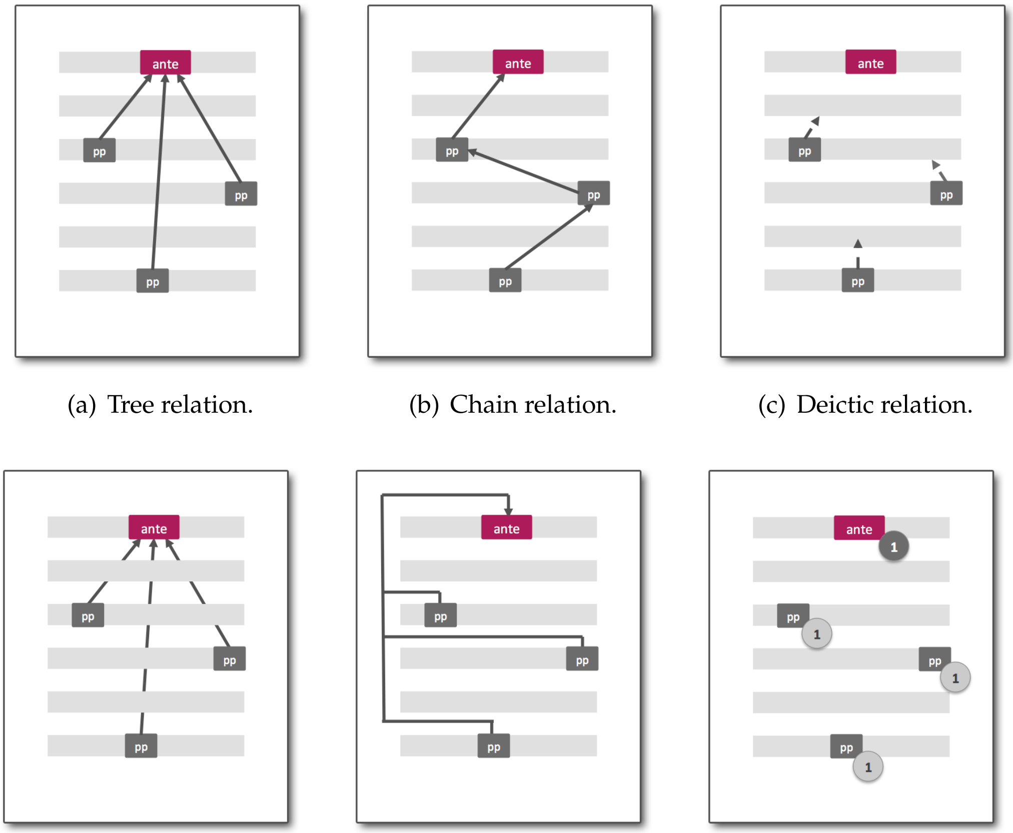
Implications for linguistic annotation:

- In a digital annotation scenario, it is vital to explicitly mark the anchors that are part of the relation annotation in order to be able to create a representation in the digital annotation format. The case study shows that annotating the anchors of a relational annotation is already performed by many of the participants, i.e. it seems to be a natural annotation behavior that will not generate much overhead when required in a digital annotation tool.
- Most relational annotations were realized by means of arrows or connecting lines between the participating anchors. Digital annotation tools should

[26] One annotator exclusively used short arrows, the other annotator used a combination of short and long arrows.

[27] This participant only encoded one anaphoric relation. If he had annotated more, his approach obviously would not have worked.

[28] One participant used (together with arrows on the first page) running numbers from 1-3 for each anaphor, the other used asterisks on the second page instead of arrows.



(a) Tree relation.

(b) Chain relation.

(c) Deictic relation.

(d) Relation "behind" text.

(e) Margin relation.

(f) Indices relation.

Figure 2.10.: Examples for different kinds of relations.

try to implement a similar mechanism, i.e. allow the user to create relations by means of drag-and-drop (most similar behavior to real world relational drawing) and visualize relations by means of lines or arrows (cf. Figure 2.10, variants "a", "b", "d" and "e").

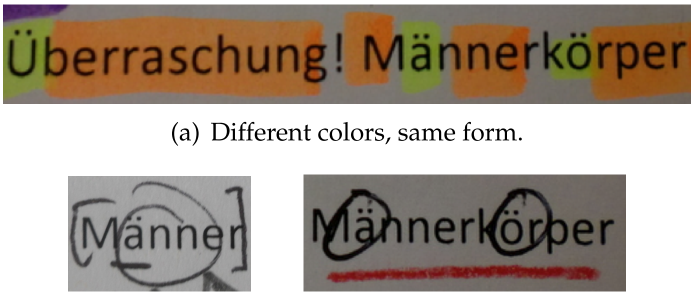
2.8.5. Findings on the creation of parallel annotation

One characteristic of linguistic annotation is the large number of parallel annotations, a phenomenon that typically occurs less frequently in active reading annotation contexts. Parallel annotation means that the same anchor (or parts of it) are annotated with different annotation contents. The same word might for instance be annotated on the word, phrase and sentence level, which makes for a total of three parallel annotations. In our test setting parallel annotations included double and triple annotations of one anchor. Most parallel annotations (14) were implemented by using a combination of different forms and colors (c). Six participants used the same form (underline , circle or highlight) in different colors (a). Only one participant, who throughout the whole test only used a black pen, implemented parallel annotation by mere variation of forms (b)[29]. Eight participants used only one of the three described ways ("b" or "c") to create a parallel annotation, the other seven varied between "a" and "c".

Synopsis

The key finding here is that parallel annotations are obviously best visualized by a combination of form and color. As the participants did not know how many or exactly which tasks they would have to perform during the test they could not make up a strategy for parallel annotations in the first place. That is why all parallel annotations were created on the fly, and build on previous annotations.

It must also be noted that paragraph annotations were not counted as parallel annotation here, because they are positioned mostly on the left margin or implemented with stand-alone start or end markers. Notably, the two participants, who annotated the clauses in the same fashion (i.e. with start and end markers) had more space for other parallel annotations on subordinated anchors within the clause scope.



(a) Different colors, same form.

(b) Same color, different forms.

(c) Combination of different forms and colors.

Figure 2.11.: Examples for different kinds of parallel annotations.

Implications for linguistic annotation:

- In the case study, parallel annotations are realized by means of different forms and colors, with color being the predominant means for distinguishing different, parallel annotations. Accordingly, digital annotation tools should distinguish parallel annotations primarily by means of color, but may also combine different forms for an enhanced distinction.
- The study showed that in many cases parallel annotation looked crammed, as different colors and forms were combined. The advantage of digital tools is that specific annotations can be shown or hidden, to avoid crammed, parallel visualizations. Such a show/hide function should be implemented for any annotation tool.

[29] It must be noted that this participant was very poorly motivated, and was overall not very successful in achieving the tasks. Hence he only drew very few parallel annotations.

2.9. Summary

This chapter has introduced existing models and theoretic approaches to the task of annotation. It has shown that most of the related work is dedicated to human handwritten annotations, both in digital and analog usage contexts. Although linguistic annotation is far more structured and formalized than handwritten annotation, there are many aspects that can be transferred to the domain of linguistic annotation. Various implications that can be derived from the existing handwritten annotation models have been discussed throughout this chapter whenever appropriate. These implications help to get a broad understanding of the basic elements, characteristics and functions of annotations. This domain knowledge will be useful for the assessment of typical tasks and requirements for annotation tools as well as for the identification and documentation of usability strengths and weaknesses (cf. chapter 4). Finally, it provides basic terminology and task models for the discussion of the created patterns (cf. chapter 5). The next chapter presents a detailed account of linguistic annotation, which is a recurring task in the domain of corpus linguistics.

3. Linguistic annotation

3.1. Introduction

While the previous chapter has introduced theoretical and formal aspects of annotations, this chapter deals with the practical application of annotation in the context of corpus linguistics. An introduction of the characteristics and most important features of linguistic annotation is essential to be able to evaluate and improve the usability of annotation tool interfaces.

Chapter structure

The chapter is structured as follows: Section 3.2 provides an overview of corpus linguistics as an empirical method and also gives a quick summary of the most important milestones in the history of corpus linguistics. In section 3.3, the technical foundations and standards for linguistic annotations are presented; section 3.4 gives a more detailed account of linguistic annotation, by defining and discussing its basic properties. Section 3.5 describes the larger context in which linguistic annotations typically occur and provides a typical workflow model for the creation and usage of linguistic corpora. Section 3.6 presents an overview of different forms of annotations (manual vs. automatic annotation) and underlines the main theme of this work, which is to provide user-friendly interfaces for manual annotation tasks. Finally, a brief summary of the whole chapter is presented in section 3.7, leading over to the next chapter on *usability* and *linguistic annotation tools*.

3.2. Corpus linguistics as an empirical method

Language corpora

Corpus linguists study language by collecting and observing evidence of actual language use (McEnery & Wilson, 1996, p. 1). This collection of evidence is called a *corpus*, which may consist of a multitude of different texts. Yet, there are some fundamental differences in how a corpus is used as compared to a single text (Tognini-Bonelli, 2001, p. 3): A text may be seen as a specific instance of language (cf. Saussure's concept of *parole*), while a corpus stands for a self-contained language system (cf. Saussure's concept of *langue*) that contains instances of language, i.e. texts or fragments of texts. If we take a corpus as a self-contained system of language, there are certain points that need to be considered when building it (Tognini-Bonelli, 2001, p. 54):

- *Authenticity* of the texts
- *Representativeness* of language
- *Sampling criteria* for the selection of texts

McEnery & Wilson (1996, p. 24) sum up the basic characteristics of a corpus, including its limitations, with the following words:

> A corpus in modern linguistics, in contrast to being simply any body of text, might more accurately be described as a finite-sized body of machine-readable text, sampled in order to be maximally representative of the language variety under consideration. However, the reader should be aware of the possibilities for deviation in certain instances from this 'prototypical' definition.

Empiricism vs. rationalism

In the philosophical debate between empiricists and rationalists, corpus linguistics takes on the role of an empirical method, which is opposed to theoretical linguists. McEnery & Wilson (1996, p. 5) describe the fundamental divide between theoretical linguistics (rationalists) and corpus linguistics (empiricists) as related to the "basic decision of whether to rely on naturally occurring observations or to rely on artificially induced observations". Another distinction has to be made between *corpus-based* and *corpus-driven* linguistics (cf. Tognini-Bonelli, 2001): Corpus-based studies use corpora to explore, validate and refine preexisting theories (corpus linguistics as a method), whereas corpus-driven studies use corpus data to generate new theories about language (corpus linguistics as a branch of linguistics, also cf. McEnery & Wilson, 1996, 5-6).

Digital corpus linguistics

Although there has been a lively debate between corpus linguists and theoretical linguists[30], with some arguments against corpus linguistic approaches (e.g. infiniteness of language, language competence vs. performance, etc.) being fairly reasonable, the method has gained a significant boost from the advent of high-capacity computers and the availability of large amounts of machine-readable texts. The improved efficiency of computer-driven corpus linguistics can be best illustrated by comparing a corpus from the late 19th century and a state of the art corpus created from digitized material: In 1891, the German stenographer and linguist Friedrich Wilhelm Kaeding started to create a corpus to get insights on the frequencies of phonemes, syllables and words in order to improve the existing stenographic system. It took him several years (1891-1897) and the joint efforts of hundreds of voluntary assistants[31] to create a corpus of approximately 11 million words and 20 million syllables (Kaeding, 1898). Almost a century later, in 1991, the *British National Corpus* (BNC) was gathered in a collaboration of publishers, universities and the *British Library*. Within four years a project team had created a 100-million-word corpus, which is freely available on the Internet[32]. Roberto Busa (2004, p. xvi ff.), one of the pioneers in the field of computer-based

[30] For an extensive review of the debate known as the "Linguistics Wars" cf. Harris (1993).

[31] Kaeding (1898, p. 22 ff.) describes the corpus creation as a process with altogether twelve stages, where the first stage alone required 665 assistants.

[32] `http://www.natcorp.ox.ac.uk/`

text analysis, subsumes the development of computer technology and storage media as *technological miniaturization*[33]. The 19th century example of Friedrich Wilhelm Kaeding not only shows that corpus creation without computers and digitized material is a laborious task, but also that a mere collection of words and their frequencies does not allow for deeper insights into human language, but only for a quantitative analysis. Nevertheless, corpora can be used to answer qualitative linguistic questions, too. The crucial requirement for corpora to be used in such a way, however, is the availability of *linguistic annotation*, or as Geoffrey Leech puts it:

Linguistic annotation

> To extract information from a corpus, we often have to begin by building information in – that is, by adding annotations (Leech, 1997, p. 4-5).

3.3. Standards of linguistic annotation

In order to improve the interface of linguistic annotation tools it is essential to have a basic understanding of the technical foundations of linguistic annotations. This section provides an overview of the evolution of markup languages and related techniques, but also describes more specific standards for the area of linguistic annotation.

3.3.1. The evolution of markup languages

Generic coding

In 1967, William Tunnicliffe first proposed the idea of *generic coding*, which means to separate a document's form from its content. A few years later, the book designer Stanley Rice introduced the idea of *editorial structure tags* to implement Tunnicliffe's concept of generic coding that would later be adopted in SGML and XML. By annotating documents in a generic way, a text's logical and structural features are described rather than its visual appearance for a specific scenario. The actual appearance of the text is handled by a styling language (e.g. *Cascading Style Sheets* (CSS)) that can refer to the logically and structurally annotated text elements. In 1969 Charles Goldfarb, Edward Mosher and Raymond Lorie designed the first markup language that incorporated the concept of generic coding: the *General Markup Language* (GML). The following years are characterized by an endeavor to further enhance the GML and to formulate a generic markup language in a standardized way. The final result was released by the *International Organization of Standardization* (ISO) in 1986: the *Standard Generalized Markup Language* (SMGL ISO 8879). The more recent history of markup languages began in 1989, when Tim Berners-Lee introduced the *World Wide Web* (WWW) as one of the most successful Internet applications of all times. In the

GML

SGML

[33] Busa himself experienced the impact of *technological miniaturization* while building his infamous *Index Thomisticus* for 30 years, starting in 1946 (cf. Busa, 1980).

HTML

same year, Berners-Lee also decided to create an SGML-based markup language to describe the structure and content of web documents. This language was named *Hypertext Markup Language* (HTML).[34]

DTD

At this point, it becomes necessary to explain the concept of *markup languages* in more detail: In actual fact, SGML is a *meta markup language* that provides concepts and formalisms to define concrete markup languages like the HTML by means of a *Document Type Definition* (DTD). A DTD can be seen as an abstract data model that may be compared to the *relational model* in the world of database design (Mehler & Lobin, 2004, p. 3). A DTD may contain information about the structure and content of an actual document of the defined document type. The HTML-DTD[35] for instance describes the document type *webpage* by defining the basic elements (and attributes) of which a webpage may consist (HTML tag set) as well as rules on the usage of the these tags. A concrete instance (e.g. an actual HTML-webpage) of such a generic document type may then be *validated* against the specified DTD, to automatically check if any elements were used in the document that have not been defined in the DTD before. At the same time, there are some general rules for any markup language that need not be specified explicitly in a DTD, but which are known as rules of *wellformedness*. A wellformed document for instance has exactly one root element, and makes sure that elements are nested properly within the document.

XML

The *Extensible Markup Language* (XML) was introduced in 1997, to overcome some of the problems of SGML that were caused by the language's complexity, and in turn prevented SGML from being used outside of universities and big companies. XML is a subset of the SGML that was radically cut down and only contains the most important markup concepts. Like SGML, XML is a meta markup language that can be used for the definition of concrete markup languages by using data models such as the DTD, or the more sophisticated XML Schema. Similar to DTDs, XML Schema can be used to define document types, but was designed to overcome some shortcomings of the DTD concept: DTDs are defined in a proprietary notation, whereas XML Schemas are defined in XML. XML Schema also provides more data types and more flexibility for defining a document's structure (cf. Lobin, 2004, p. 28). There are many examples for XML based markup languages, for instance for the document types *website* (XHTML), *spoken dialogue* (VoiceXML) or *2D vector graphic* (SVG). Many of the linguistic annotation standards are also defined by means of XML-DTDs or XML Schemas.

[34] For an extensive review of the history of markup languages cf. Mintert (1998). The markup synopsis in this work is also based on information from Mintert (1998).

[35] Cf. the Strict DTD for HTML 4.01, available at `http://www.w3.org/TR/html401/sgml/dtd.html`

3.3.2. Toward a science of annotation

Nancy Ide (2007) introduced the idea of a *science of annotation*, which focuses on technologies for the linguistic annotation of corpora, especially on the interoperability of annotations, emphasizing the importance of norms and standards in this field:

> A "science" of annotation has now evolved that reflects the collective experience within the community. This new science includes the study and development of precise criteria for corpus design, appropriate statistics for measuring inter-annotator agreement and confidence, and means to define a set of annotation categories that reflect an underlying linguistic theory. It is also concerned with the design of an architecture for annotated resources that supports interoperability, and its implementation in systems and frameworks that support the creation and exploration of annotations. (Ide, 2007, para. 5)

TEI

The TEI is one of the most prominent examples for standardized tag sets that can be used for the encoding of texts in a humanities context (TEI, n.d., para. 1). The first versions of the TEI tag sets had been formulated with SGML, the latest version TEI p5 was created using XML. TEI p5 comes with a collection of different tag sets that can be used for all kinds of literary and linguistic text annotation. It also provides guidelines and documentation for the proper application and combination of the respective tag sets. Among these tag sets are generic sets, like for instance the *TEI Header*, which can be used for any kind of text, and more specific tag sets for special requirements, such as *Verse*, *Performance Texts*, *Transcription of Speech*, etc. (TEI, 2014b). Of particular interest for the case of linguistic annotation are the tag sets *Language Corpora* and *Simple Analytic Mechanisms*.

CES

The *Corpus Encoding Standard* (CES)[36], which is part of the *Expert Advisory Group on Language Engineering Standards* (EAGLES)[37] (Ide, 1998, p. 2), is another important standard for corpus linguistics. CES is an adaption and extension of the TEI scheme with the goal to support standardized and reusable encoding of language corpora. More specifically "the CES limits the TEI scheme [which is quite bulky] to include only the sub-set of the TEI tag set relevant for corpus-based work" (Ide, 1998, p. 2). Similar to the TEI, the CES was first developed as an SGML application, and is now based on XML, which is why the more recent CES version is called XCES[38]. According to Ide, the CES applies to corpora that are defined as collections of linguistic data. These collections consist of primary data and the actual linguistic annotation. CES provides elements for encoding different regions within the primary data, ranging from *document-wide markup* to *gross structural markup* such as the annotation of chapters and paragraphs, and *markup for sub-paragraph structures*, including the annotation of sentences, quota-

[36] http://www.cs.vassar.edu/CES/CES1.html

[37] http://www.ilc.cnr.it/EAGLES96/home.html

[38] http://www.xces.org

tions, and words. CES also covers linguistic annotation such as morphosyntactic tagging (Ide, 1998, p. 4).

LAF Building on the "big ideas" which have been introduced by *de facto* standards like the TEI or the CES, Ide (2007, p. 2ff.) notes a second generation of more binding annotation standards, which are defined by the *International Organization for Standardization* (ISO). With six different working groups, the ISO technical committee on *Language Resource Management* (ISO TC37 SC4)[39] is an important organ that is dedicated to various aspects of the standardization of language resources (cf. Table 3.1):

Subcommittee / Working Group	**Title**
TC 37/SC 4/WG 1	Basic descriptors and mechanisms for language resources
TC 37/SC 4/WG 2	Semantic annotation
TC 37/SC 4/WG 3	Multilingual information
TC 37/SC 4/WG 4	Lexical resources
TC 37/SC 4/WG 5	Workflow of language resource management
TC 37/SC 4/WG 6	Linguistic annotation

Table 3.1.: Working groups of the ISO TC37 SC4 technical committee according to ISO TC 37/SC 4 (2001).

Among the published standards so far are frameworks on *morphosyntactic annotation* (MAF), *syntactic annotation* (SynAF), *semantic annotation* (SemAF), and many more. At the core of the TC37 SC4 work, however, is the *linguistic annotation framework* (LAF):

> LAF is intended to provide a standardized means to represent linguistic data and its annotations that is defined broadly enough to accommodate all types of linguistic annotations, and at the same time provide means to represent precise and potentially complex linguistic information. (Ide & Romary, 2004, p. 1)

This goal is achieved by means of a single high-level data model[40] and an XML serialization of that model, which serves as a pivot format that can be used to map annotations in different representation formats onto one another (ISO 24612, 2012, p. v-1). The model was designed on the basis of different annotation schemes of all types and provenance. It tries to alleviate the exchange and reuse of annotated resources across different platforms and formats, and it promotes the interoperability of such resources (Ide & Romary, 2004, p. 4). The LAF pivot format is realized by the XML-based *graph annotation format (GrAF)*, which represents the referential structure of the annotations by means of *node* and *edge*

[39] `http://www.iso.org/iso/home/standards_development/list_of_iso_technical_committees/iso_technical_committee.htm?commid=48104`

[40] For a more detailed account on the model and its architecture cf. Ide et al. (2003) and Ide & Romary (2004).

elements (Ide & Suderman, 2007, p. 3). A demonstration of the flexibility and applicability of GrAF can be found in Ide & Suderman (2009), where the authors successfully transform the proprietary annotation formats of two different annotation environments (GATE[41] and UIMA[42]) into the GrAF representation, thus enabling the exchange of annotations between both tools.

3.4. A detailed account of linguistic annotation

The implications of general annotation theory for the specific case of linguistic annotation have been discussed in the preceding chapter. This section provides a definition of linguistic annotation as well as an overview of the basic forms and characteristics of linguistic annotation. This overview will help to design user-friendly annotation tools by considering specific task contexts for linguistic annotation (e.g. POS annotation vs. syntactic annotation).

3.4.1. Defining linguistic annotation and related terms

Linguistic annotation shows all the characteristics that apply to annotations in general, i.e. it is some additional information added to a piece of text[43] with the purpose of making some implicit phenomena of the text explicitly available for automatic analysis. Leech (1997, p. 2) defines linguistic annotation[44] as "the practice of adding interpretative, linguistic information to an electronic corpus of spoken and/or written language data". Besides this rather informal definition, there is a more elaborate definition of linguistic annotation and some related key terms in the LAF. The terms and definitions used in the ISO standard resemble the basic elements of annotations that have been identified in the previous chapter. For means of standardization and consistency, some appropriate terms and definitions from the LAF will be used throughout this dissertation (cf. Table 3.2)[45].

LAF terminology

[41] *General Architecture for Text Engineering*, available at `https://gate.ac.uk/`

[42] *Unstructured Information Management Architecture*, available at `http://uima.apache.org/`

[43] This work focuses on the annotation of textual material – there are, however, other types of media that can be annotated linguistically, too (cf. section 3.4.2).

[44] In actual fact, Leech (1997, p. 2) is using the term *corpus annotation* – although it is possible to linguistically annotate single texts, the usual case is the annotation of text corpora. Therefore it seems legitimate to use the terms linguistic annotation and corpus annotation synonymously.

[45] Note: The LAF also defines the terms *graph*, *node*, *vertex* and *edge*, which refer to the technical realization of the pivot format by means of the GrAF (cf. section 3.3.2).

Term	Definition	Notes
Primary data	Electronic representation of language data	Typically, primary data objects are addressed by "locations" in an electronic file, for example, the span of characters comprising a sentence or word, or a point at which a given temporal event begins or ends (as in speech annotation). More complex data objects may consist of a list or set of contiguous or non-contiguous locations in primary data.
Annotate	Process of adding linguistic information to *primary data*	–
Annotation	Linguistic information added to *primary data*, independent of its representation	–
Representation	Format in which the *annotation* is rendered, independent of its content	–
Segmentation annotation	*Annotation* that delimits linguistic elements that appear in the *primary data*	These elements include (1) continuous segments (appearing contiguously in the primary data), (2) super- and subsegments, where groups of segments will comprise the parts of a larger segment (e.g. contiguous word segment typically comprise a sentence segment), (3) discontinuous segments (linking continuous segments), and (4) landmarks (e.g. timestamp) that note a point in the primary data. In current practice, segmental information may or may not appear in the document containing the primary data itself.
Linguistic annotation	*Annotation* that provides linguistic information about the segments in the *primary data*	The identification of a segment as a word, sentence, noun phrase, etc. also constitutes linguistic annotation. In current practice, when it is possible to do so, segmentation and identification of the linguistic role or properties of that segment are often combined (e.g. syntactic bracketing, or delimiting each word in the document with an XML element that identifies the segment as a word or sentence).

Table continues on next page.

Term	Definition	Notes
Stand-off annotation	*Annotation* layered over *primary data* and serialized in a document separate from that containing the primary data	Stand-off annotations refer to specific locations in the primary data, by addressing character offsets, elements, etc. to which the annotation applies. Multiple stand-off annotation documents for a given type of annotation can refer to the same primary document (e.g. two different part of speech annotations for a given text).
Annotation document	XML document containing *annotations*	–
Anchor	Fixed, immutable position in the *primary data* being *annotated*	The medium determines how an anchor is described. For example, text anchors may be character offsets, audio anchors may be time offsets, video anchors may be time offsets or frame indices, image anchors may be coordinates.
Region	Area in the *primary data* defined by a non-empty, ordered list of *anchors*	–
Original artefact	Artefact or *annotation* from which the *primary data* is derived	–

Table 3.2.: Terms, definitions and notes in the context of linguistic annotations as described in ISO 24612 (2012, p. 1-2).

Primary data is derived from an *original artefact* and *annotated* by adding *linguistic information*. *Annotation* comprises *segmentation annotation*, which delimits linguistic elements as segments, and *linguistic annotation*, which adds the actual linguistic information to the text. Segmentation annotation is realized by means of *anchors*, which basically are an index of each character of the text, and *regions* that are defined by using the anchor structure (start point anchor and end point anchor for each region). Ideally, the annotation is separated from the primary data as *stand-off annotation*, i.e. it is stored in a separate *annotation document*. The format in which the annotation is stored is called its *representation*, which in most cases will be an XML based markup language.

Annotation schemes

The term *annotation scheme*, which is not explicitly defined in the above ISO standard, describes the set of items (tags) that can be used as valid annotations during the annotation process. An annotation scheme may describe tags for different *levels* of annotation, i.e. parallel annotations for some regions in the primary data. The existence of multiple, parallel annotation levels for one text document was one of the main reasons for the advance of *stand-off annotation*. The concept of stand-off annotation (in contrast to so-called inline annotation)

Stand-off annotation

was first introduced by H. S. Thompson & McKelvie (1997), and has now become a de facto standard for storing annotations in recent tools for linguistic annotation. The basic idea of stand-off annotation is to use linking mechanisms such as XLink (DeRose et al., 2001), XPath (Clark & DeRose, 1999) and XPointer (DeRose et al., 2002) to establish semantic roles that are virtually added to the primary data, while the actual annotations are stored in one or more separated annotation documents (H. S. Thompson & McKelvie, 1997; Ide, 2000). Among the main advantages of stand-off annotation formats are that *read only* primary data can be annotated as well, and that the annotations may consist of multiple, overlapping levels of annotations (H. S. Thompson & McKelvie, 1997; Burghardt & Wolff, 2009a).

3.4.2. Different sources of primary data and annotation modalities

Until now, linguistic annotation has been exclusively discussed with regard to written language. Although the focus of this dissertation is on annotation tools for written text, it must be noted that there are other, non-textual primary sources that can be the target of linguistic annotation as well. These sources include media data, like for instance image, audio and video files, but also more complex objects like web sites. One major requirement for multimodal annotation tools[46] therefore is to provide a method for the *transcription* of non-textual data.

Transcription

As the ultimate goal of any annotation is to make implicit information explicitly available for automatic processing and analysis, corresponding annotation formats have to be represented in a machine-readable, textual format. This format is commonly based on SGML or XML, which implies that the primary data is also formalized as textual data in order to become the target of a linguistic annotation. The process of formalizing primary data in such a way is typically known as *transcription*. Transcription is a laborious, interpretative task, and oftentimes a first level of annotation. If we imagine a sentence of spoken, maybe dialect language, we have to decide how to write down the dialect vocabulary. We also would have to decide if and how we want to transcribe prosodic features, such as tone or pauses.

Modalities

The different kinds of primary sources can be further distinguished by the *modalities* they describe. Humans use different modalities to communicate with each other. Most modalities can be categorized as speech, either spoken or written, or body movements, which may include "gesture, gaze, facial expression, head movement, bodily posture, and object manipulation" (Dybkjaer & Bernsen, 2002, p. 1). If several modalities are combined (e.g. speech and facial expression) this

[46] EXMARaLDA (`http://www.exmaralda.org/`) can be seen as a prominent example for the transcription and annotation of spoken language, while ANVIL (`http://www.anvil-software.org/`) is a widely used tool for the annotation of video files.

is called *multimodality* (in contrast to *unimodality*). The *Natural Interaction and Multimodality* (NIMM) working group, which is part of the *International Standards Language for Language Engineering* (ISLE) project, has contributed to the area of multimodal annotation tools with extensive surveys on existing tools, standards and user needs (Dybkjaer, Berman, Kipp, et al., 2001) as well as on the requirements for a multimodal annotation tool (Dybkjaer, Berman, Bernsen, et al., 2001).

Besides the modality of written text, every other modality requires transcription in order to become the target of machine-readable, linguistic annotation. Because of the enormous heterogeneity of non-textual primary sources, and the individual transcription and annotation problems, the focus of this dissertation is on textual data.

3.4.3. A taxonomy of linguistic annotation

As a user-friendly annotation tool will have to support different types of annotation, it is important to know about the basic forms of linguistic annotation and their respective characteristics. A taxonomy of the most common linguistic annotation tasks will also be useful for the design of an adequate evaluation study of existing tools (cf. chapter 4).

Anchor scope taxonomy

Burnage & Dunlop (1992, p. 152-153) provide a first taxonomy by systematizing linguistic annotations by the scope of their anchors, which can reach from single character scopes (e.g. encoding of special characters as entities) to scopes that contain the whole text (e.g. author information):

- character level
- word level
- phrase level
- sentence level
- structural level
- text level

While this taxonomy is easy to comprehend, it addresses different kinds of annotation tasks on a rather generic level that nevertheless will have implications for the interaction design of linguistic annotation tools: It makes a huge difference if the user wants to select single characters, words or sentences as the regions that will be annotated.

Linguistic level taxonomy

Another way to describe linguistic annotations is by differentiating the levels of linguistic information that are to be encoded. From this perspective, annotations can basically be categorized according to the major fields of linguistic theory (cf. Table 3.3, based on Leech (1997, p. 12)[47]):

	Linguistic level	Typical annotation task
(1)	Orthography	Proper encoding of the primary text with all the special characters (Leech, 1997, p. 14) and setting of the anchors to define basic regions (tokens, sentences) (Wilcock, 2009, p. 19).
(2)	Phonology, phonetics	Segmental annotation of phonemes in transcribed speech (Leech, 1993, p. 2), which typically occurs in multimodal corpora.
(3)	Prosody	Annotation of features of spoken (yet transcribed) language, for instance intonation, pauses, rhythm, etc. (Leech, 1993, p. 2).
(4)	Morphology (grammatical tagging)	Grammatical tagging is mostly concerned with part of speech tagging. In terms of theoretical linguistics, this would correspond to the level of *morphology*. Grammatical tagging is oftentimes the basis for more complex annotations, for instance on the syntactic level.
(5)	Syntax	Marking and labeling of phrases and sentences; corpora annotated in this manner are also called *treebanks*. According to Leech, there is a "tendency to move away from the more detailed annotation schemes to more simplified schemes. . . . [which is] known as skeleton *parsing*" (Leech, 1993, p. 3).
(6)	Semantics	Semantic annotations that help to distinguish the lexicographic senses of ambiguous words by using predefined *sense inventories* as tag sets (*semantic classification*, *sense resolution*) (Palmer & Xue, 2010, p. 244-246); labeling of semantic roles and relations (within one sentence) results in so called *proposition banks*, the annotation of a corpus with temporal information is called a *TimeBank*; Palmer & Xue (2010, p. 246ff.) also mention opinion/sentiment tagging as an example for semantic annotation: "Annotation of opinions, evaluations, emotions, sentiments, and other private states in text" (Palmer & Xue, 2010, p. 240-241).
(7)	Discourse	Any annotation that goes beyond the sentence boundary may be labeled discourse annotation; typical example: resolution of anaphoric relations (Leech, 1993, p. 4).
(8)	Pragmatics, stylistics	Annotation of pragmatic features such as speech acts or stylistic features such as speech and thought presentation (Leech, 1997, p. 90ff.).

Table 3.3.: Levels of linguistic annotation according to Leech (1997, p. 12) and typical annotation tasks.

[47] As Leech's taxonomy is intrinsically motivated by the main branches of theoretical linguistics, variations can be found throughout literature; cf. for instance the taxonomies of Wilcock (2009, p. 19-20) and Palmer & Xue (2010).

Wilcock (2009, p. 19-20), however, notes that "there is only an approximate correspondence between the levels of the tasks performed in practical corpus annotation work and the levels of description in linguistic theory", i.e. not every annotation task can be precisely assigned to one of the above linguistic levels. This is best illustrated by so called lower level annotations like for instance *tokenization* or *sentence boundary detection*. These annotations may be part of the orthographic level, but at the same time they are a necessary means to be able to describe higher level annotations such as *syntactic annotation*. The same holds true for POS annotation: the syntactic structure of a sentence cannot be annotated without annotating the POS beforehand. Fortunately, most lower level annotation tasks can be automatized to a large degree. Nevertheless, some languages remain, like e.g. Old or Middle German, where it is not possible to do automatic POS tagging for reasons of very heterogeneous orthography (Pilz et al., 2008, p. 66-67).

For the evaluation design of existing annotation tools, both, the *anchor scope* as well as the *linguistic level* taxonomies described in this section will be considered. I will design test tasks that cover different region scopes and the most frequent linguistic annotation tasks (cf. chapter 4.5).

3.5. Linguistic annotation in the corpus creation and usage workflow

Linguistic annotation is not an isolated activity, but rather one that is accompanied by other tasks that are necessary to create and use a corpus. Figure 3.1 shows the typical stages which are necessary during the creation and the actual usage of a text corpus.

Digitization

First, if the primary data is not in machine-readable format, it needs to be digitized. This can either mean the transcription of spoken texts into a digital representation, or the transformation of the pages of a book to a digital, machine-readable document. Much of the digitization of printed books can be automatized by using advanced *optical character recognition* algorithms. Yet, in many cases the digitized data is not ready to be used for annotation and querying.

Normalization

In the second stage, the digitized primary data needs to be normalized: scanning-errors have to be corrected, and the encoding, which is important for special characters (consider e.g. a set of runes for Old English texts), has to be set properly, preferably in *Unicode*[48] format. After these preliminary steps, we have set up a machine-readable, normalized corpus that is still lacking additional linguistic information which is important for later analysis.

[48] "Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language." (Unicode Consortium, 2013, para. 3)

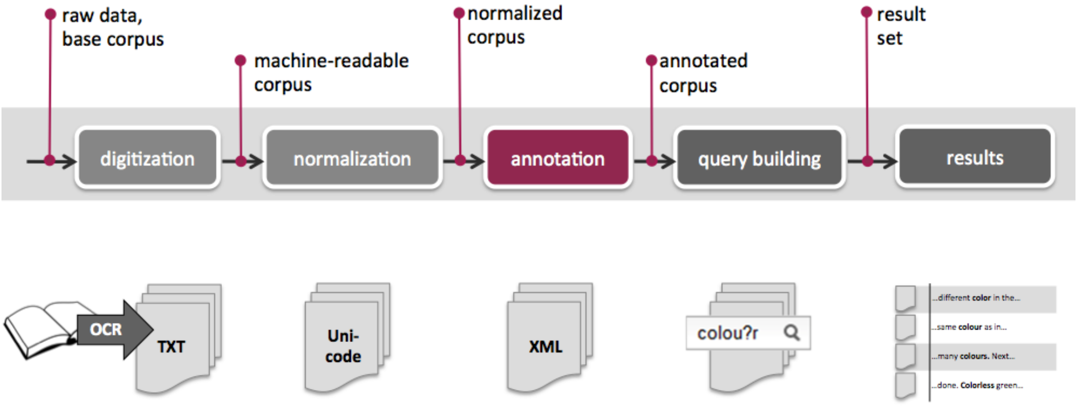


Figure 3.1.: Typical stages during the creation and usage of a corpus.

Annotation

Linguistic annotation, which includes the creation of a suitable annotation scheme (tag set), is probably the most laborious task. It can, however, be automatized to a certain degree (cf. section 3.6), and also be facilitated by user-friendly annotation tools. The actual annotation process and the interactions that are necessary to create an annotation scheme and to annotate a document on multiple levels of linguistic information are crucial for this dissertation. Therefore, a detailed account of the annotation process will be given in the the next chapter.

Query building

Up to here, the described stages were mainly concerned with the creation of the corpus. Once the annotated corpus is available, it can be used to answer linguistic questions. Exploring the corpus, however, is not a trivial task, as it requires a mechanism to query the data that has been annotated in a way that it delivers answers to previously formulated questions. It is important to note that query building depends on the available annotations, i.e. during the annotation process, the annotator has to keep in mind the questions he wants to ask the corpus later on, and make sure that all the necessary information is explicitly encoded. Soehn et al. (2008) provide a basic taxonomy for different types of corpus query interfaces, including prominent examples such as *COSMAS II*[49] and *TIGERSearch*[50]. Query interfaces typically implement logical search operators as well as *regular expressions*, which can be used to define patterns that match/select specific text strings in the corpus. Although query interfaces are typically realized as free text forms that act like a command shell, there have been attempts to provide *graphical user interfaces* (GUIs) for the query building process. An early attempt to realize such a GUI for queries can be seen in the *Query by Example (QBE)* language for querying relational databases, as suggested by Zloof

[49] *Corpus Search, Management and Analysis System* (COSMAS II), maintained by the *Institut für Deutsche Sprache* (IDS) in Mannheim (cf. `www.ids-mannheim.de/cosmas2/`).

[50] *Tools for Linguistic Text Exploration* (TIGERSearch), maintained by the *Institut für Maschinelle Sprachverarbeitung* (IMS) in Stuttgart (cf. `http://www.ims.uni-stuttgart.de/forschung/ressourcen/werkzeuge/tigersearch.html`).

(1977). Soehn et al. (2008, p. 4-5) also report on a prototypical corpus interface that is realized as a *graphical tree fragment query editor*. The interface allows the user to submit complex queries via drag-and-drop of the query components into the tree structure. COSMAS II is another example for a graphical user interface, where query components can be visually arranged and combined in a nested input field (cf. Figure 3.2).

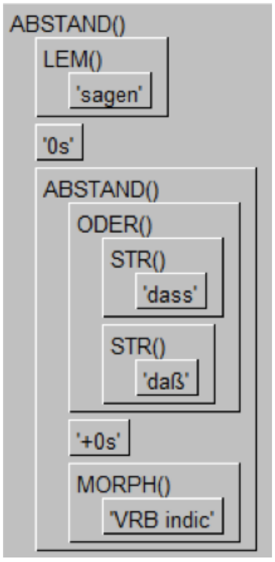


Figure 3.2.: COSMAS II graphical query builder, taken from the COSMAS II tutorial by Bopp (2010, p. 9).

Analyze results

The last stage in corpus creation and usage workflow comes right after a query has been submitted: the visualization and analysis of results. First suggested by Luhn (1960), the *Keyword in Context (KWIC)* has quickly become the standard for displaying corpus results. The KWIC visualization lists all keywords that match the corpus query as an alphabetical concordance list, but also provides the left and right neighbors (context) of the keyword. There are, however, alternative attempts to visualize corpus results, for instance the *Word Tree* (Wattenberg & Viégas, 2008), which is a treelike, graphical variant of the traditional KWIC, or the *Double Tree* (Culy & Lyding, 2010), which provides left and right contextual information of a keyword as two separate trees. Other forms of visualization include the well-known *word clouds* (cf. e.g. Wordle.net), which are used to indicate the frequency of certain words as compared to other words in the same collection. While current text analysis tools such as *Voyant Tools*[51] still use word clouds as a means of data visualization, we can observe a growing weariness of word clouds, as several studies have shown that they are outperformed by other forms of visualization, such as one-dimensional lists[52].

[51] `http://voyant-tools.org/`

[52] For an overview of related work on word cloud evaluations cf. Lohmann et al. (2009).

Accordingly, Voyant also offers a multitude of other experimental visualization techniques that can be subsumed as instances of *rich-prospect browsing interfaces*. Rich-prospect browsers display every item (e.g. words or images) of a collection on one overview screen, and allow the user to manipulate and explore any item in more detail (Ruecker et al., 2011, p. 3). The basic idea of *rich-prospect* is to provide "the user with a visual basis for understanding what is available in a collection" (Ruecker et al., 2011, p. 4) – no matter how big this collection is.

Along with the analysis of data within the query interface, the final step in the corpus workflow is to export the relevant results (Soehn et al., 2008, p. 2) in a way they can be used for further studies or for publication (preferably in an XML based format).

3.6. Automatic vs. manual annotation

Taxonomy

McEnery & Hardie (2012, p. 30) identify three basic approaches for linguistic annotation: automatic annotation, semi-automatic annotation (automatic annotation with manual correction) and manual annotation. This basic taxonomy of approaches to annotation is also backed up by a topic analysis of the proceedings of the *Linguistic Annotation Workshop (LAW)*. The LAW is held annually since 2007, and is part of the *ACL Special Interest Group for Annotation* (SIGANN[53]). I tagged all articles from 2007 - 2011[54] by their title and abstract. 41 unique tags (for instance *manual annotation*, *automatic annotation*, etc.) were used to describe 140 articles, which made for a total of 228 tags. Figure 3.3 shows the most frequent topics from this study, including important topics like for instance *corpus annotation* (40 articles), *annotation schemes* (24 articles), *annotation practices* (22 articles), *treebanks* (21 articles), *evaluations* (16 articles) and *annotation formats* (12 articles). The frequencies for articles that deal with *annotation tools* (11 articles), *automatic annotation* (11 articles), *semi-automatic annotation* (9 articles) and *manual annotation* (6 articles) seem to emphasize McEnery's categorization of approaches to annotation. Interestingly, automatic and semi-automatic annotation appear more often than manual annotation, indicating that manual annotation per se is very costly and laborious, but that automatic annotation is also combined with manual approaches (semi-automatic annotation) in many cases.

The rest of this section discusses the chances and limitations of manual and automatic annotation approaches, and argues for a combined, computer-aided manual annotation process.

[53] `http://www.cs.vassar.edu/sigann/`

[54] The full proceedings are available at `http://www.cs.vassar.edu/sigann/previous_workshops.html`.

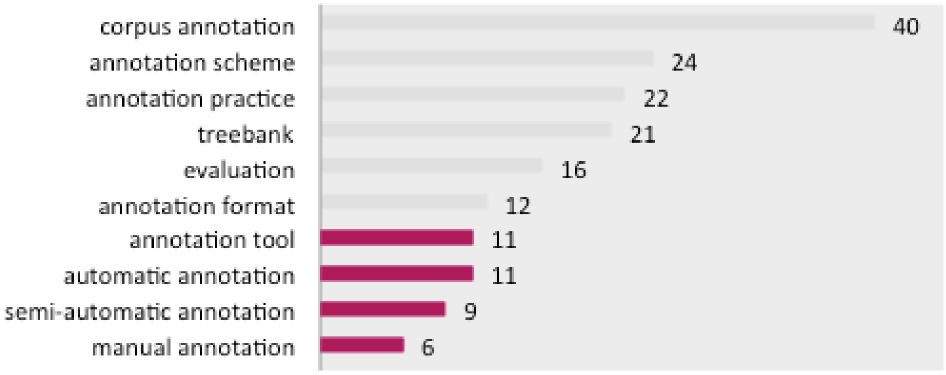


Figure 3.3.: Analysis of the most frequent topics of the LAW proceedings from 2007 – 2011.

3.6.1. Automatic annotation

POS tagging

Computers perform well in automatic annotation when the area of knowledge is self-contained and when the linguistic or semantic knowledge in question can be translated into a lexicon or into algorithmic rules. Typically, automatic annotation approaches are successful for POS annotation, i.e. the annotation of word form and lemma information. Oftentimes POS annotation is synonymously called POS tagging, thus automatic annotation tools are often called taggers. According to Meyer (2002, p. 88ff.) and Leech et al. (1994, 102ff.), there are two main types of taggers[55]: *rule-based* taggers and *probabilistic* or *stochastic* taggers.

Rule-based taggers

Rule-based taggers basically look up the words they are going to annotate in a lexicon, where one or more values (tags) are stored for each word (Meyer, 2002, p. 88). As many words are ambiguous by their mere orthographic form (example: "ship" may be a noun as well as verb), contextual information has to be considered by means of rules (also called templates). Some exemplary templates from the renowned Brill Tagger illustrate the form and function of such rules (Brill, 1992, p. 2)[56]:

Rule 1 If a word is tagged *a* and it is in context *C*, then change that tag to *b*, or

Rule 2 If a word is tagged *a* and it has lexical property *P*, then change that tag to *b*, or

[55] Merialdo (1994, p. 1) and Schmid (1994b, p. 1) describe the same two main types, but also mention neural network-based approaches as a third category. This survey, however, focuses on rule-based and probabilistic taggers, as most available taggers belong to one of those two categories. For more information on taggers that rely on neural networks cf. Benello et al. (1989), Nakamura & Shikano (1989), and Schmid (1994a).

[56] Other rule-based taggers like e.g. *TAGGIT*, employ so called context-frame rules, which are based on linguistic observation of language data to disambiguate a word by its context (Leech et al., 1994, p. 103).

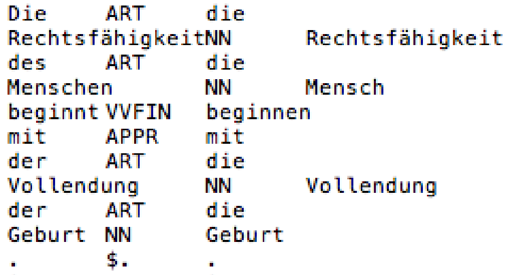
Rule 3 If a word is tagged *a* and a word in region *R* has lexical property *P*, then change that tag to *b*.

Probabilistic taggers

Most rule-based taggers have been superseded by probabilistic taggers, which generally have substantially lower error rates (Brill, 1992, p. 1). The basic idea of probabilistic taggers is to "assign tags based on the statistical likelihood that a given tag will occur in a given context" (Meyer, 2002, p. 88). Probabilistic taggers typically use *Hidden Markov Models (HMM)*[57] (cf. Poritz, 1988) to capture this contextual information (Brill, 1992, p. 1). As one word can have several tags, the HMM calculates the probability of a sequence of words (context) rather than tagging the words isolated from each other:

> If we have a sequence of words, each with one or more potential tags, then we can choose the most likely sequence of tags by calculating the probability of all possible sequences of tags, and then choosing the sequence with the highest probability. (Leech et al., 1994, p. 103)

In contrast to rule-based approaches, probabilistic taggers need to be trained so they can compare the sequences of untagged words to a tagged training corpus. However, not all probabilistic taggers rely on HMMs. The *TreeTagger* by Schmid (1994b) for instance makes use of binary decision trees that use (similar to the rule-based taggers) a lexicon that contains the a priori tag probabilities. The TreeTagger's output, the original word, the POS as well as the lemma, are depicted in Figure 3.4. The TreeTagger can be seen as an example for a probabilistic tagger that makes use of a typical rule-based feature, a lexicon. There seems to be a trend toward hybrid taggers[58], where the strategies described above are combined:



```
Die     ART     die
RechtsfähigkeitNN       Rechtsfähigkeit
des     ART     die
Menschen        NN      Mensch
beginnt VVFIN   beginnen
mit     APPR    mit
der     ART     die
Vollendung      NN      Vollendung
der     ART     die
Geburt  NN      Geburt
.       $.      .
```

Figure 3.4.: Example output of the TreeTagger. Input text taken from the *German Civil Law Code* (BGB – Bürgerliches Gesetzbuch).

Hybrid taggers

> Nearly all probabilistic taggers have sets of heuristic rules or guessers dealing with unknown words, while some rule-based systems use a limited amount of frequency information. (Garside & Smith, 1997, p. 106-107)

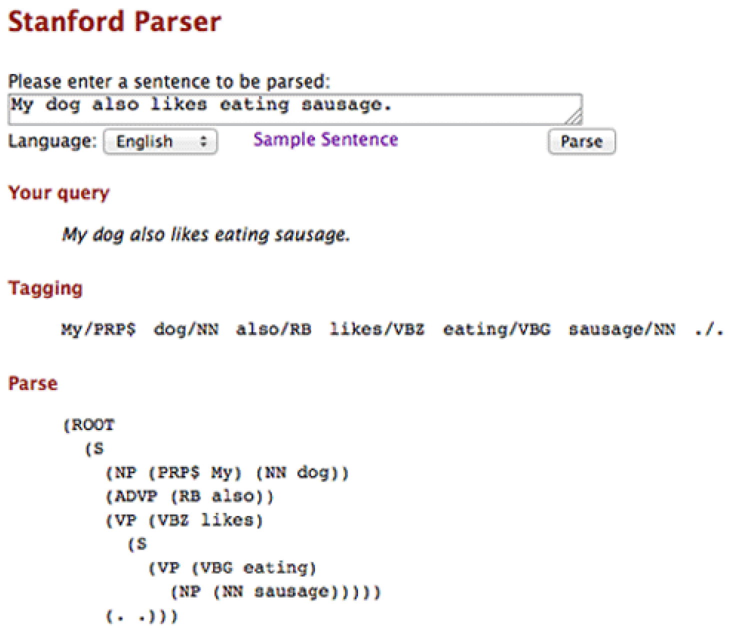
[57] Cf. the *Trigrams'n'Tags* (TnT) tagger by T. Brants (2000b), which is a prominent example for a probabilistic tagger that makes use of second order Markov models.

[58] The *Constituent Likelihood Automatic Word-tagging System* (CLAWS) is an example for such a hybrid grammatical tagger (Leech et al., 1994).

Parsing

Treebanks

While automatic POS taggers attain accuracy rates from 95-98% (Leech et al., 1994, p. 103; Garside & Smith, 1997, p. 119), automatic tools that try to annotate syntactic phenomena score much lower (Meyer, 2002, p. 91). The identification of the constituents of a sentence and the annotation of their syntactic function in the sentence is called parsing. Much like taggers, most parsers are either rule-based or probabilistic, and rely on specific grammatical models which are used as "parsing schemes" (Meyer, 2002, p. 91). Parsers typically also make use of POS annotations, which are necessary to identify syntactic chunks and their roles (cf. Figure 3.5). Syntactically parsed corpora are called *treebanks*; prominent examples for such treebanks include the German language *TIGER Treebank* (S. Brants et al., 2002) and the English language *PENN Treebank* (Marcus et al., 1993).



Stanford Parser

Please enter a sentence to be parsed:

My dog also likes eating sausage.

Language: English ⇕ Sample Sentence Parse

Your query

My dog also likes eating sausage.

Tagging

```
My/PRP$  dog/NN  also/RB  likes/VBZ  eating/VBG  sausage/NN  ./.
```

Parse

```
(ROOT
  (S
    (NP (PRP$ My) (NN dog))
    (ADVP (RB also))
    (VP (VBZ likes)
      (S
        (VP (VBG eating)
          (NP (NN sausage)))))
    (. .)))
```

Figure 3.5.: Example for combined POS tagging and syntactic chunking from the online demo of the *Stanford Parser* (available at: `http://nlp.stanford.edu:8080/parser/index.jsp`).

The discussion so far reveals that automatic annotation is limited to fields of manageable degrees of complexity (hence it is also called *shallow* annotation), including simple text processing tasks such as tokenization and sentence segmentation, or simple tagging and parsing tasks such as POS tagging or syntactic phrase detection/categorization (T. Brants & Plaehn, 2000, p. 1). At the same time it becomes evident that more sophisticated types of annotation cannot be automated, but rather have to be carried out by a human annotator (cf. T. Brants & Plaehn, 2000; Dandapat et al., 2009).

3.6.2. Manual annotation

Man vs. machine

While computers lack the ability to interpret contextual information in the correct semantic and pragmatic way, they have strengths in combining and calculating explicitly available information, i.e. a computer can dig through a large collection of documents and return only those that contain a specific syntactic constellation within a certain genre of text. However, when it comes to a deep interpretation of words, phrases and sentences by means of semantics and pragmatics, human knowledge and intelligence are needed as input: Scattered anaphora, misspelled words, witty puns and ambiguous meanings – humans are able to activate their mental lexicon (cf. Aitchison, 1994) in the wink of an eye to make sense of even the most obfuscated linguistic constructions, and to interpret synonyms, super- and sub-categories, named entities, social roles, anaphora, etc. Although many annotation tasks can be automated quite well, it is undisputed that a certain degree of manual annotation is necessary in any case. McEnery & Hardie (2012, p. 30) note that "it is not currently possible to reliably undertake automated corpus annotation for all types of linguistic analysis", i.e. there is always a need for manual annotation. As many automatic approaches require correctly annotated training data[59] in the first place, another case for human annotation is made:

> It is tempting to assume that recent advances in semi-supervised and unsupervised machine learning . . . may eventually obviate the need for linguistic annotation, but this is not likely. Even unsupervised systems rely on manually annotated data for evaluation purposes. (Palmer & Xue, 2010, p. 239)

Manually annotated corpora may not only serve as training data for machine learning approaches, but also as *gold standard* that can be used as a benchmark for testing the quality of automatic annotation tools (McEnery & Hardie, 2012, p. 31). A prominent example for a corpus with manual annotations is the *Manually Annotated Sub-Corpus* (MASC, Ide et al., 2008; Ide & Fellbaum, 2010)[60], which is part of the *American National Corpus*[61]. MASC can be seen as "the first large-scale, open, community-based effort" (Ide & Fellbaum, 2010, p. 2) to create a corpus that is annotated for multiple linguistic phenomena.

Consistency

At the same time it is important to note that there is no guarantee that manual annotations are 100% correct (McEnery & Hardie, 2012, p. 31), especially if several annotators are involved. While humans may have strengths in identifying complex linguistic phenomena a machine could not have known, they are more prone to make easy mistakes over the course of time. Human annotators may

[59] To reduce the effort for manually created training data, *active learning* has been adopted to the area of corpus annotation (Ringger et al., 2007; Song & Yao, 2010). Active learning methods typically formulate queries, to select only the most informative examples from the training data (C. Thompson et al., 1999, p. 1). The human annotator plays the role of an *oracle*, only annotating the examples that are requested by the machine (Settles, 2010, p. 3ff.).

[60] `http://www.anc.org/data/masc/`

[61] `http://www.anc.org/`

oversee some linguistic phenomena or annotate them inconsistently. The aspect of consistency of annotation becomes even more problematic when multiple annotators are involved in a corpus project: *Inter-annotator consistency* is concerned with how different humans annotate a specific sentence, while *intra-annotator consistency* describes if one annotator is consistent throughout the annotation process (T. Brants, 2000a, p. 1).

Besides consistency issues, another drawback of manual annotation in contrast to automatic approaches is its cost and effort (T. Brants et al., 1999; Ide et al., 2008; Santos & Frankenberg-Garcia, 2007). With manual annotation being such a laborious and time-consuming task, it is typically combined with automatic approaches. Eryigit (2007, p. 117) distinguishes a manual *from-scratch* procedure from manual controlling and correcting of automatic annotations. Marcus et al. (1993, p. 318-320) describe an experiment where these two modes of annotation are evaluated: Annotators did POS tagging for blank, unannotated text in the first mode. In the second mode they verified and corrected automatically generated output from a POS tagger (PARTS algorithm). The POS tagging from scratch took twice as long, annotators disagreed twice as much, and it produced 50% more errors. This, at least for the area of POS tagging, clearly speaks for an automatic approach with manual correction.

As was described earlier, many annotation tasks such as POS tagging or shallow syntactic parsing can be automatized to a certain degree and may be efficiently combined with a manual correction approach. Nevertheless, there are still areas of more complex or highly specific linguistic annotation tasks, where manual annotation will have to be applied from scratch (Castilho et al., 2007, p. 1). While most corpus projects annotate texts on various, parallel levels, a combination of manual and automatic annotation approaches may often be the case. Castilho et al. (2007, p. 1) suggest a basic 4-step workflow for the integration of manual from-scratch annotation with automatic annotations:

Step 1 Basic automatic analysis (tokenization, etc.)

Step 2 Select candidate units for further manual annotation by means of query

Step 3 Extract the selected candidates from different source documents and aggregate them in a single document

Step 4 Merge manual annotation back to original corpus

Summing up the opportunities and limitations of automatic and manual annotation, a linguistic annotation tool should provide the following main features:

- Support automatic annotation for simple tasks
- Provide a correction-mode for automatically generated annotations
- Provide a from-scratch manual annotation mode
- Allow the user to integrate automatic annotation with manual annotation

One main challenge for the interface design will be the integration of different annotation modi.

3.6.3. Toward user-friendly interfaces for manual annotation

Man vs. machine

By accepting that computers have their very own strengths as compared to human annotators (and vice versa), a combined approach that integrates the best of both worlds seems most promising. Computers should do the programmable routine annotation work, while humans annotate the more complicated and complex phenomena of language and its context. Computers should also cover an adequate visual representation of data and provide efficient editing mechanisms (Dandapat et al., 2009, p. 1). This view enforces the tool character of computer programs, with the main requirement not being artificial intelligence, but a high level of usability. In fact, there seems to be a trend in the field of artificial intelligence which points toward an integrated approach of human intelligence and computers as a supportive tool, accounting for the specific strengths and weaknesses of both, man and machine. The web service *Amazon mechanical turk*[62] implements this idea in a crowdsourcing[63] context, allowing for the use of human intelligence in order to perform on-demand sub tasks (Lenk et al., 2009, p. 26) which computers are not able to cope with. In many cases these *human intelligence tasks* (HITs) are annotation tasks on the semantic level:

Example 1 Categorize the sentiment of a sentence

Example 2 Choose the best category for this product

Artificial artificial intelligence

In analogy to Wolfgang von Kempelen's legendary automaton chess player ("The Turk"), which actually was a hoax, concealing a human chess master inside a seemingly chess-playing automaton, Amazon puts human intelligence inside simple computer programs as a problem-solving component, thus creating *artificial artificial intelligence* (Pontin, 2007, para. 3). Mechanical turk illustrates the idea of joint intelligence of humans and computers, where computers do the routine work and humans merely use the computer as a tool to perform different kinds of annotation tasks.

Usability matters

Along these lines, linguistic annotation tools should try to make manual annotation as convenient and efficient as possible. This goal can only be achieved by providing appropriate functionality as well as a high level of usability for the annotation interfaces. Unfortunately, there seems to be a huge gap between the desired usability of linguistic annotation tools and the degree of usability most existing tools can offer: Throughout literature we find hints on the importance of *easy-to-use*, *simple* and *intuitive* interfaces (Dybkjaer, Berman, Bernsen,

[62] `https://www.mturk.com`

[63] The large-scale aggregation of information from crowds of people is often referred to as crowdsourcing. In the cloud computing context, where *Everything is a Service* (Xaas), crowdsourcing is similar to the *Humans as a Service* (HuaaS) concept (cf. Lenk et al., 2009).

et al., 2001; Dipper et al., 2004; Reidsma et al., 2004; Eryigit, 2007; Dandapat et al., 2009; Palmer & Xue, 2010; Hinze et al., 2012), but evaluations (Burghardt & Wolff, 2009b; Burghardt, 2012) of existing tools indicate that a decent degree of usability is largely missing[64]. McEnery & Hardie (2012, p. 33) come to a similar conclusion:

> Corpus search tools have over the years developed to become very user-friendly. By contrast, corpus annotation programs – while widely available – typically require so much advanced computer expertise to install and use that they are, effectively, not accessible to most linguists.

Usability as a cost saver

The demand for usability of annotation tools is also strongly motivated by the cost aspect of manual annotation. In her seminal article on "Functionality and Usability", Goodwin (1987, p. 231) underlines the importance of usability with regard to cost savings:

> For computer systems used for large-scale transactions, seemingly small improvements in usability can translate into large cost savings: Saving as little as 1 second per transaction can mean a savings of thousands of dollars as well as significantly improved productivity.

This statement can be easily transferred to the area of linguistic annotation, where user-friendly interfaces can speed up the manual annotation process significantly (Eryigit, 2007, p. 1). Dandapat et al. (2009, p. 1) note that the cost of an annotation project can be measured by the number of man-hours and the level of expertise that was required. They suggest several techniques that can boost the benefit-cost ratio of annotation tasks: These include automatic approaches that rely on supervised learning algorithms or an active learning procedure, crowdsourced annotation approaches, and "smartly designed user interfaces for aiding the annotators" (Dandapat et al., 2009, p. 1).

Toward ergonomic annotation tools

Although this section has shown that automatic annotation is an important means to automatize certain types of annotation tasks, it has also become evident that manual annotation is still needed for other, more complex annotation tasks, or that it is combined with automatic approaches, mainly to check and correct machine annotations. As the point for manual annotation has clearly been made, this work will not try to enhance automatic annotation algorithms, but rather aims to provide suggestions for the design of user-friendly annotation tools that, along with existing automatic annotation functionality, support the manual annotation process of complex linguistic phenomena with an intuitive and user-friendly interface.

[64] Also cf. the tutorial on "Understanding and Improving Annotation Usability" in a biomedical informatics context, which was presented by Harry Hochheiser at the iDash NLP Annotation Workshop, September 29, 2012. A video recording of the tutorial is available at `http://www.youtube.com/watch?v=joYVHjB5tgs`.

3.7. Summary

As a quantitative branch of linguistics, corpus linguistics strongly benefits from the availability of large amounts of machine-readable data that can be compiled into language corpora. This chapter has shown that linguistic annotation – besides other activities such as digitization, normalization, etc. – is an essential task during the creation of these corpora. Accordingly, there is an abundance of markup formats and standards for the field of linguistic annotation. This chapter also has introduced different approaches to linguistic annotation: while automatic annotation obviously is less laborious than manual annotation, it cannot be applied to the full range of linguistic and language-specific problems. At the same time, manual annotation is a cumbersome task, as the interfaces of corresponding annotation tools in most cases suffer from severe usability problems that impede a smooth annotation experience. This leads to the conclusion that it is important to improve the usability of linguistic annotation tools. The next chapter illustrates how usability for annotation tools can be measured and improved systematically, as it describes the basics of usability engineering, and also presents an evaluation study for linguistic annotation tools.

4. Usability and the case for annotation tools

4.1. Introduction

This chapter introduces the concept of *usability*, and also shows how it can be defined, tested and engineered systematically. Moreover, typical users, requirements and tasks from the domain of linguistic annotation tools will be introduced as the basis for a large-scale usability evaluation of linguistic annotation tools. The details of the study design as well as the quantitative results are described toward the end of this chapter.

Chapter structure

The chapter is structured as follows: Section 4.2 introduces important concepts from the field of usability engineering; section 4.3 gives an overview of different usability evaluation methods. In section 4.4 the characteristics of the application domain are presented from an HCI perspective. These characteristics include different user groups for annotation tools, requirements for linguistic annotation tools and typical micro-tasks that will occur during the annotation process. Finally, this section provides a categorization scheme for linguistic annotation tools and related tools and resources. Section 4.5 is dedicated to the usability evaluation of existing annotation tools. First, the choice of evaluation method is elucidated. Next, the chosen method is described in more detail. A previous pilot study that has used the same method for a smaller set of annotation tools is recapitulated. As part of the overall evaluation design, this section also clarifies the generation of tasks as well as the selection of annotation tools as appropriate test objects. Finally, the actual evaluation procedure is documented. In section 4.6, the quantitative results of the evaluation study are presented. At the same time it is argued that usability patterns are an appropriate means to document the qualitative results of the study. Section 4.7 presents a brief summary of the whole chapter and leads over to the next chapter on *usability patterns*.

4.2. Fundamentals of usability engineering

Whenever we have a feeling that a product is easy – and maybe even fun – to use, this is due to its inherent *usability*[65]. It may, however, be hard to articulate the reasons why we have these positive feelings about a specific product, as usability typically is a factor that is not clearly visible on the surface of the product (Barnum, 2011, p. 1), but rather is the result of many smaller factors.

Components of usability

Accordingly, Nielsen (1993, p. 26) suggests that usability must not be treated as an abstract, one-dimensional concept, but rather should be broken down into multiple, more concrete components such as *learnability*, *efficiency*, *memorability*, *error rate* and *satisfaction*. Each of these components can be tested and engineered individually, which makes usability as a whole much more graspable. These components are typically influenced by different aspects, such as human performance, learning, cognition, and collaboration (Rosson & Carroll, 2002, p. 9).

ISO 9241-11

A rather formal definition of usability is provided in the ISO 9241-11 (1999) standard[66]:

> The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.

This definition includes three critical measures of usability (Barnum, 2011, p. 11) – *effectiveness*, *efficiency*, and *satisfaction* – which are the subject of usability testing. The multiple use of the word *specified* in this definition indicates that there is no single, common formula for systems with good usability, but that usability rather depends on the specific circumstances of a project.

Guidelines, heuristics, etc.

While these definitions describe usability on a rather abstract level, it is also possible to describe it on a more practical level, by giving concrete advice for the implementation of user-friendly interfaces. Typical means to document such practical advice for the realization of usability include *guidelines*, *heuristics*, *rules*, *principles*, *recommendations*, *best practices*, etc.

Layered model of usability

While the basic idea of all those different definitions (on different levels of detail) is to make the concept of usability more comprehensible and more graspable, it can be challenging to paint the big picture of *usability* and to relate the different definitions to each other. Therefore, Van Welie et al. (1999) suggest a comprehensive usability framework that can be used to integrate different definitions (with different levels of detail). Van Welie et al. (1999, p. 4-5) break down theoretical and practical concepts and definitions of usability and relate them to each other in a *layered model of usability* (cf. Figure 4.1). On the highest, most abstract level (no immediate practical applicability) are the ISO 9241-11 criteria

[65] More recently, the term *user experience* (UX) has been established to express a more holistic view of relevant *usability factors*, including aspects such as *satisfaction*, *hedonic qualities* and *joy of use* (also cf. Hassenzahl et al., 2001); cf. Bevan (2009) for a comprehensive overview and a discussion of similarities and differences between the concepts of *usability* and *user experience*.

[66] As I do not have access to the original ISO standard in English language, I refer to the definition as provided by Barnum (2011, p. 11).

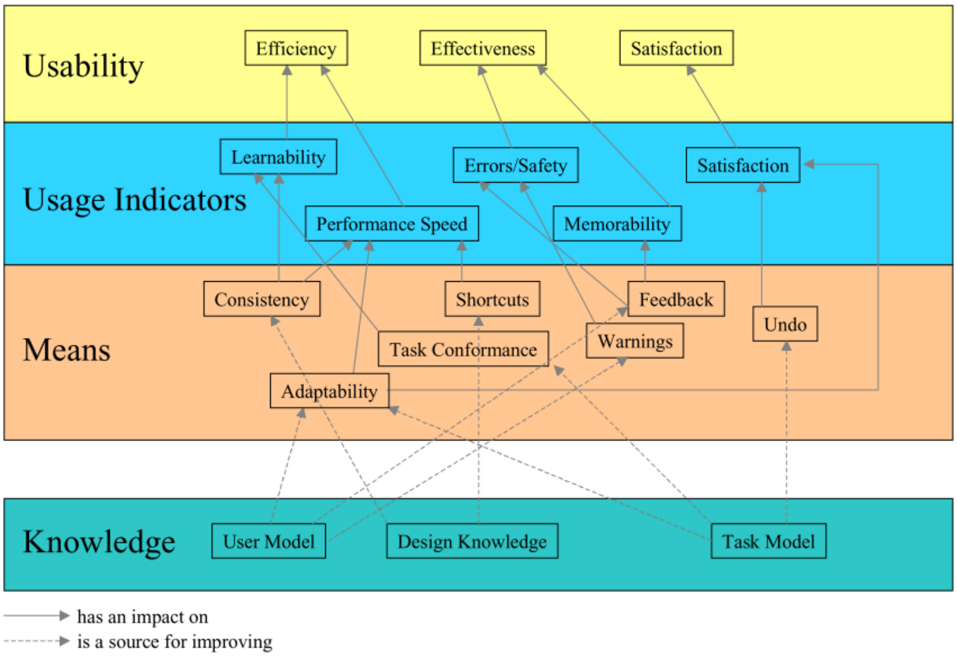


Figure 4.1.: Layered model of usability (image source: Van Welie et al., 1999, p. 5, Figure 1).

for *usability*: *effectiveness* describes the accuracy and completeness of an achieved goal), *efficiency* relates the expenditures to the level of effectiveness, and *satisfaction* describes the comfort and acceptability of use (Bevan, 1999). On the next level we find so called *usage indicators* that can actually be observed and tested in practice. Table 4.1 shows that these usage indicators can also be found in definitions by Nielsen (1993, p. 26) and Shneiderman & Plaisant (2009, p. 34).

Nielsen	Shneiderman
Efficiency	Speed of performance
Learnability	Time to learn
Memorability	Retention over time
Errors/Safety	Rate of errors by users
Satisfaction	Subjective satisfaction

Table 4.1.: Similar usage factors as described by Nielsen (1993, p. 26) and Shneiderman & Plaisant (2009, p. 34); comparison adopted from Van Welie et al. (1999, p. 3).

On the lower level of the layered model are typical *means* of usability that are often described in usability guidelines and heuristics. According to Van Welie, these means are not goals in themselves, but rather affect the usability indicators in a positive or negative way. Means need to be used with respect to the specified user group, user goals and context of use (cf. the ISO definition). In order to decide when to use which means, the designer can consult different domains of knowledge, which form the lowest layer of the model.

Usability engineering

By breaking down the abstract concept of usability into more concrete components and criteria it becomes possible to systematically *design for usability*, a process that is referred to as *usability engineering* (cf. Nielsen, 1993). The basic idea of a systematic usability engineering process is to embed it into the development lifecycle of a product, and to consider usability as a crucial design factor throughout the whole engineering process. The ISO standard on *human-centered design for software* (ISO 9241-210, 2010) describes a cyclic usability engineering approach that accompanies the whole design process (hence it is referred to as *human centered design*). The ISO model consists of four elementary stages that can be iterated whenever it becomes necessary during the design process: (1) First of all, it is important to gather information about the intended context of use, and to specify it as a basic input for the rest of the design process. (2) It is also necessary to specify the requirements of the product that is about to be developed. (3) Next comes the production of design solutions, which should meet the previously defined requirements and also take into account the specified context of use. (4) The last stage in this human-centered development cycle is the evaluation of the design solutions. If the requirements are met, the design is finished – if not, the designers need to iterate the previous phases until they are able to evaluate a design solution as successful.

4.3. Usability testing and evaluation methods

When to test?

Why to test?

This section gives an overview of the wide spectrum of the available usability testing methods. Such an overview is necessary to legitimate the choice of method for the evaluation study described in section 4.5.1. The design and implementation of a software system, or any other kind of product, typically is a continuous process. The first question that needs to be answered when conducting a usability evaluation therefore is "When to test?". Accordingly, evaluations can be distinguished by the moment of their application: *Formative* evaluation methods are mostly used during the design process of a new system, whereas *summative* methods are rather used to assess the quality of a finished product (Rosson & Carroll, 2002, p. 228). The previous question is closely related to the question about the "Goal of the evaluation?". Gediga et al. (2002, p. 3) identify the following three types of goals:

1. **Comparison**: "Which one is better?" – Compare several systems from one domain to find out which system is best / worst.
2. **Summative judgement**: "How good is it?" – Evaluate a finished system to assess its overall quality.
3. **Reveal problems**: "Why is it bad?" – Evaluate one or more systems to reveal weaknesses.

According to Gediga et al. (2002, p. 3), the first two goals are usually achieved during summative evaluations, while the third goal is more likely to be an instance of formative evaluation, toward the end of the design process of a product. There are several usability methods that can be used to achieve these goals. Basically, they can be distinguished as being *empirical* or *analytic* methods[67]. Rosson & Carroll (2002, p. 228-230) summarize the main characteristics of both classes of methods as follows: Empirical methods typically investigate the usability of a system by observing actual users and their characteristic behaviors while using the system[68]. Empirical user testing may be more formal (e.g. controlled experiment in a laboratory setting) or more informal (e.g. field studies), but in the end the results always need to be interpreted by a usability expert, which makes these methods generally more labor-intensive than most analytic methods. Examples for this kind of methods are *controlled experiments*, *think-aloud experiments* or *field studies* that collect feedback from the real-world usage of a system (Rosson & Carroll, 2002, p. 230). Analytic methods on the other hand are less laborious than empirical methods, as they do not rely on the test results of actual users, but rather build on the analytic skills of one or more usability experts who try to put themselves in the position of actual users. This aspect is underlined by Nielsen (1994d, p. 413), who notes that it can be difficult or expensive to recruit real users in sufficient numbers. Besides formal analytic methods such as model-based analyses[69], which are using "established theories in science and engineering to build a predictive model" (Rosson & Carroll, 2002, p. 235), there is also a number of more informal analytic methods. Most of these methods belong to the class of so called *inspection methods*. Nielsen suggests *inspection* as a cost-efficient alternative[70] to empirical methods. Usability inspection methods generally rely on evaluators who analyze and judge a user interface with regard to aspects of usability (Mack & Nielsen, 1994, p. 1). As inspection methods are based on the judgment of usability and / or domain specialists, they are sometimes also called *expert-based methods*. Nielsen (1994d) provides an extensive overview of available inspection methods, among which are the *heuristic evaluation* (Nielsen, 1994c; Nielsen & Molich, 1990), the *cognitive walkthrough* (Lewis et al., 1990; Polson et al., 1992; Wharton et al., 1994), the *pluralistic walkthrough* (Bias, 1991, 1994), and many more.

How to test?

Empirical methods

Analytic methods

Inspection methods

[67] Nielsen (1994d, p. 413) identifies *automatic evaluation* as a category of its own. It is, however, not part of this overview, as it is not really a method, but rather a tool that allows the evaluator to automatically check predefined interface specifications.

[68] For the digital humanities context of this work, it is worth noting that user studies are particularly popular in the field of digital library development (cf. Dobreva et al., 2012).

[69] One of the most prominent examples for this type of analytic methods is the *GOMS* (goals, operators, methods, and selection rules) analysis (Card et al., 1983).

[70] Jakob Nielsen also coined the terms *Discount Usability* and *Guerrilla HCI* (cf. Nielsen, 1994b), which basically describe low-cost methods that can be used for usability evaluations. The bottom line of this approach is that it is better to conduct usability tests at a small scale or with only few resources than to do without any evaluation at all.

4.4. Characteristics of the application domain

While chapter 2 has introduced general concepts from the area of annotation theory, this section addresses the application domain of linguistic annotation tools in more detail, by describing characteristics such as typical user groups and their requirements as well as concrete tasks. As the field of existing annotation tools is very heterogeneous, a classification of existing annotation tools and related software is provided as well.

4.4.1. User groups for linguistic annotation tools

Typical annotation tool users

It is important to consider different groups of users when gathering the requirements for an annotation tool, as the requirements might differ with regard to a user's individual domain experience and technological knowledge (EAGLES Evaluation Working Group, 1999, p. 25). Such different user groups of annotation tools have long been identified and described in respective literature: Carletta & Isard (1999, p. 13 ff.) introduce three different tool users: *the coder*[71], the *coding consumer*, and the *coding developer*. Dybkjaer, Berman, Bernsen, et al. (2001, p. 3) identify three similar user groups in the context of multimodal annotation tool users. The most recent adaption of these basic user groups is to be found in Reidsma et al. (2005, p. 1), who suggest *annotators*, *corpus consumers*, and *corpus developers*:

Annotators are users who are typically domain experts, i.e. trained linguists, who are aware of linguistic theories and phenomena. They are, however, not trained in programming and markup languages, which means they need an easy-to-use tool to speed up their work. Related quotes:

> Users who need a tool for their annotation task. They should not be bothered about data representation, internal design, or API design. A tool should help them work as quickly and efficiently as possible. (Reidsma et al., 2005, p. 1)
>
> Typically, these users are experts in their area . . . and they regard the annotation tool simply as a vehicle for making their work more efficient and its results more useful and more widely available (Dybkjaer, Berman, Bernsen, et al., 2001, p. 3)
>
> [These users] are typically the cheapest labour source available. They do not wish to know anything about how the coding interface works or even how different sets of tags relate to each other. Their needs are fairly simple: an intuitive coding interface so that they can concentrate on the code distinctions, documentation of how to use the interface . . . and the coding instructions nearby. (Carletta & Isard, 1999, p. 13)

Corpus consumers are people who use the annotated corpus, hoping to discover new insights about language, to test hypotheses and to answer re-

[71] *Coding* in this context does not mean programming computers, but rather is used as a synonym for annotation.

search questions. As they typically have to create the annotation scheme in advance, they are not only domain experts, but also have the technical skills to implement an annotation scheme by means of SGML or XML. They are double experts who understand the linguistic as well as the technical implications of a particular scheme design for annotation and later querying of the data. Related quotes:

> Users who want to use annotated data for all kinds of reasons. . . . They have needs for querying and browsing annotated data. (Reidsma et al., 2005, p. 1)
>
> User who have become so used to data coding formalisms, such as SGML or XML, or who experience that existing editors are not good enough so that they feel most comfortable if they can edit the coded data directly. (Dybkjaer, Berman, Bernsen, et al., 2001, p. 3)
>
> Whatever the reason for interest in the corpus, consumers are united in their need to ask questions of the corpus, looking for places which match a specific form, and to display the results. (Carletta & Isard, 1999, p. 14)

Corpus developers build on an existing scheme or corpus, and modify it to suit their particular needs[72]. They have deep knowledge in the domain of linguistics, but also have advanced skills in the technical area of markup and programming languages. Related quotes:

> Users responsible for corpus design and maintenance (e.g. design of new annotation schemas or altering existing ones, understanding of data representation supported by the tool and mapping of their data to existing structures. (Reidsma et al., 2005, p. 1)
>
> [These users wish] to design their own coding schemes, either to improve on the reliability or suitability of an existing scheme or in order to test a particular research question. (Carletta & Isard, 1999, p. 15)

Previous studies by Burghardt & Wolff (2009b) have shown that the majority of existing annotation tools are being developed for the "corpus developers", i.e. users of tools are required to have linguistic domain knowledge as well as technical knowledge about markup languages. This work will focus on the needs of users from the first group: plain "annotators", who are experts in their linguistic domain, without deeper technical knowledge about markup and programming languages.

Humanities scholars

In addition, a fourth group of users, who may be best described as *traditionalists*[73], is proposed in this dissertation. Traditionalists are similar to the first group of plain annotators, who are first and foremost experts in their domain, and who are willing to learn and use digital annotation tools as a means to achieve their

[72] In earlier work, Reidsma et al. (2004) distinguish corpus developers from system developers. This distinction does not seem relevant for the focus of this work, therefore the user groups from their more recent work are adapted. Dybkjaer, Berman, Bernsen, et al. (2001, p. 3) promote a similar view on the role of the *developer*.

[73] Note: I have used a similar argumentation in an article on an interactive tool for the visualization and analysis of Shakespeare plays in Wilhelm et al. (2013, p. 1-2)

scholarly tasks. While the first user group may be characterized as *digital humanists*, there remains the larger group of traditional humanists who may never have used digital tools and resources in their scholarly practices because they do not have the technical skills, or the tools do not provide a user-friendly interface (also cf. the characterization of the *typical* linguist by McEnery & Hardie, 2012, p. 33):

> At present too many digital resources require users either to struggle with unfriendly interfaces or to be technical experts even to begin to use them. (Warwick et al., 2008, p. 16)

Although ideas such as "Distant Reading" (Moretti, 2007, 2013) and "Culturomics" (Michel et al., 2011) are about to proclaim a new era of text analysis, many traditional humanists still wince at digital methods. While Stone (1982, p. 300) mentions the general belief that "it may be part of the humanistic tradition to be anti-machine", it is also possible that the main reason for a rather reluctant use of digital tools and resources is grounded in the technical hurdles and complicated interfaces of existing systems. A similar view is held by Warwick (2012), who notes that traditional humanists may well behave different in terms of scholarly practices as compared to the sciences, but are not per se *luddites*:

> Could it be that users did not adopt resources because they were not useful or did not fit what they would like to do as scholars? Could there be other reasons to do with design, content, presentation, or documentation? (Warwick, 2012, p. 2)

Another reason is described by Gibbs & Owens (2012, para. 8), who asked 213 historians how they used digital tools and resources for their scholarly practices, and to which extent these tools met their requirements: The authors describe a so called "expectation gap" (Gibbs & Owens, 2012, para. 19), which means that "non-technical users either could not generally appreciate what several of the more complex tools were designed to do, or were unable to recognize their potential value" (Gibbs & Owens, 2012, para. 18). As a conclusion, Gibbs & Owens (2012, para. 6-7) stress that digital humanities tools need to focus on ease of use and transparency (i.e. it should be clear how the tool can support the user) in order to attract more traditional humanities scholars to make use of digital methods. The usability patterns suggested in this dissertation are meant to help developers build annotation tools that are also attractive for this fourth group of users:

Traditionalists are users who refrain from using existing digital tools because of technical complexity and a misunderstanding of what such tools are capable of (Gibbs & Owens, 2012, para. 21). Interfaces for this user group do not only need to abstract technical complexity, but also need to be largely self-explanatory, transparent, intuitive and well documented. To capture the special requirements of this user group, I propose the term *humanist-computer interaction*. Related quotes:

Humanist-computer interaction

> Most users, especially humanities academic users, do not want to have to be trained to use digital resources, regarding it as a waste of time. (Warwick, 2012, p. 9)
>
> For many potential users, existing software still seems very hard to learn. (Unsworth, 2003, para. 8)
>
> If users sense that something looks 'wrong' – which may simply mean that the interface looks unfamiliar, is difficult to use or lacks information about its creation and provenance – users may regard it as untrustworthy, neglect it and revert to more familiar resources, whether printed or digital. (Warwick, 2012, p. 14)

This section has introduced four basic groups of annotation tool users. Two groups, the plain *annotator* and the *traditionalist*, will be in the center of the evaluation study described in section 4.5.

4.4.2. Requirements for linguistic annotation tools

Multimodal annotation tools

Like for the definition of basic user groups, there are also many examples for annotation tool requirements in the respective literature. One branch of publications is dedicated to requirements in the context of multimodal annotation: Dybkjaer, Berman, Kipp, et al. (2001) review twelve tools for the transcription, annotation, information extraction and analysis of natural and multimodal interaction data, and extract a list of requirements that are intended to reflect the general user needs for such tools (Dybkjaer, Berman, Bernsen, et al., 2001, p. 1-2)[74]. Garg et al. (2004) describe a list of generic requirements for transcription and annotation tools for multi-party, multi-modal dialogues. Dipper et al. (2004, p. 2-3) present a list of requirements for annotation tools in the context of a large-scale, multimodal project on linguistic information structuring.

Project management

Kaplan et al. (2011) describe their work as complimentary to the requirements formulated by Dipper et al. (2004). Although the context of their publication is not multimodal, but rather focused on written texts, it presents an extension of annotation tool requirements on the project management level by introducing requirements such as *user* and *role management*, *delegation of tasks*, *diffing*, *merging*, *version control*, etc.

Linguistic corpora

Another branch of requirements is not directly focused on tools, but rather formulates requirements for the creation of linguistic corpora and annotations in general: Ide & Brew (2000) discuss issues related to the ecology of corpora and suggest some general requirements for reusable and extensible language data, which also include tools. The authors argue for standardized, interoperable tool frameworks that can be tailored to specific needs by means of a component system rather than reinventing proprietary tool solutions over and over again (Ide

[74] Both reports by Dybkjaer et al. were preliminary studies for the development of the MATE workbench (cf. Isard et al., 2000), a multimodal annotation tool that is no longer available.

& Brew, 2000, p. 3-4). Leech (1993, p. 1) defines linguistic annotation as "the practice of adding interpretative . . . information to an existing corpus" and hence suggests a set of maxims that are meant to support the distinction between representational (primary data) and interpretive (annotations) data. Some of these maxims can also be interpreted as functional requirements for annotation tools.

Relevance for user groups

Table 4.2 gives an overview of altogether 50 requirements for annotation tools that are described throughout respective literature (S1 = Leech (1993), S2 = Ide & Brew (2000), S3 = Dybkjaer, Berman, Bernsen, et al. (2001), S4 = Garg et al. (2004), S5 = Dipper et al. (2004), S6 = Kaplan et al. (2011)). As many requirements were described redundantly by the different authors[75], a total number of 17 unique requirements could be extracted[76]. The requirements were also categorized according to their relevance for the previously defined four user groups of annotation tools (cf. section 4.4.1): Traditionalists (T), annotators (A), corpus consumers (CC), and corpus developers (CD).

Requirement	**User group**	**Aspect**	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**
Transparency and reusability of annotation scheme	CD	F	2	1	–	–	–	–
Tool portability	A, CC, CD	F	–	–	–	1	–	–
Modular architecture	CD	F	–	1	2	–	–	–
Support of multi-modal data	A, CC	F	–	1	1	1	–	–
I/O flexibility	CD	F	–	1	2	1	1	1
Robustness and stability	A, CC	F	–	–	1	–	–	–
Multi language support	A, CC	F	–	1	–	–	–	1
Multiple levels of annotation	A, CC	F, U	–	1	1	–	1	1
Integration of automatic annotation tools	A, CC	F, U	–	–	1	–	–	1
Visualization of primary data	T, A	F, U	1	–	1	–	1	–
Visualization of annotations	T, A	F, U	–	–	1	–	–	–
Flexibility of annotation scheme	A, CC	F, U	–	–	1	2	1	2
Annotator management	A, CC, CD	F, U	1	–	–	–	–	2
Annotation management	A, CC, CD	F, U	–	1	–	1	1	2
Marking of anchors	T, A	F, U	–	1	–	1	1	–
Documentation	T, A	U	–	–	–	1	–	–
Easy-to-use interface	T, A	U	–	–	1	1	1	–

Table 4.2.: Requirements for annotation tools (based on a review of appropriate literature). The usability requirements that are relevant for this work are rendered in italic type.

[75] Sometimes requirements were even described redundantly by the same author. In these cases the count in the table is "2".

[76] This approach is very similar to previous work by Reidsma et al. (2004, 2005). I decided to adapt and extend this existing work, because I wanted to add some more requirements from different authors. Besides, I did not always agree with the wording or the overall interpretation of certain requirements. The credits for the initial idea to such a systematic approach for the documentation of requirements for annotation tools must, however, be given to Reidsma et al.

Functionality vs. usability

While requirements are commonly defined to check a system's overall quality or acceptability (Nielsen, 1993, p. 25), it is important to distinguish different types of requirements. The ISO/IEC 9126-1 standard on "Software engineering – Product quality" defines a quality model that comprises of six different characteristics which in turn can be broken down into further sub-characteristics. However, the criteria that are used most often to categorize requirements and respective evaluation criteria are *functionality* and *usability* (cf. Dipper et al., 2004; Burghardt & Wolff, 2009b)). According to Dipper et al. (2004, p. 3), functional requirements describe the *tool-task* relation, i.e. how well a tool is suited for achieving typical tasks, whereas the usability category captures requirements that concern the *tool-user* relation, which is also known as human-computer interaction. Nielsen (1993, p. 24-25) suggests a similar model, splitting up the criterion of *usefulness* into *utility* (functionality of a system) and *usability* (how well can the system's functionally be used). Accordingly, the requirements were categorized as describing an aspect of *functionality* (F) or of *usability* (U), or both.

Functional requirements

Seven of the seventeen requirements have been categorized as being purely functional. These include *transparency and reusability of annotation scheme*, a requirement that addresses the format of the annotation scheme, but has no direct effect on the human-computer interaction with an annotation tool. The requirements *tool portability* and *modular architecture* are important for tool developers (cf. the user group of *corpus developers*) and people who have to migrate an existing tool and / or annotation project to another platform. Again, there is no direct connection to the usability on the interaction level. *Support of multi-modal data* is a functional requirement that comes from the context of a multi-modal annotation scenario and is not relevant for text annotation tools (specific scope of this work). *I/O flexibility*, *robustness and stability* as well as *multi language support* describe functions of an annotation tool that should not be missing, but cannot be improved on the HCI-level. These seven purely functional requirements will not be considered when designing the tasks for the evaluation study on the usability of annotation tools.

Usability and mixed requirements

It also shows that only two of the requirements are pure usability requirements. These are *documentation*, which is important for the learnability of a system, and the general purpose requirement of an *easy-to-use interface*, which will be the main subject of this work. Most of the requirements are functional to a certain degree, but they also have the potential to be improved on the level of human-computer interaction, which is directly related to usability. For this reason, eight requirements have been categorized as mixed requirements (F, U). The requirements relevant for this work are either pure or mixed usability requirements that address at least one of the two previously defined core user groups (*annotators* and *traditionalists)*. They give important hints for an evaluation design that will reveal many interesting aspects about the usability of annotation tools. The following passage describes ten relevant requirements in some more detail:

R01 – Multiple levels of annotation Corpora are usually annotated on several levels of linguistic information. Oftentimes, lower level annotations such as POS are needed for the annotation of more complex, syntactic phenomena. Annotation tools should allow the user to annotate on multiple, parallel levels of information. Technically, such multi-level annotations are realized by means of stand-off annotation (cf. chapter 3.4.1).

R02 – Integration of automatic annotation tools The previous chapter has described automatic and manual annotation approaches. Some aspects of language are relatively easy to annotate and can thus be automated quite well. Other tasks are more complex and require manual annotation. An annotation tool should provide an interface that allows the user to integrate automatic tools (cf. Castilho et al., 2007) such as POS taggers or lemmatizers, as well as preprocessing tools such as tokenizers, sentence splitters, etc.

R03 – Visualization of primary data The primary data, i.e. the original text, should be displayed correctly (with all special characters) in the annotation tool. If the corpus consists of several documents, the annotation tool should allow the user to navigate between the single documents. The tool should visualize the primary data in such a way that it can be annotated on different linguistic levels. It must, however, be clear where the original text ends and where the annotation begins.

R04 – Visualization of annotations Much like the primary data, the annotations need to be visualized in an appropriate way. This becomes more challenging with every additional annotation level. The tool should provide flexible annotations and allow the user to control which annotations are shown at a given point in time, to reduce unnecessary interface complexity.

R05 – Flexibility of annotation scheme Ideally, an annotation scheme has been tested successfully before it is used on an actual corpus project. In most cases, there is no way of knowing if a scheme that was developed a priori covers all phenomena that actually appear in the primary data. Therefore, it is common to modify and extend the annotation scheme gradually during the annotation process. An annotation tool should allow the user to display and modify an annotation scheme inside the tool.

R06 – Annotator management Typically, larger corpus projects are annotated by multiple persons. Therefore, an annotation tool should provide basic mechanisms to manage different user roles (e.g. annotators, administrators, etc.) and to assign different annotation tasks to different persons, to avoid redundant annotations and to document authorship information about the annotations in the corpus (cf. Kaplan et al., 2011).

R07 – Annotation management The annotations themselves need to be managed, too: During the course of an annotation project an annotation may be modified (correction, deletion) several times. For this reason it is important to provide a version control for the annotations and the corpus as a whole. Annotators should also be able to comment on annotations, to mark linguistic problems that need to be double-checked or that need to be analyzed at a later point in time. Finally, a tool should be able to check if annotations are consistently used by different annotators (inter-annotator agreement).

R08 – Marking of anchors One of the most important interaction[77] steps during the annotation process is the selection of anchors in the primary data. The tool should provide an interface for an intuitive and effective selection of different anchor scopes (cf. chapter 3.4.3). Particularly challenging is the marking of anchors that are distributed among the primary data, possibly even on different pages.

R09 – Documentation This requirement describes the availability of a user manual and possibly active support for the tool in the form of a newsgroup or a mailing list. Documentation is important for the learnability of an annotation tool.

R10 – Easy-to-use interface The last requirement is described rather unspecific in the reviewed literature and may be seen as a placeholder for all kinds of different human-computer interaction aspects that are important during the course of an annotation scenario. As this requirement is most important for the focus of this work, more specific interaction elements that need to be considered when designing a user-friendly annotation tool, will be presented in the next section (cf. section 4.4.3).

This section has introduced some general requirements for annotation tools, which have been extracted from relevant literature. Basically, requirements can be categorized as being functional and / or being a usability requirement. It has become obvious that usability is connected to many functional requirements and that usability itself is often formulated as a rather generic requirement ("easy-to-use interface"). The next section describes the typical process of linguistic annotation with a digital tool and clarifies the concept of an annotation tool interface and corresponding user interactions and tasks.

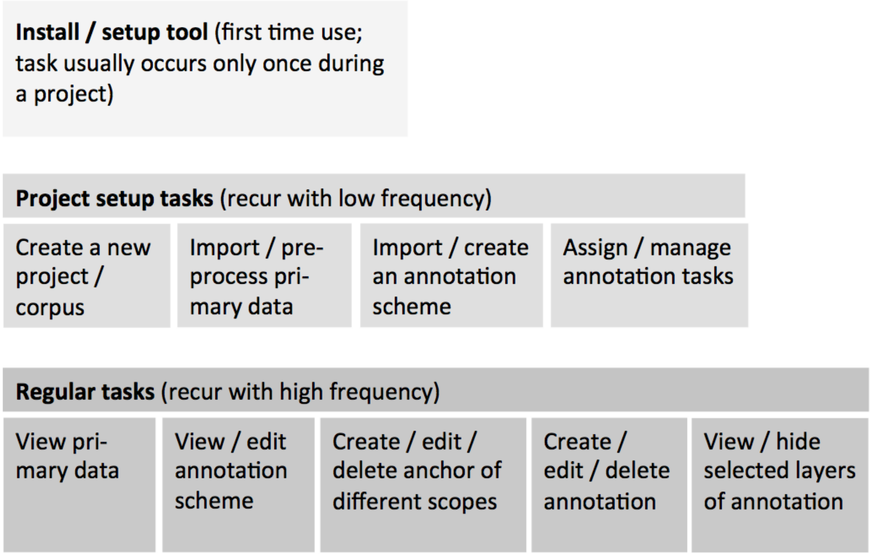
4.4.3. Typical micro-tasks during the annotation process

This section presents a basic annotation workflow[78] and corresponding micro-tasks. Higher level tasks such as "Annotate all noun phrases in a given docu-

[77] Another important interaction aspect is the selection of an appropriate item from the annotation scheme. Although it is not described as a requirement in respective literature, this aspect will be taken into account when designing tasks for the intended evaluation of annotation tools.

[78] For a more comprehensive view on the *whole* annotation process – including corpus selection, annotation scheme creation, annotation evaluation, etc. – cf. Palmer & Xue (2010, p. 258ff.).

ment" typically can be subdivided into a number of smaller, recurring micro-tasks such as "Import document", "Select anchor", "Apply annotation", etc. Typical micro-tasks for linguistic annotation tools have been identified in previous evaluation studies (cf. Dipper et al., 2004; Burghardt & Wolff, 2009b; Burghardt, 2012). In the following, a basic interaction model that contains elementary tasks that affect human-computer interactions during the annotation process[79] is presented (cf. Figure 4.2).



Install / setup tool (first time use; task usually occurs only once during a project)

Project setup tasks (recur with low frequency)

Create a new project / corpus

Import / pre-process primary data

Import / create an annotation scheme

Assign / manage annotation tasks

Regular tasks (recur with high frequency)

View primary data

View / edit annotation scheme

Create / edit / delete anchor of different scopes

Create / edit / delete annotation

View / hide selected layers of annotation

Figure 4.2.: Interaction model with typical tasks during the annotation process.

Classes of tasks

After installing and setting up the tool and the working environment, which may include the configuration of a local server and a database, there are two main classes of user interactions with the annotation tool (Kaplan et al., 2010, p. 512): The first class contains interactions necessary to configure the overall corpus and project settings, which include the creation of the actual corpus, the preparation of the primary data, the definition of an annotation scheme as well as the basic management of annotator roles and the assignment and management (versioning, diffing, etc.) of annotation tasks. Tasks from this class recur with a lower frequency, and are typically available only for project administrators. The second class of tasks recurs with a much higher frequency, as it describes typical interactions that occur during the annotation process. These include viewing the primary data and the annotation scheme, the ad hoc modification of the annotation scheme, and most importantly, a mechanism to create anchors with various scopes within the primary data and to assign values from

[79] A predecessor of the annotation process and task model described in this chapter has also been presented at the workshop "JMCE & RECON Workshop on Computer-Aided Methods of Textual Analysis: How to Analyse Millions of Texts and Still Be Home for Tea? Innovations in Textual Analysis in the Social Sciences." in Berlin, 2010 (no publication available).

the annotation scheme to those anchors. Another important task is about being able to display the appropriate amount of annotation data, which is particularly challenging when many levels of annotation are involved[80].

4.4.4. Categorizing linguistic annotation tools

Linguistic tools and resources

As the landscape of annotation software is characterized by a vast number of different tools and a high degree of heterogeneity, there have been several attempts to gather existing linguistic annotation tools and to make them available to others via overview pages on the Internet. Among the most extensive of such tool overviews are the *Bamboo DiRT wiki*[81], which provides a huge collection of all kinds of tools that are relevant for the digital humanities (including a number of linguistic annotation tools) and the *TAPOR* portal[82], which has gathered numerous research tools for textual study.

Steven Bird and Mark Liberman created one of the first overview pages for the creation and managing of linguistic annotations. The page was hosted on the servers of the *Linguistic Data Consortium* (LDC)[83], but is no longer online. The most recent overview page for linguistic annotation tools is the *Linguistic Annotation Wiki* (LAW)[84] which is part of the German *EXMARaLDA* project. While the existence of such overview pages is very helpful for identifying available annotation tools that can be used as test objects in a systematic usability evaluation, it is important to distinguish linguistic annotation tools from other tools and resources that are typically involved in the creation and analysis of language corpora. Each of the above overview pages mixes up tools and resources from the following sub-categories:

Initiatives and organizations who create standards and resources; examples: *EAGLES* (Expert Advisory Group on Language Engineering Standards)[85], *ISLE* (International Standards for Language Engineering)[86], *ISO/TC37/SC4* (Terminology and other language and content resources)[87]

[80] The "export" of the annotated data in a reusable, standardized format, has not been included in the typical annotation workflow, as it marks the transition to another stage in the corpus creation process (cf. Figure 3.1 – "Typical stages during the creation of a corpus"). Moreover, many tools offer built-in analysis and query features that do not make export of data necessary at all.

[81] http://dirt.projectbamboo.org/

[82] http://tapor.ca/

[83] http://www.ldc.upenn.edu/

[84] http://annotation.exmaralda.org/index.php/Linguistic_Annotation

[85] Examples without an explicit URL have already been referenced in the previous chapters, and are therefore not referenced again to avoid redundant information in this dissertation.

[86] http://www.ilc.cnr.it/EAGLES96/isle/ISLE_Home_Page.htm

[87] http://www.tc37sc4.org/

Meta formats and standards for data exchange; examples: *LAF* (Linguistic Annotation Framework)[88], *PAULA* (Potsdam Exchange Format for Linguistic Annotations)[89]

Annotation schemes and tag sets for the linguistic context; examples: *TEI* (Text Encoding Initiative), *XCES* (Corpus Encoding Standard for XML), *STTS* (Stuttgart-Tübingen Tagset)[90]

Annotated corpora, ready for querying; examples: *BNC* (British National Corpus), *DWDS* (Digitales Wörterbuch der Deutschen Sprache)[91]

Tools for text analysis, typically for documents without annotation or encoding; mostly statistical programs or concordancing programs; examples: *AntConc*[92], *R Project*[93], *Wordcruncher*[94], *WordSeer*[95], *WordSmith*[96], *Voyant*

Tools for corpus analysis, typically for collections of annotated documents; examples: *ANNIS* (ANNotation of Information Structure)[97], *CWB* (Corpus Workbench)[98], *eXist* (Open Source Native XML Database)[99]

Tools for text manipulation and preprocessing such as filtering, sorting, substitution, or normalization; examples: *grep*[100], *TAPoR* (Text Analysis Portal for Research)[101]

Tools for transcription and time-alignment of speech and video data; examples: *Praat*[102], *TranscriberAG*[103]

Tools for field linguists, typically for the creation of dictionaries while collecting data from native speakers in the field; examples: *Field Linguist's Toolbox*[104], *FLEX* (Fieldworks Language Explorer)[105]

[88] http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=37326

[89] http://www.sfb632.uni-potsdam.de/en/paula.html

[90] http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/GermanTagsets.html

[91] http://www.dwds.de/

[92] http://www.antlab.sci.waseda.ac.jp/software.html

[93] http://www.r-project.org/; also cf. the practical introduction to "R for corpus linguistics", by Gries (2009).

[94] http://www.wordcruncher.com/

[95] http://wordseer.berkeley.edu/

[96] http://www.lexically.net/wordsmith/

[97] http://www.sfb632.uni-potsdam.de/annis/

[98] http://cwb.sourceforge.net/

[99] http://exist-db.org/exist/apps/homepage/index.html

[100] http://www.gnu.org/software/grep/

[101] http://taporware.ualberta.ca/

[102] http://www.fon.hum.uva.nl/praat/

[103] http://transag.sourceforge.net/

[104] http://www-01.sil.org/computing/toolbox/

[105] http://fieldworks.sil.org/flex/

Tools for web page annotation, typically in the context of semantic web annotation; examples: *AnnotateIt*[106], *Domeo*[107], *Pundit*[108]

Tools for PDF annotation, typically for desktop computers and tablet PCs[109]; examples: *Adobe Acrobat*[110], *GoodReader*[111]

Tools for qualitative data analysis typically allow the user to apply *codes* (=annotations) to qualitative data, and to analyze the data according to these codes; examples: *MAXQDA*[112], *ATLAS.ti*[113]

Virtual research environments are typically complex frameworks that support collaboration between scholars. Together with other processing and analysis tools, annotation tools may be part of these infrastructures; examples: *eHumanities Desktop*[114], *eSciDoc*[115], *TextGrid*[116]

Editing software for structured documents, typically XML or other markup editors[117]; examples: *Altova*[118], *Oxygen*[119], *TEXnicCenter*[120], *TexShop*[121]

Tools for automatic annotation, mostly command line, but also as complex frameworks/processing architectures; examples: *CLAWS* part-of-speech tagger for English[122], *TreeTagger*[123], *UIMA*, *WebLicht*[124]

Tools for manual annotation, basically tools that allow the user to create an annotation scheme and to manually apply annotations. As the focus of this work is on annotation tools from this class, Table 4.5 shows many examples for this type of tools.

[106] http://annotateit.org/

[107] http://swan.mindinformatics.org/index.html

[108] http://www.thepund.it/

[109] For a comprehensive overview of annotation tools for the iPad cf. Hastreiter et al. (2013, p. 5).

[110] http://www.adobe.com/products/acrobat.html?promoid=JOLIR

[111] http://www.goodreader.com/

[112] http://www.maxqda.com/

[113] http://www.atlasti.com/

[114] http://www.hucompute.org/ressourcen/ehumanities-desktop

[115] https://www.escidoc.org/

[116] http://www.textgrid.de/

[117] For related work on the usability of editing software for structured documents cf. Flynn (2009).

[118] http://www.altova.com/

[119] http://www.oxygenxml.com/

[120] http://www.texniccenter.org/

[121] http://pages.uoregon.edu/koch/texshop/

[122] http://ucrel.lancs.ac.uk/claws/

[123] http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

[124] http://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/Main_Page

As was described in chapter 3.6.3, only the last category of "Tools for manual annotation" is relevant for this evaluation study. Tools from that category may be further distinguished by three criteria[125]: Firstly, annotation tools may be distinguished by the *annotation modalities* they support (tools that support multiple modalities are called multi-modal tools, also cf. chapter 3.4.2). There is a large number of tools that can be used to annotate dynamic data (e.g. audio or video files). Such tools typically have a built-in transcription feature and provide a mechanism to align the transcription along a virtual timeline. Examples for such tools are *Anvil*, *ELAN*[126], and *EXMARaLDA*. There are also several tools that are specialized in the transcription and annotation of static images, such as *DM*[127] or *TILE*[128]. Finally, a large number of tools is dedicated to the annotation of text documents, which usually need no explicit transcription.

Modality of annotation

The second criterion to distinguish annotation tools is their *type of software*. Software types may reach from simple stand-alone programs to complex annotation and text processing-frameworks, or more abstract programming toolkits and APIs. While Kaplan et al. (2011, p. 99) distinguish web-based tools and desktop applications, stressing the architecture of the tool, the focus of this work is on the technical complexity and the immediate applicability of a tool (graphical user interface vs. collection of abstract functions and programming routines). Two extreme examples are the ready-to-use, web-based tool *CATMA*[129], an application that can be used immediately from within the web browser, and the *Annotation Graph Toolkit* (AGTK)[130], a formal framework for representing linguistic data as graphs.

Software type

A third criterion is introduced by Kaplan et al. (2011, p. 99-100). Tools may be classified by the scope of their applicability to different annotation scenarios:

Task scope

- *Project-specific tools*: Very specialized for one specific project; oftentimes the data format is not interoperable, but rather proprietary, and the annotation scheme may be predefined, and unchangeable.
- *Task-oriented tools*: More generic, but still rather specific concerning a certain kind of task, e.g. the creation of an annotated treebank.
- *Generalized, multi-purpose tools*: Generic and "capable of adapting a variety of tasks" (Kaplan et al., 2011, p. 100); typically the architecture of such tools is flexible and allows the user to include plugins with different functionality in a modular way.

[125] The criteria "Modality of annotation" and "Software type" were identified in a previous study that compared more than 50 annotation tools in the context of a diachronic annotation project (cf. Burghardt & Wolff, 2009b).

[126] http://tla.mpi.nl/tools/tla-tools/elan/

[127] http://dm.drew.edu/dmproject/

[128] http://mith.umd.edu/tile/blog/

[129] http://www.catma.de/

[130] http://agtk.sourceforge.net/

4.5. Evaluation design

As there is already a large number of existing annotation tools (cf. section 4.5.5), the goal of this dissertation is to provide generic solutions for the creation of easy-to-use annotation tool interfaces, which can either be applied for the design of new tools, or for the redesign of existing tools. These generic solutions will be derived from a *comparative analysis* (cf. Nielsen, 1993, p. 78-79) of available annotation tools. The evaluation builds upon the user groups, requirements, tasks, and annotation tool characteristics that have been presented in the previous section. This section describes the key aspects of the evaluation design, such as the choice and description of the testing method, the generation of test tasks, a related pilot study, the selection of appropriate test objects (i.e. annotation tools), and the actual evaluation procedure.

4.5.1. Reflections on the choice of evaluation method

In Table 4.3, the most important decisions for the selection of an appropriate usability evaluation method for linguistic annotation tools are summarized. These decisions will be discussed in more detail in the following passages.

Characteristic	**Decision for the evaluation of linguistic annotation tools**
Type	Formative
Goal	Reveal problems
Approach	Analytic approach (inspection method)
Method	Heuristic walkthrough

Table 4.3.: Characteristics of the chosen evaluation method.

Type and goal of evaluation

First, the type and goal of the evaluation will be explained in more detail: Although the test objects of the evaluation are readily implemented annotation tools, it is not a summative evaluation approach. Essentially, this approach puts a number of finished tools back to the drawing board, to re-evaluate them with regard to usability qualities. Therefore, it is a formative evaluation that uses existing products as high-level prototypes which can be analyzed with regard to aspects of interaction design and usability issues:

> Prototyping is an important part of the usability process, and existing, perhaps competing, products are often the best prototypes we can get of our own product (Byrne, 1989) . . . If several competing products are available for analysis, one can furthermore perform a comparative analysis of their differing approaches to the various user interface design issues for the kind of product being studied. (Nielsen, 1993, p. 78-79)

The goal of the evaluation is to reveal typical usability problems[131] from the application domain of linguistic annotation tools. These problems will be the basis for the formulation of generic patterns for the design of user-friendly tool interfaces.

Analytic vs. empirical approach

Next, a basic decision of whether to test analytically or empirically has to be made: The question of how analytic methods are different to empirical methods has been addressed in several studies. Karat (1994) provides a "Comparison of user interface evaluation methods", where she reviews and summarizes a number of previous studies (Jeffries et al., 1991; Desurvire et al., 1991, 1992; Karat et al., 1992) that compare empirical methods to analytic methods. The bottomline of this review is that in most cases empirical user testing is favored over inspection methods, as it has some overall advantages, like for instance being suited for a wider range of evaluation goals, being able to reveal more usability problems, etc. (Karat, 1994, p. 221). One aspect that is particularly important for the context of this dissertation, is how well an evaluation method supports the generation of solutions for identified problems. According to Karat (1994, p. 218-219), inspections methods in general do not facilitate the generation of design solutions and recommendations in the same way empirical methods do. This does, however, not mean that inspection methods are entirely inapt for the generation of recommendations:

> Typically, a usability inspection is aimed at finding usability problems in an existing user interface design, and then using these problems to make recommendations for fixing the problems and improving the usability of the design. (Mack & Nielsen, 1994, p. 3)

Despite some general advantages of empirical methods over analytic methods, an analytic approach was chosen for this evaluation study for several reasons:

Up-to-dateness

First of all, it must be noted that although *usability inspection* has had its peak in the early 1990s[132], a recent review (Novick & Hollingsed, 2007) of reported work on using inspection methods, such as *heuristic evaluation*, *cognitive walkthrough*, *formal usability inspection* or *pluralistic usability walkthrough*, has shown that analytic methods are still being used on a regular basis:

> Both empirical usability testing and usability inspection methods appear to be in wide use, with developers choosing the most appropriate method for their purposes and context. . . . With usability inspection methods solidly represented in the methodological repertoires of usability professionals . . . research issues are shifting from showing efficacy of the methods toward further adaption of the methods as interaction media evolve. (Novick & Hollingsed, 2007, p. 5)

Feasibility and cost

Extracting good and bad design practices from a number of existing annotation

[131] Gediga et al. (2002, p. 3-4) note that the goal to "reveal problems" is typically an instance of formative evaluation.

[132] According to Novick & Hollingsed (2007, p. 2), important milestones in the development of analytic inspection methods were the ACM CHI'92 workshop on "Usability Inspection Methods" (Mack & Nielsen, 1993), and a book with the same title, that was edited after the workshop (Nielsen & Mack, 1994).

tools by means of empirical user studies would be very laborious and costly, as several participants, who are experts in the linguistic domain, would be needed to conduct the respective evaluation. Having no access to such a pool of potential test participants clearly speaks against an empirical approach with multiple tool users. Although the recommended use of inspection methods is typically in the early phase of the development cycle (because of the relatively low cost and ease of realization), Karat (1994, p. 217) notes that there is evidence in the relevant literature, suggesting that inspection methods may also be well suited "when deciding among competing design solutions", which clearly is the case in this evaluation study. Analytic approaches typically recommend the use of multiple evaluators; the study described in this work, however, only uses a single evaluator: As this study aims at analyzing a number of different annotation tools, a single evaluator is more likely to produce consistent results[133]. The involvement of multiple evaluators (with different levels of expertise) is more prone to produce rather heterogeneous results with regard to the form, description and severity of identified usability problems. Hertzum & Jacobsen (2003, p. 200-201) have described this issue as the *evaluator effect*:

Applicability

Number of evaluators

> Based on a review of 11 studies of CW [*cognitive walkthrough*], HE [*heuristic evaluation*], and TA [thinking aloud], we have found that different evaluators evaluating the same system with one of these methods detect substantially different sets of usability problems in the system. This evaluator effect persists across differences in system domain, system complexity, prototype fidelity, evaluator experience, problem severity, and with respect to detection of usability problems as well as assessments of problem severity. In the reviewed studies, the average agreement between any two evaluators ranged from 5% to 65%, and none of the UEMs [*usability evaluation methods*] is consistently better than the others.

Results that are consistent in form and content are of particular importance for this work, as they are better suited as input for the identification of generic usability patterns for linguistic annotation tools[134]. While more evaluators obviously increase the number of usability problems, a comparative study carried out by Sears (1997, p. 225) indicates that during a heuristic walkthrough, one evaluator (as compared to the findings of five evaluators) finds approx. 70% of the serious problems, 56% of the intermediate problems, and 35% of the minor problems (cf. Table 4.4). At the same time, Sears' study indicates that one evaluator is less likely to identify false positives, whereas more evaluators tend to find more false positives. Considering that the goal of this evaluation is not to identify as many problems as possible for one specific annotation tool, but rather to identify the most common, recurring problems of a particular application domain, Sear's study supports the single evaluator approach. The lower number of identified usability problems of a system as compared to empirical

[133] It must be noted that the *consistency* of the results is achieved at the expense of *objectivity*. This trade-off will be discussed in more detail in chapter 7.1.2.

[134] For a detailed description of how to integrate an analytic inspection method with the pattern writing process cf. chapter 5.6.

Problem severity	1 Eval.	2 Eval.	3 Eval.	4 Eval.	5 Eval.
Serious	2.1	2.8	3.0	3.0	3.0
Intermediate	3.3	4.8	5.4	5.7	5.9
Minor	3.4	5.6	7.3	8.6	9.6
False positives	0.3	0.6	0.9	1.1	1.4

Table 4.4.: Number of problems identified on average by one to five heuristic walkthrough evaluators (adopted from Sears (1997, p. 225)).

methods does not affect the evaluation goals of this work too much, as the total number of evaluated systems is likely to compensate an expectedly lower number of usability problems per tool. Furthermore, the evaluation of annotation tools is conducted by a *double expert* (Nielsen, 1992), i.e. an evaluator who is experienced in both, the domain of linguistic annotation and usability testing[135]. Nielsen (1992) showed that in the case of *heuristic evaluation* a double expert is likely to find more usability problems (60 %) than a regular usability expert (41 %) who is trained in the area of usability testing, but has no specific expertise in the application domain. Unsurprisingly, so called novice evaluators, who have no prior knowledge, neither in usability testing nor in the application domain, found the lowest number of usability problems in Nielsen's study (22 %).[136]

Double expert

Choice of method

From the broad spectrum of available inspection methods, the *heuristic walkthrough* method, as described by Sears (1997), was chosen. The method has proven to be successful in other evaluation studies with similar objectives, one of which was designed as a pretest for the very evaluation design described in this section (cf. Burghardt, 2012; Burghardt et al., 2013). The heuristic walkthrough method will be explained in more detail in the following section.

4.5.2. Description of evaluation method

As the *heuristic walkthrough* borrows concepts from existing inspection methods, first of all, the key features of the *heuristic evaluation*, the *cognitive walkthrough*, and the *usability walkthrough* are introduced.[137]

Heuristic evaluation – The heuristic evaluation is an unstructured inspection method, where the evaluators do not need to perform an actual, predefined

[135] I consider myself a *double expert*, as I hold a Magister degree in Information Science and English Linguistics. I also have several years of teaching experience in University courses that address topics like corpus linguistics and digital humanities. My usability expertise is underlined by a number of publications that cover a broad range of usability-related topics (especially usability evaluations).

[136] According to Nielsen, these rates can be raised if the tests are conducted by groups of evaluators. Approx. 7-8 double experts or 15 regular usability experts are needed to identify all potential usability problems of a system; larger groups of novice evaluators (around 15) are likely to identify 75-80% of all usability problems (Nielsen, 1992, p. 377).

[137] This description of relevant inspection methods is based on a less detailed text passage from Burghardt (2012, p. 105-106).

task, but are free to explore the system on their own (Nielsen, 1994c, p. 29). Nielsen (1994c, p. 28-29), however, recommends to go through the interface at least twice, first to get a general overview of the system, and a second time to focus on specific interface elements, and judge them with regard to a list of predefined usability principles, the heuristics. Heuristics are very similar to usability guidelines, principles and rules, as they are trying to capture and promote good design in a generic way (Johnson, 2010, p. xi). There are many examples for such generic guidelines[138] and oftentimes they seem to be overlapping or are even redundant. Example:

- "Strive for consistency" (Shneiderman & Plaisant, 2009, p. 88)
- "Consistency and standards" (Nielsen, 1994c, p. 30)

Heuristics

This is largely because most of these guidelines and heuristics share a common basis and origin, which is knowledge about human psychology, for instance *perception*, *reasoning*, *memory*, etc. (Johnson, 2010, p. xiii). The following set of usability heuristics is among the most widely used heuristics[139]:

H1 Visibility of system status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

H2 Match between system and the real world: The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

H3 User control and freedom: Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

H4 Consistency and standards: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

H5 Error prevention: Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

H6 Recognition rather than recall: Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for

[138] Cf. Johnson (2010, p. xi) for an overview of some of the most prominent guidelines and heuristics in the field of HCI.

[139] The detailed description of the heuristics is taken from Nielsen (1994c, p. 30, Table 2.2).

use of the system should be visible or easily retrievable whenever appropriate.

H7 Flexibility and efficiency of use: Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

H8 Aesthetic and minimalist design: Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

H9 Help users recognize, diagnose, and recover from errors: Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

H10 Help and documentation: Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

A first version of the set was suggested by Molich & Nielsen (1990), who conducted a survey with 77 participants who were asked to find as many usability problems as possible in an existing human-computer interface. The identified problems were classified according to a checklist of personal experience for good dialog design. The resulting nine problem categories are mentioned as the first set of heuristics for a heuristic evaluation (Nielsen & Molich, 1990). In his book on usability engineering, Nielsen (1993) describes the nine heuristics in more detail, and adds a tenth heuristic ("Help and documentation"). One year later, Nielsen (1994a) revises the set of usability heuristics, after having conducted a factor analysis of 249 usability problems[140]. The most recent version of Nielsen's usability heuristics can be found in Nielsen (1994c, p. 30) or as an online version[141].Heuristics not only facilitate the discovery of usability problems as they sensitize the evaluator for problematic aspects of an interface design, but also allow the evaluator to categorize identified usability problems to larger thematic areas, which in turn makes it easier to assess the overall usability of a system and prioritize actions to solve these problems[142].

Categorizing problems

Severity rating

Another means to prioritize usability problems and actions to respond to them is by ranking the *severity* of each problem. According to Nielsen (1994c, p. 47),

[140] Note: Although Nielsen (1994a) originally described a candidate set of nine revised heuristics, "Help and documentation" was added as a tenth heuristic in later references to that revised set of heuristics (cf. Nielsen, 1994c, p. 30).

[141] `http://www.nngroup.com/articles/ten-usability-heuristics`

[142] Example: 80% of all usability problems collected in an evaluation belong to the category "Help and documentation". The implication clearly is that this aspect of the system should be improved before other aspects.

the severity of a problem should be assessed by considering the *frequency* of the problem in the evaluated interface (common vs. rare), the overall *impact* of the problem, and the *persistence* of the problem (one-time problem vs. persisting problem). Nielsen (1994c, p. 49, Table 2.3) suggests a five-point scale for rating usability problems discovered through heuristic evaluation:

0 "I don't agree that this is a usability problem at all"

1 "Cosmetic problem only – need not be fixed unless extra time is available on project"

2 "Minor usability problem – fixing this should be given low priority"

3 "Major usability problem – important to fix, so should be given high priority"

4 "Usability catastrophe – imperative to fix this before product can be released"

Although heuristic evaluation "does not provide a systematic way to generate fixes to the usability problems" (Nielsen, 1994c, p. 31), Nielsen expects that it is fairly easy to find solutions for usability problems, once they are identified and categorized according to the different heuristics they violate.

Cognitive walkthrough – In contrast to the heuristic evaluation, the cognitive walkthrough is a structured evaluation method that relies on the systematic exploration ("incremental approach to learning"; cf. Wharton et al., 1994, p. 105) of a particular system design by an evaluator (typically a usability expert) who tries to put himself in the position of an actual user. Basically, the approach is a *review process* (Wharton et al., 1994, p. 106) where one or more aspects of a design are evaluated with regard to its usability from the user perspective. Wharton et al. (1994, p. 112) suggests four control questions the evaluator should keep asking himself throughout the whole evaluation process:

Control questions

Q1 "Will the users try to achieve the right effect?"

Q2 "Will the user notice that the correct action is available?"

Q3 "Will the user associate the correct action with the effect trying to be achieved?"

Q4 "If the correct action is performed, will the user see that progress is being made toward solution of the task?"

These questions are meant to increase the level of empathy with real users and also serve as basic usability guidelines for the assessment of the quality of a particular design.

Preliminaries

Before the cognitive walkthrough can be conducted, it is necessary to define some preliminary inputs for the evaluation (Wharton et al., 1994, p. 109ff.): First of all, the intended target audience needs to be described, to give the evaluator the necessary background information. Next, one or more typical tasks are defined together with the correct sequence of actions that is needed to accomplish that task. Finally, the interface of the tested tool has to be defined in a comprehensible way. The cognitive walkthrough is a rather structured and task-oriented method, where the evaluator tries to tell a convincing success or failure story for each single action that is taken to accomplish the overall task (Wharton et al., 1994, p. 114ff.). One of the limitations of the cognitive walkthrough method is that it only evaluates one specific aspect of usability, which is *ease of learning* (Wharton et al., 1994, p. 107).

Usability walkthrough – Usability walkthroughs[143] are a technique that was introduced by Karat et al. (1992). The method was developed as part of a comparative study of empirical testing and walkthrough methods, trying to increase the usability problem identification of walkthrough methods (Karat et al., 1992, p. 398-399). Similar to the heuristic evaluation, the evaluators use a list of 12 usability guidelines to identify existing usability problems (Karat et al., 1992, p. 399). A distinctive feature of the usability walkthrough, however, is the idea of a *two-pass process*: In the first pass, evaluators can freely explore the interface on their own, and in a second pass they are asked to inspect the interface again, while being guided by a list of scenarios (Karat et al., 1992; Sears, 1997).

Problems of existing methods

Sears (1997, p. 218-219) suggests the heuristic walkthrough as a method that combines the advantages of the inspection methods described above, and at the same time eliminates some of their specific problems. One issue of heuristic evaluation is its lack of structure, as the only guidance comes from a list of generic, unspecific heuristics which may be hard to apply for unexperienced evaluators (Sears, 1997, p. 219). Sears also notes the following problems: Unexperienced evaluators tend to focus on heuristics to rigidly and may only find problems that (they think) match one of the heuristics. At the same time, not every aspect of the interface that violates a heuristic is necessarily a usability problem, as it might be the result of a trade-off that had to be made to prevent even more usability problems. In general, heuristic evaluation evaluators are more prone to focus on the heuristics than on the interface elements. The cognitive walkthrough on the other hand runs the risk of being too structured, as it relies on a list of detailed user tasks and a set of four control questions (Sears, 1997, p. 219). Thus, the cognitive walkthrough is likely to discourage the discovery of usability problems that are not directly related to one of the predefined tasks or to the set of general questions (Sears, 1997, p. 219).

A hybrid approach

Heuristic walkthrough – The heuristic walkthrough borrows ideas from heuristic evaluation, cognitive walkthrough[144] and usability walkthroughs (cf. Figure 4.3) to make up for the specific problems of each single method (Sears, 1997, p. 219). Basically, the heuristic walkthrough is a two-pass (cf. usability walk-

[143] As Sears (1997, p. 221) notes that "usability walkthroughs never have been formally defined", the description of the method in this passage is rather brief.

[144] The cognitive walkthrough has not only inspired the heuristic walkthrough method, but rather has led to many other walkthrough variants. Mahatody et al. (2010) give an overview of the evolution of the cognitive walkthrough, and identify altogether eleven variants of the original cognitive walkthrough.

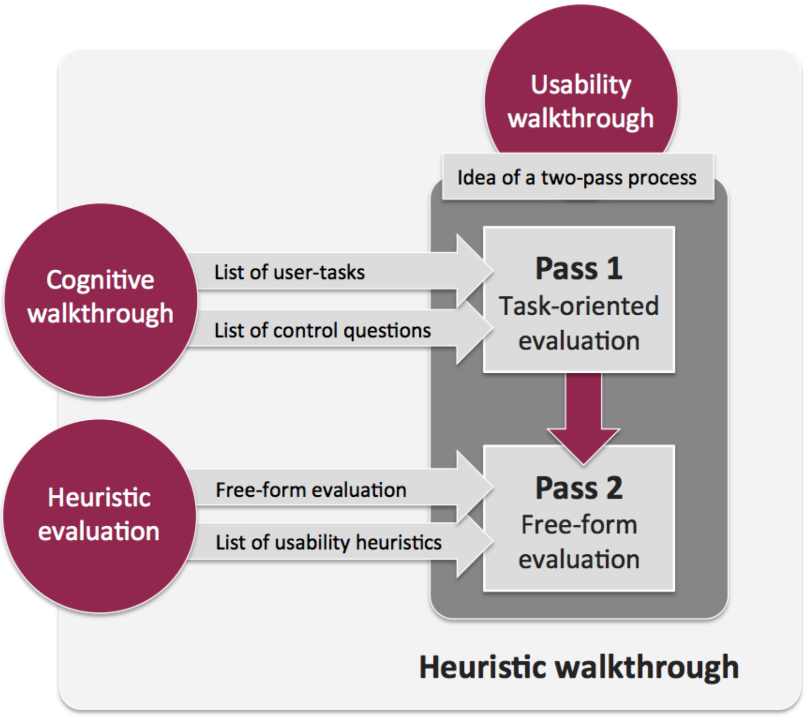


Figure 4.3.: Genesis of the heuristic walkthrough method according to Sears (1997, p. 219-221).

through) inspection method that can be described as follows (Sears, 1997, p. 219-221): In the first pass, the task-oriented evaluation, the evaluator explores the system by using a list of tasks that serves as a rough guide through the system. During the task-solving, the evaluator always keeps four *thought-focusing questions*[145] in mind. In the second pass (free-form evaluation) the evaluator can freely use and further explore the system. The evaluation of the system is based on the knowledge that was gained during the first pass and is further guided by any appropriate set of heuristics. Usability problems are documented (together with their individual severity ranking) during both passes (Sears, 1997, p. 219-220), i.e. whenever the evaluator happens to discover a problem he writes it down.

4.5.3. Insights from a pilot study

A preliminary pilot study by Burghardt (2012) with three evaluators[146] and three annotation tools (*GATE*, *MMAX2*, *UAM CorpusTool*) confirmed that the heuristic walkthrough method is an appropriate instrument for usability evaluations in

[145] The questions suggested by Sears are very similar to the original questions as proposed by Wharton et al. (1994, p. 112).

[146] The evaluators that were involved in this pilot study are Isabella Hastreiter, Florian Meier, and Manuel Burghardt.

this domain of application. Figure 4.4 shows the basic evaluation approach and the genesis of the results.

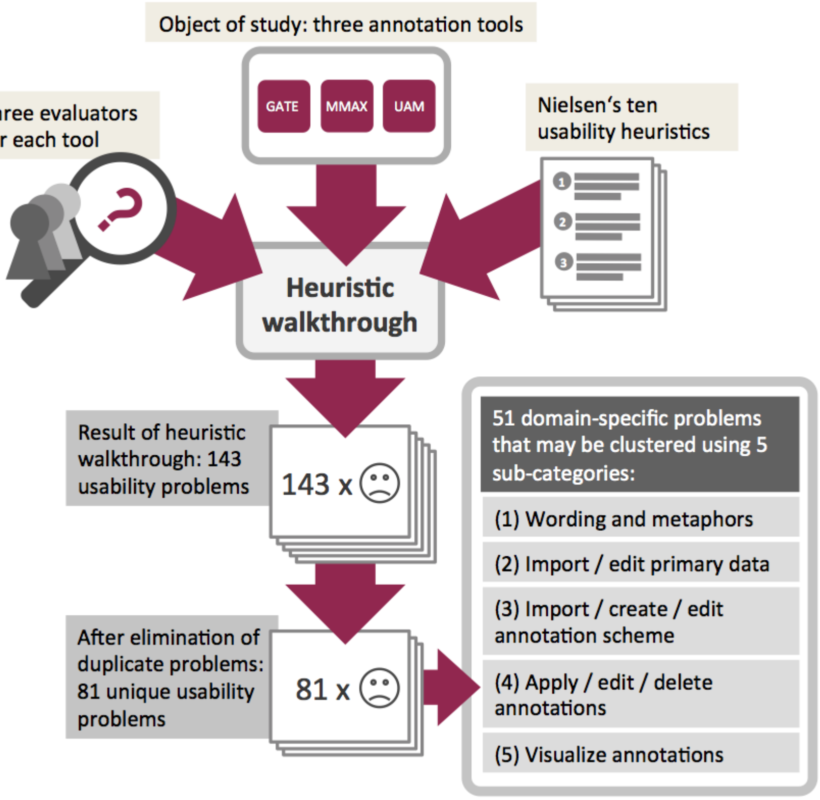

Object of study: three annotation tools
GATE
MMAX
UAM
Three evaluators for each tool
Nielsen's ten usability heuristics
1
2
3
Heuristic walkthrough
Result of heuristic walkthrough: 143 usability problems
143 x ☹
After elimination of duplicate problems: 81 unique usability problems
81 x ☹
51 domain-specific problems that may be clustered using 5 sub-categories:
(1) Wording and metaphors
(2) Import / edit primary data
(3) Import / create / edit annotation scheme
(4) Apply / edit / delete annotations
(5) Visualize annotations

Figure 4.4.: Evaluation approach and genesis of results in the pilot study on the usability of linguistic annotation tools.

Evaluation approach and results

The annotation tools in the pilot study were evaluated using Nielsen's 10 usability heuristics. Three evaluators were able to identify a total of 143 usability problems. After the elimination of *duplicate problems*, i.e. identical problems that were identified by two or more evaluators, a total of 81 unique usability problems remained (Burghardt, 2012, p. 109-110). A bottom-up clustering approach was used to derive higher level categories from the identified usability problems. It showed that 30 of the identified problems belong to the category of *general usability problems*, which means that the problems are not directly related to the domain of linguistic annotation tools but rather concern general aspects such as *user guidance*, *error messages*, *UI elements*, etc. The remaining 51 usability problems were classified as *domain-specific problems* that could be assigned to five sub-categories (cf. Figure 4.4). As a result of this pilot study, the identified problems and their implications for the design of annotation tool interfaces were formulated as 28 design recommendations for usable annotation tools. The recommendations as well as the study design were presented at the *6th Linguistic*

Annotation Workshop (peer reviewed), which took place during the *50th Annual Meeting of the Association for Computational Linguistics* in 2012.

Differences

The evaluation study described in this chapter differs from the pilot study as there is only one evaluator rather than three. At the same time, the number of annotation tools to be tested has been increased. The tasks have been revised to be more detailed and more comprehensive for the application domain – that is why the three tools that were already tested in the pilot study were tested again in the main evaluation study. Another difference between the two studies is that the main evaluation study does not only document usability problems, but also tool-specific strengths with regard to interface and interaction design.

4.5.4. Generation of tasks

Tasks

In order to help expose the evaluator to the most important parts of the system, a list of tasks is needed for the first pass of the heuristic walkthrough. This list should include the most frequent, most important tasks that real users will try to achieve with the evaluated system (Sears, 1997, p. 220). According to Sears, the evaluators should be able to choose from a list of predefined tasks. The number of predefined tasks as well as the order in which they are achieved can be determined by the respective evaluators (Sears, 1997, p. 220) – whenever an evaluator has the impression that he has explored the most important parts of the system, he can proceed to the second pass of the heuristic walkthrough, the free-form evaluation, where usability problems are identified by using a set of heuristics. Sears suggests to rank the tasks according to their priority (Sears, 1997, p. 220), to provide a basic decision guidance for the evaluators which tasks to choose, in order to experience the important system features most efficiently.

Modifications

For the intended heuristic walkthrough evaluation of annotation tools, some minor modifications will be applied to the task-based evaluation pass. These modifications are influenced by the following aspects: First, there are multiple systems that need to be evaluated (comparative evaluation), and second, there is only a single evaluator who also defines the tasks for the heuristic walkthrough. Accordingly, a comprehensive set of tasks that can be applied to most of the test objects is defined. The idea of a priority ranking of the tasks was abandoned, as the evaluator is meant to achieve *all* of the tasks with every tool (if possible) systematically anyway. Although this has been stated before it is worth to note again that for a heuristic walkthrough it is not important to define a comprehensive list of all possible tasks (which is hard when evaluating multiple systems), but rather to define tasks that support a thorough system exploration. The evaluator may still explore aspects of the system that have not been covered by the tasks in the second, free-form pass of the evaluation.

Task creation

The creation of tasks for the evaluation study builds on the usability requirements of annotation tools that have been described in section 4.4.2. All but two of the requirements are reflected in the evaluation tasks. Requirement "R02 – Integration of automatic annotation tools" is not explicitly modeled as a task for the heuristic walkthrough evaluation, as most of the tested tools support manual annotation only. In the few cases where this feature was available in a tool, the evaluator was aware of the requirement and explored the functionality in the free-form evaluation part. Requirement "R09 – Documentation" was not modeled as an explicit task, as scanning the manual or documentation before the actual start of the evaluation as well as looking up specific problems or obscurities during the evaluation process is part of the evaluation routine of a new software system anyway (also cf. section 4.5.6 for more details on the evaluation procedure).

The linguistic tasks are based on the taxonomies for *anchor scopes* and *linguistic levels* described in chapter 3.4.3. They are also inspired by the typical micro-tasks that have been identified for a generic annotation process (cf. section 4.4.3). Figure 4.5 shows a list of all evaluation tasks and subtasks that will be used to explore different linguistic annotation tools. Whenever possible, the table also relates each task to a specific *requirement*, *type of task*, *anchor scope* and *linguistic level*.

Task 1 is concerned with the installation and the setup of the tools. This preliminary task is included as part of the usability evaluation, as it might be a first technical hurdle for novice users.

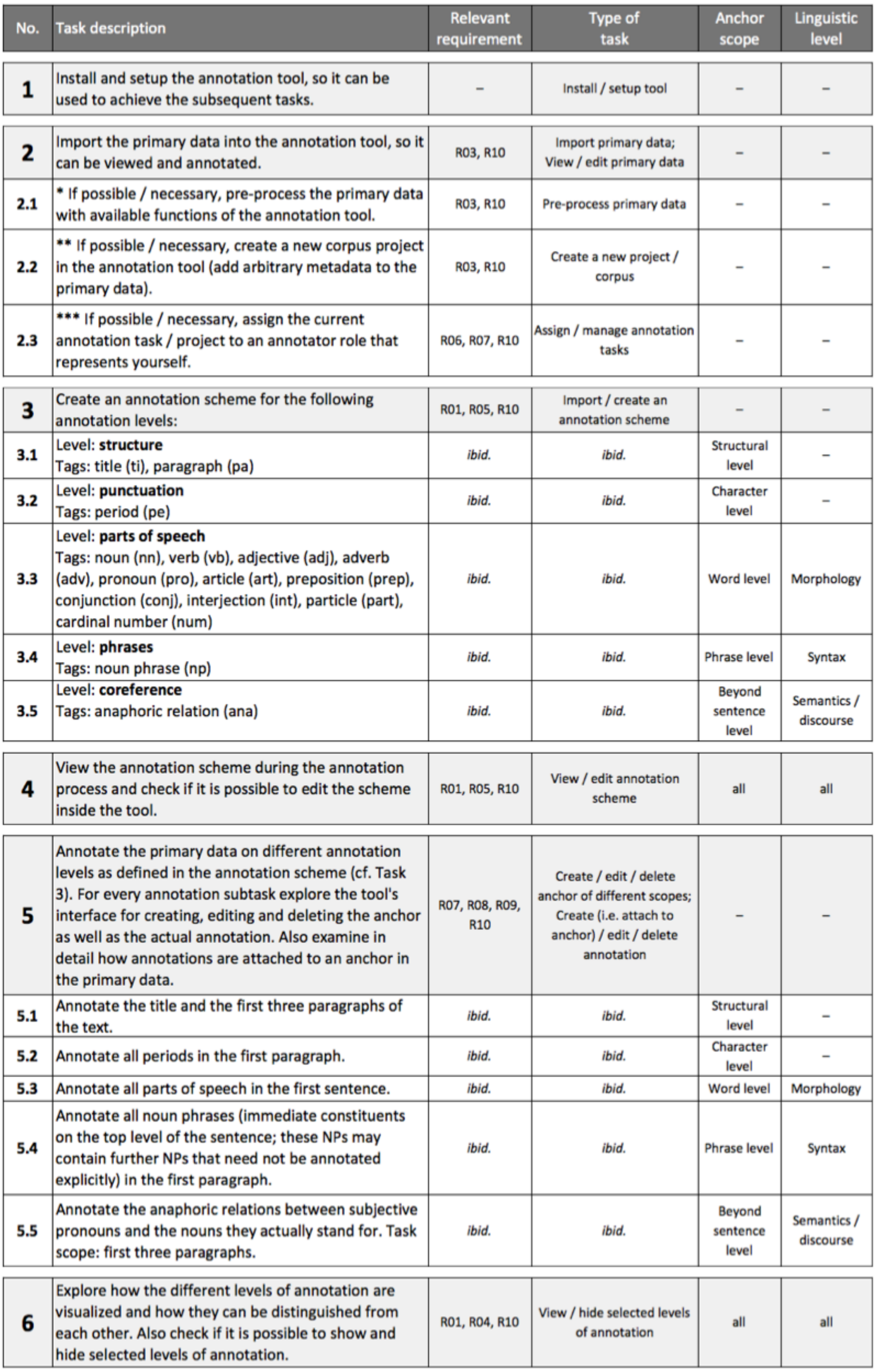
Task 2 covers the aspect of importing primary data into the annotation tool, so it can be annotated during the subsequent tasks. Related subtasks – which may differ from tool to tool – may include the pre-processing of the data (e.g. tokenization, sentence splitting, etc.), the creation of a new corpus project, and the assignment of the current annotation tasks to a specific annotator (basic corpus project management).

Task 3 asks to create an annotation scheme with altogether five different levels of annotation. The levels were mainly adopted from the previously described human handwritten annotation study (cf. chapter 2.8.2), as they cover most of the anchor scopes[147] and some exemplary linguistic levels, such as morphology, syntax, semantics and discourse.

Task 4 is closely related to the previous task, as it makes the evaluator check if the annotation scheme can be viewed and edited from within the annotation tool.

Task 5 guides the evaluator to set into practice different linguistic annotation sub-tasks by using tags from the previously defined annotation scheme. The different annotation tasks ensure that tags from every annotation level

[147] The anchor scope *text level*, which describes information about the text as a whole, is typically realized as a metadata dialogue by most annotation tools, and therefore is not addressed by the annotation scheme creation subtasks.



No.	Task description	Relevant requirement	Type of task	Anchor scope	Linguistic level
1	Install and setup the annotation tool, so it can be used to achieve the subsequent tasks.	–	Install / setup tool	–	–
2	Import the primary data into the annotation tool, so it can be viewed and annotated.	R03, R10	Import primary data; View / edit primary data	–	–
2.1	* If possible / necessary, pre-process the primary data with available functions of the annotation tool.	R03, R10	Pre-process primary data	–	–
2.2	** If possible / necessary, create a new corpus project in the annotation tool (add arbitrary metadata to the primary data).	R03, R10	Create a new project / corpus	–	–
2.3	*** If possible / necessary, assign the current annotation task / project to an annotator role that represents yourself.	R06, R07, R10	Assign / manage annotation tasks	–	–
3	Create an annotation scheme for the following annotation levels:	R01, R05, R10	Import / create an annotation scheme	–	–
3.1	Level: **structure** Tags: title (ti), paragraph (pa)	*ibid.*	*ibid.*	Structural level	–
3.2	Level: **punctuation** Tags: period (pe)	*ibid.*	*ibid.*	Character level	–
3.3	Level: **parts of speech** Tags: noun (nn), verb (vb), adjective (adj), adverb (adv), pronoun (pro), article (art), preposition (prep), conjunction (conj), interjection (int), particle (part), cardinal number (num)	*ibid.*	*ibid.*	Word level	Morphology
3.4	Level: **phrases** Tags: noun phrase (np)	*ibid.*	*ibid.*	Phrase level	Syntax
3.5	Level: **coreference** Tags: anaphoric relation (ana)	*ibid.*	*ibid.*	Beyond sentence level	Semantics / discourse
4	View the annotation scheme during the annotation process and check if it is possible to edit the scheme inside the tool.	R01, R05, R10	View / edit annotation scheme	all	all
5	Annotate the primary data on different annotation levels as defined in the annotation scheme (cf. Task 3). For every annotation subtask explore the tool's interface for creating, editing and deleting the anchor as well as the actual annotation. Also examine in detail how annotations are attached to an anchor in the primary data.	R07, R08, R09, R10	Create / edit / delete anchor of different scopes; Create (i.e. attach to anchor) / edit / delete annotation	–	–
5.1	Annotate the title and the first three paragraphs of the text.	*ibid.*	*ibid.*	Structural level	–
5.2	Annotate all periods in the first paragraph.	*ibid.*	*ibid.*	Character level	–
5.3	Annotate all parts of speech in the first sentence.	*ibid.*	*ibid.*	Word level	Morphology
5.4	Annotate all noun phrases (immediate constituents on the top level of the sentence; these NPs may contain further NPs that need not be annotated explicitly) in the first paragraph.	*ibid.*	*ibid.*	Phrase level	Syntax
5.5	Annotate the anaphoric relations between subjective pronouns and the nouns they actually stand for. Task scope: first three paragraphs.	*ibid.*	*ibid.*	Beyond sentence level	Semantics / discourse
6	Explore how the different levels of annotation are visualized and how they can be distinguished from each other. Also check if it is possible to show and hide selected levels of annotation.	R01, R04, R10	View / hide selected levels of annotation	all	all

Figure 4.5.: Tasks for the heuristic walkthrough evaluation of linguistic annotation tools.

are used, and that some portions of the primary text are annotated on parallel levels. As the annotation tasks will typically recur with a high frequency, it is important for the evaluator to examine the interface for creating and modifying anchors and for attaching annotations from the predefined scheme to these anchors. Special attention needs to be paid on how the creation of coreference annotations is realized.

Task 6 is meant to make the evaluator explore how multiple, parallel levels of annotations are visualized in the primary data, and to check if it is possible to show and hide selected levels of annotation. The adequate visualization of multiple, parallel annotations is supposedly one of the biggest challenges an annotation tool has to meet.

During the task-based evaluation pass, first impressions and general observations, with regard to the respective tasks, will be documented as informal notes. These notes will be refined into concrete usability problems and individual strengths of a tool in the second pass, the free-form, heuristic evaluation.

4.5.5. Selection of test objects

The previous section has introduced three general criteria to categorize linguistic annotation tools. These criteria were used to identify appropriate *test objects* for the usability evaluation. As the focus of this work is on tools for manual annotation of text documents, *appropriateness* in terms of these criteria can be defined as follows:

Criterion 1: Modality of annotation Only annotation tools for textual data were selected, i.e. tools for transcription or for audio, video or image annotation were not considered as test objects.

Exemplary tools to be excluded from the evaluation: *Anvil*, *ELAN*, *EXMARaLDA*, *Multitool*[148], *Pacx*[149], *Praat*, *Sign Stream*[150] *The Observer XT*[151], *Transcriber AG*[152], *WaveSurfer*[153], *Xtrans*[154]

Criterion 2: Software type Only ready-to-use tools with a graphical user interface (GUI) were selected, no APIs or programming toolkits.

Exemplary tools to be excluded from the evaluation: *AGTK*, *ATLAS* (Archi-

[148] http://www.ling.gu.se/projekt/tal/multitool/index.htm
[149] http://pacx.sourceforge.net/
[150] http://www.bu.edu/asllrp/SignStream/
[151] http://www.noldus.com/human-behavior-research/products/the-observer-xt
[152] http://transag.sourceforge.net/
[153] http://sourceforge.net/projects/wavesurfer/
[154] https://www.ldc.upenn.edu/language-resources/tools/xtrans

tecture and Tools for Linguistic Analysis Systems)[155], *NITE XML Toolkit*[156], *Snack Sound Toolkit*[157]

Criterion 3: Task scope Only tools able to achieve Tasks 1-5 were selected. Task 6 (coreference annotation) was treated as an optional tasks.

Exemplary tools to be excluded from the evaluation: *Annotate* (syntactic annotation only)[158], *RSTTool* (rhetorical structure annotation only) [159], *Serengeti* (semantic relations only)[160], *Synpathy* (syntactic annotation only)[161]

After the application of these three criteria, a total of twenty-one tools was gathered as potential test objects. While trying to download and set up these tools, it became obvious that some additional selection criteria were needed:

Criterion 4: Availability A number of tools that is described in the literature or on one of the overview websites on linguistic annotation tools were no longer available[162]. This is an indicator for the short lifespan of an average annotation tool that could be interpreted as an additional argument for a lack of usability in many of the existing tools.

Examples for tools that are no longer available on the web include: *Alembic Workbench*, *Callisto*, *CLaRK*, *LDC ACE Annotation Toolkit*, *PALinkA*

Criterion 5: Feasibility of installation / setup Some of the tools were quite cumbersome to install and set up. It was not possible to set up the following tools on the test environment that was used during the evaluation: *Anafora*[163] *AnChoraPipe*[164], *Djangology Web Annotator*[165], *Slate*[166]

After applying the above selection criteria, a total of eleven tools remained as test objects for the evaluation study. Although a thorough review of available

[155] `http://jatlas.sourceforge.net/`

[156] `http://groups.inf.ed.ac.uk/nxt/index.shtml`

[157] `http://www.speech.kth.se/snack/`; *Snack* is a tool kit for the creation of audio applications (spectrum analysis, waveforms, etc.).

[158] `http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/annotate.html`

[159] `http://www.wagsoft.com/RSTTool/`

[160] `http://coli.lili.uni-bielefeld.de/serengeti/`

[161] `http://tla.mpi.nl/tools/tla-tools/older-tools/synpathy/synpathy-description-download/`

[162] The evaluation was conducted from October 2, 2013 – December 9, 2013, i.e. the tools could not be retrieved on the web during this period.

[163] `https://github.com/weitechen/anafora`; Anafora is also available as a static, not configurable web demo (cf. `https://verbs.colorado.edu/anaforademo/annotate/Demo/samplenotes/doc22/Medicine/`). The web demo was excluded from the evaluation study, as it was not able to achieve the predefined tasks (cf. *criterium 2 – task scope*).

[164] `http://clic.ub.edu/mbertran/tbfeditor/`; the tool was relatively easy to install. It is, however, not clear how to set it up, in order to use it as an annotation tool. Unfortunately, the manual is only available in Catalan language and therefore was not of much help.

[165] `http://djangology.sourceforge.net/`

[166] `https://www.cl.cs.titech.ac.jp/slate/`

overview pages (*DiRT wiki*, *TAPOR portal*, *Linguistic Annotation Wiki*) and respective literature in the field of linguistic annotation was conducted, the list of annotation tools that meets the above criteria is not meant to be comprehensive, but rather tries to provide a snapshot of available tools from October 2, 2013 – December 9, 2013. Table 4.5 shows the eleven annotation tools (in the order of evaluation) that were selected as test objects for the evaluation study. The table contains the name of the tools, the download source, and the date of download and evaluation.

Name	**Source**	**Download**	**Evaluation**
Dexter	`http://www.dextercoder.org/`	October 2, 2013	October 2, 2013
CATMA	`http://www.catma.de/`	October 7, 2013	October 7, 2013
Glozz	`http://www.glozz.org/`	October 8, 2013	October 10, 2013
UAM Corpus Tool	`http://www.wagsoft.com/CorpusTool/index.html`	October 11, 2013	October 11, 2013
Brat	`http://brat.nlplab.org/`	October 15, 2013	October 15, 2013
MMAX2	`http://mmax2.sourceforge.net/`	November 12, 2013	November 12, 2013; November 17, 2013
WordFreak	`http://wordfreak.sourceforge.net/index.html`	November 22, 2013	November 22, 2013
Analec	`http://www.lattice.cnrs.fr/Telecharger-Analec?lang=fr`	November 24, 2013	November 24, 2013
WebAnno	`https://code.google.com/p/webanno/`	December 3, 2013	December 3, 2013
Knowtator	`http://knowtator.sourceforge.net/docs.shtml`	December 6, 2013	December 6, 2013; December 8, 2013
GATE	`https://gate.ac.uk/`	December 9, 2013	December 9, 2013

Table 4.5.: Tools that were selected as test objects for the usability evaluation. The order of the tools also reflects the order of evaluation.

4.5.6. Evaluation procedure

Preliminaries

The heuristic walkthrough evaluation of eleven linguistic annotation tools was conducted by one double expert evaluator, who is also the author of this work. Except for *Brat*, all tools were tested on a PC with Windows 7 and the Java runtime environment (Java 7). *Brat* was evaluated on a MacBook with OS X (Mountain Lion), as it required a UNIX like environment in order to be set up properly. The first pass of the heuristic walkthrough (task solving) was recorded with the free screen recording tool CamStudio[167]. The evaluation on the MacBook was recorded with the default screen recording tool that is part of OS X. All eleven tools were tested in randomized order between October 2, 2013 – December 9, 2013. In most cases, the whole evaluation was carried out on the same day. In two cases, the tool was installed (Task 1) on one day, while Tasks 2-6 were achieved a couple of days later (cf. Table 4.5).

Evaluation process

The basic evaluation process for each tool can be described as follows (cf. appendix A for a detailed list of all steps): The first step is to obtain the actual tool, which means to download the files to the test environment. Before starting with the task-oriented evaluation pass, the tool manual, information on the tool website, and existing publications are consulted, to get a rough overview of the tool. Next, the first pass of the evaluation is conducted, which means to subsequently achieve the six predefined tasks. Each task is recorded with a screen recording software. In addition to the screen recording, screenshots of interesting tool features are made. The basic steps that were necessary to achieve a task are all written down, together with a short roundup of the respective tool (cf. appendix C). Usability problems identified during the first pass of evaluation are loosely written down with pen and paper. During the second pass, while the tool is freely explored with Nielsen's heuristics in mind, usability problems and individual strengths are documented in a more systematic manner, by means of a structured Excel spreadsheet.

Scope of usability problems

As the previous pilot study has shown, there are two main classes of usability problems: *general* usability problems and *domain-specific* problems. There is a great number of guidelines (cf. Johnson, 2007) and patterns (cf. Tidwell, 2011) that help the designer to avoid general problems, like for instance "use of color" and "use of fonts", but also provide hints on how to design user-friendly UI elements such as "menu structures", "buttons", etc. The focus of this evaluation study is on usability problems that are characteristic for the domain of linguistic annotation tools, i.e. general usability problems will not be documented. It is not always easy to differentiate the two problem categories, as sometimes a general problem gets mixed up with a domain-specific feature, for instance: bad color scheme (general) for the visualization of multiple annotation items (domain-specific). General problems that have no direct impact on domain-specific interface characteristics will be neglected in this evaluation study.

Test materials

Throughout the evaluation, Nielsen's ten usability heuristics, Sears' four thought-focusing questions, and a basic description of the main user groups (annotators and traditionalist, cf. subsection 4.4.1) of annotation tools were visible as large printed posters. Also, a printed version of the evaluation tasks was available during the tests.

Documentation of problems

As Sears (1997) does not give a detailed description on how to document usability problems during a heuristic walkthrough, the approach from heuristic

[167] `http://camstudio.org/`

evaluation will be adopted, which means to document each problem – together with the violated heuristic(s) and the severity of the problem – with enough contextual information to be able to comprehend it in the future (cf. Sears, 1997, p. 217). When formulating the problems, the following best practices as described by Dumas et al. (2004) will be adhered:

- Emphasize the positive
- Express your annoyance tactfully
- Avoid usability jargon
- Be as specific as you can

The identified usability problems are documented by means of an Excel spreadsheet that contains metadata about the test object, such as *tool name*, *source of download*, *date of download* and *date of evaluation*. Each usability problem is written down with a unique *ID*, a short problem description that acts as the *title of the problem*, and a more detailed *problem description*. The spreadsheet also documents which *heuristics* were violated while the problem occurred, and how the *severity* of the problem was ranked (cf. section 4.5.2 for a detailed description of the heuristics as well as the severity ranking metrics). Finally, the five subcategories for domain-specific usability problems that were identified during the pilot study are used to classify and organize the problems that are revealed during the heuristic walkthrough. The categories were rephrased to be more comprehensive; furthermore, a sixth problem category "Installation", a task that was not evaluated in the preceding pilot study, was added (cf. Table 4.6).

Pilot study	**Rephrased version**
–	Installation
Wording and metaphors	General UI
Import / edit primary data	Primary data
Import / create / edit annotation scheme	Annotation scheme
Apply / edit / delete annotations	Annotation process
Visualize annotation	Annotation visualization

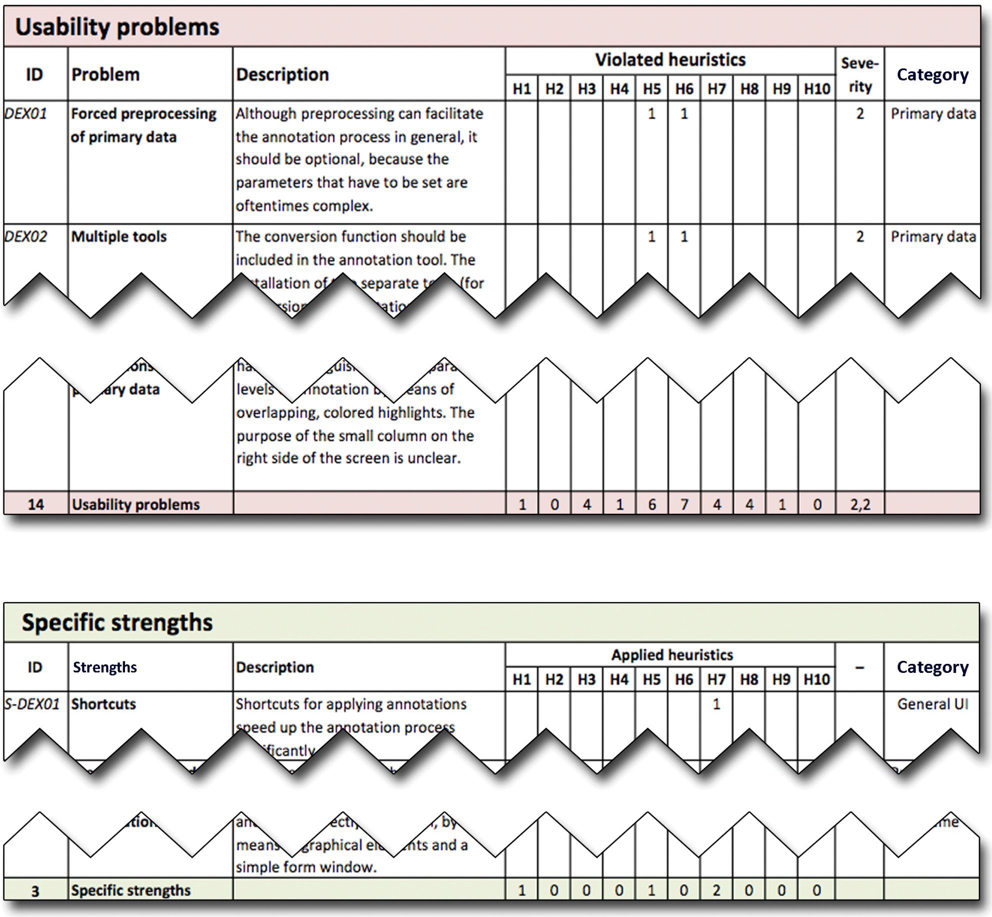
Table 4.6.: Original categories (pilot study) and rephrased versions used in the series of heuristic walkthroughs.

Documentation of strengths

As several competing tools are analyzed in the course of this heuristic walkthrough evaluation, it is very likely that the solution to usability problems identified for one tool is present in another tool. For this reason, not only usability problems are documented, but also good designs and potential solutions for previously identified problems. Such specific strengths of a tool are written down with the same information as the usability problems (ID, title, description, etc.)[168] in the "strengths" section of the spreadsheet. Like the problems,

[168] Obviously, the *severity* criterion does not apply for a tool's strengths, and is therefore not

particular strengths of an annotation tool are organized according to the domain-specific categories. This consistent categorization scheme will greatly facilitate the posterior *identification* (cf. chapter 5.6.3) and *organization* of patterns (cf. chapter 6.2) Figure 4.6 shows an excerpt of the spreadsheet to illustrate what the documentation for usability problems and positive design solutions looks like.



Usability problems

ID	Problem	Description	Violated heuristics										Seve-rity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
DEX01	**Forced preprocessing of primary data**	Although preprocessing can facilitate the annotation process in general, it should be optional, because the parameters that have to be set are oftentimes complex.					1	1					2	Primary data
DEX02	**Multiple tools**	The conversion function should be included in the annotation tool. The ...tallation of ... separate t... (for ...sion ...atio					1	1					2	Primary data
	...ons... p...ary data	ha... ...uis... ...para... levels ...nnotation b... eans of overlapping, colored highlights. The purpose of the small column on the right side of the screen is unclear.												
14	**Usability problems**		1	0	4	1	6	7	4	4	1	0	2,2	

Specific strengths

ID	Strengths	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-DEX01	**Shortcuts**	Shortcuts for applying annotations speed up the annotation process ...ificantly							1					General UI
	...tion...	an... ...ecti... ..., by means ...graphical el... ...nts and a simple form window.												...me
3	**Specific strengths**		1	0	0	0	1	0	2	0	0	0		

Figure 4.6.: Excerpt of the spreadsheet used for documenting usability-related observations during the evaluation.

Final steps

Toward the end of the evaluation, the number of usability problems / specific strengths, the average severity of the identified problems, and the number of violated heuristics[169] is counted. Finally, a feature matrix that captures basic characteristics of a tool is filled in. All files and documents used and created during the annotation (screen recordings, screenshots, installation files, test data, man-

documented in the spreadsheet.

[169] For the specific strengths, the number of successfully applied heuristics was counted.

ual, publications, spreadsheets) are archived. The next section summarizes the quantitative results of the evaluation study and illustrates how the data can be interpreted in a way to derive usability patterns for linguistic annotation tools.

4.6. Results of the heuristic walkthrough evaluation

This section provides a comparison of the evaluated tools according to some basic characteristics that could be explored during the heuristic walkthroughs. In addition, the evaluation results are discussed in a quantitative manner by addressing aspects such as:

- Total number of usability problems / strengths
- Average severity of usability problems
- Number of usability problems / strengths per tool
- Number of usability problems / strengths per domain-specific category
- Number of total violations / successful applications for each of the usability heuristics

4.6.1. Tool overview

Architecture

At the end of each heuristic walkthrough, a feature matrix (cf. Figure 4.7) was filled in to document some common characteristics of annotation tools[170]. The matrix shows that most of the evaluated tools are available as desktop tools that need to be installed and setup on a local system. There are, however, some examples that are either readily available as a web service (cf. *CATMA*), or that can be setup as a web service on a private server (cf. *Brat* and *WebAnno*).

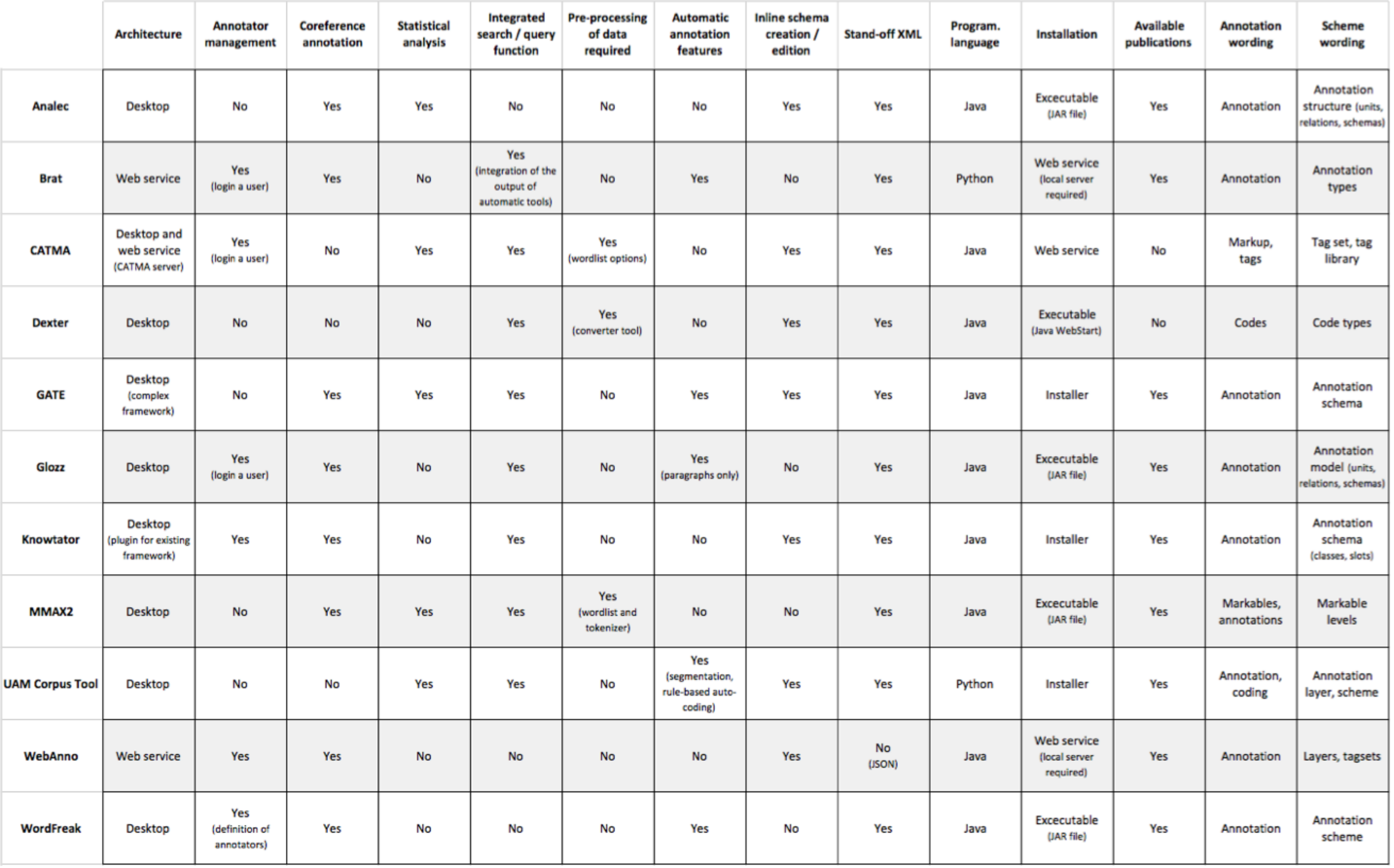
Annotator management

The management of annotators (roles and rights management) is mostly available for web service tools (cf. *CATMA*, etc.), as this type of architecture already requires users to authenticate by means of a login. Although it is a desktop tool, *Glozz* allows the user to register on the startup of the tool. *WordFreak* provides a function to define annotators that may be included to or excluded from an annotation project. *Knowtator* allows the user to define and manage annotators, including a function to calculate *inter annotator agreement*. *WebAnno* provides the most sophisticated functions to manage multiple annotators, as it allows the user to monitor the progress of different annotators on different texts or to compare annotations from different annotators in a special *curation* mode.

Coreference

Coreference annotation is a specific kind of annotation task, as it requires to establish a connection between two or more anchors. While some tools do not

[170] A detailed description of the tools with respect to the evaluation tasks can be found in appendix C.



	Architecture	Annotator management	Coreference annotation	Statistical analysis	Integrated search / query function	Pre-processing of data required	Automatic annotation features	Inline schema creation / edition	Stand-off XML	Program. language	Installation	Available publications	Annotation wording	Scheme wording
Analec	Desktop	No	Yes	Yes	No	No	No	Yes	Yes	Java	Excecutable (JAR file)	Yes	Annotation	Annotation structure (units, relations, schemas)
Brat	Web service	Yes (login a user)	Yes	No	Yes (integration of the output of automatic tools)	No	Yes	No	Yes	Python	Web service (local server required)	Yes	Annotation	Annotation types
CATMA	Desktop and web service (CATMA server)	Yes (login a user)	No	Yes	Yes	Yes (wordlist options)	No	Yes	Yes	Java	Web service	No	Markup, tags	Tag set, tag library
Dexter	Desktop	No	No	No	Yes	Yes (converter tool)	No	Yes	Yes	Java	Executable (Java WebStart)	No	Codes	Code types
GATE	Desktop (complex framework)	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Java	Installer	Yes	Annotation	Annotation schema
Glozz	Desktop	Yes (login a user)	Yes	No	Yes	No	Yes (paragraphs only)	No	Yes	Java	Excecutable (JAR file)	Yes	Annotation	Annotation model (units, relations, schemas)
Knowtator	Desktop (plugin for existing framework)	Yes	Yes	No	Yes	No	No	Yes	Yes	Java	Installer	Yes	Annotation	Annotation schema (classes, slots)
MMAX2	Desktop	No	Yes	Yes	Yes	Yes (wordlist and tokenizer)	No	No	Yes	Java	Excecutable (JAR file)	Yes	Markables, annotations	Markable levels
UAM Corpus Tool	Desktop	No	No	Yes	Yes	No	Yes (segmentation, rule-based auto-coding)	Yes	Yes	Python	Installer	Yes	Annotation, coding	Annotation layer, scheme
WebAnno	Web service	Yes	Yes	No	No	No	No	Yes	No (JSON)	Java	Web service (local server required)	Yes	Annotation	Layers, tagsets
WordFreak	Desktop	Yes (definition of annotators)	Yes	No	No	No	Yes	No	Yes	Java	Excecutable (JAR file)	Yes	Annotation	Annotation scheme

Figure 4.7.: Comparison of evaluated tools according to selected features.

support this type of annotation at all (cf. *CATMA*, *Dexter* and *UAM Corpus Tool*), other tools seem to focus on the annotation of coreference and other relational annotations (cf. *Brat*, *MMAX2* and *WebAnno*), as they provide advanced visualizations in the form of arcs and arrows inside the annotated primary data.

Text analysis

Many tools are not only annotation tools, but also allow the user to perform statistical analysis on the data inside the tool. Some tools even provide a sophisticated query mechanism (cf. e.g. *MMAX2* and *Knowtator*) to analyze the annotated data.

Pre-processing

Another characteristic of annotation tools is the need for pre-processed data: some tools cannot be used on plain text immediately, but rather require the user to conduct basic preprocessing of the data inside the tool (cf. *Dexter* and *MMAX2*).

Automatic annotation

Closely related is the characteristic of an integrated, automatic annotation feature, to allow for a mixed manual-automatic approach. Only few tools provide functions that enable the user to automate parts of the otherwise manual annotation process (cf. *Brat*, *GATE*, *Glozz*, *UAM Corpus Tool*, *WordFreak*).

Annotation scheme

The creation of an annotation scheme is another feature that can be used to characterize annotation tools: While some tools provide a graphical user interface to create and edit annotation schemes (cf. *Analec*, *CATMA*, *Dexter*, *GATE*, *Glozz*, *UAM Corpus Tool*, *WebAnno*), most of the other tools require the user to create an annotation scheme by means of XML markup.

Stand-off annotation

In terms of the storage format for annotated data, stand-off XML seems to be the de facto standard that is adhered by most of the tools (also cf. Burghardt & Wolff, 2009a). Only *WebAnno* seems to store and exchange data in the structured JSON format (attribute-value pairs) or respectively in a MySQL relational database.

Installation

Most tools are written in *Java*, which means that they can be installed and setup very easily (*Java Runtime Environment* required) on a local system by just executing a JAR-file or a windows installer. Two tools are written in *Python* (cf. *Brat* and *UAM Corpus Tool*), which is a common programming language in the field of quantitative linguistics, as it provides many practical functions and programming libraries that can be used for the manipulation and analysis of textual data. In case of the web-based tools it is necessary to set up a local server (cf. *Brat* and *WebAnno*). Both tools can, however, be downloaded as an all-in-one-version that already includes a stand-alone server.

Publications

Many of the tools have been promoted in the corpus linguistics community by means of publications at conferences such as the *Association for Computational Linguistics* (ACL) or the *International Conference on Language Resources and Evaluation* (LREC).

Wording

The last feature addresses the wording of different tools for concepts such as "annotation" and "annotation scheme". It shows that most tools actually use the term "annotation" to describe the act of adding linguistic information to primary data. *CATMA* uses the terms "markup" and "tags", while *UAM Corpus Tool* and *Dexter* use the terms "code" and "coding". *MMAX2* uses the term "markable" for an anchor in the primary data. In case of the "annotation scheme", there is an even greater diversity of terms: "Annotation structure" (cf. *Analec*), "Annotation types" (cf. *Brat*), "Tag set" (cf. *CATMA*, *WebAnno*), "Code types" (cf. *Dexter*), "Annotation schema" (cf. *GATE*, *Knowtator*, *UAM Corpus Tool*, *WordFreak*), "Annotation model" (cf. *Glozz*) and "Markable levels" (cf. *MMAX2*) are common examples for different terms that describe the same concept.

4.6.2. Quantitative results: Usability problems

Usability problem count

The quantitative results described in this subsection are mainly used to get hints on larger problem areas and to facilitate the creation of generic design patterns. As a result of the heuristic walkthrough evaluation of eleven linguistic annotation tools, a total of 207 usability problems[171] with an average severity of 2.7 could be identified (cf. Figure 4.8). At this stage of analysis, many of the identified problems may be redundant, i.e. they may be observed for several tools. The problems will be systematized to form clusters of unique problems in the following chapter, where they are used as input for the formulation of usability patterns for annotation tools. The average problem-per-tool count is slightly below 19, where the fewest problems observed for a tool made up for a total of

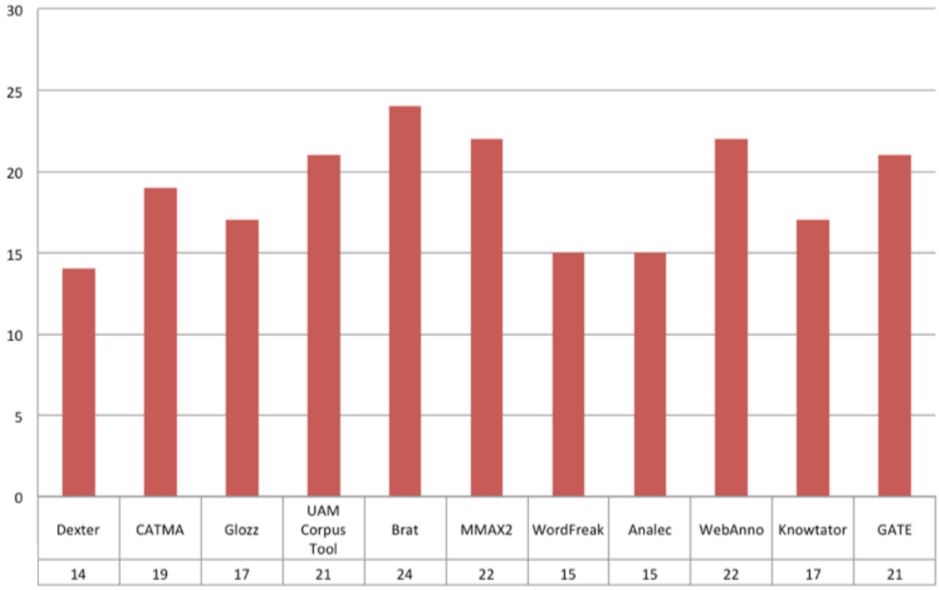


Figure 4.8.: Usability problems per tool.

[171] Note that only domain-specific problems were documented.

14 problems, and the most problems are 24. These numbers are, however, not meant to deduce any kind of tool ranking, as the problems have different severity ratings. The heterogeneous problem counts for each tool along the timeline of the evaluation, however, show that there is no specific tendency to identify more or less problems during a series of heuristic walkthrough evaluations. The results also indicate that none of the existing annotation tools that were evaluated in this study does achieve a high level of usability with the current interface design, but that all struggle from a number of more or less severe usability problems. This observation supports the approach to create generic usability patterns for the design of user-friendly annotation tools.

Categories

As was described before, all identified usability problems were assigned to one or more domain-specific categories during the evaluation process. Figure 4.9 gives an overview of the distribution of usability problems across the six categories.

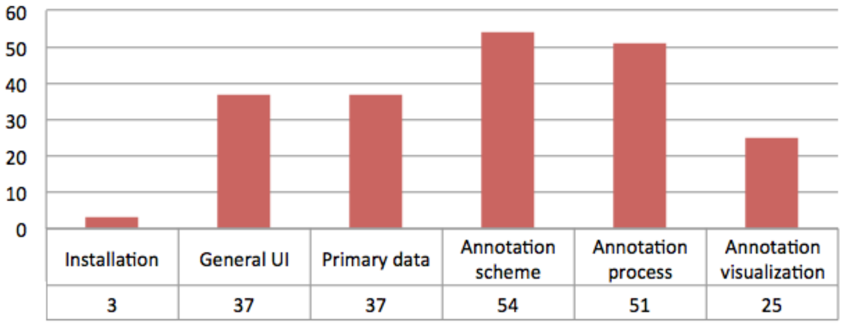


Figure 4.9.: Usability problems per domain-specific category.

It shows that only few problems occurred during the setup and installation task. This is not too surprising, as most of the tools are installed by means of executing an existing JAR-file or an installer file. Also, the tools that caused the most serious issues during the installation were already sorted out as they could not be installed at all (cf. Criterion 5: *Feasibility of installation / setup*; section 4.5.5). The *annotation process* seems to be the most problematic area, which is also plausible, as it includes the most potential human-computer interactions.

Violated heuristics

Finally, it is also possible to quantify the number of violated heuristics, to get a hint toward areas that generally tend to be more or less problematic in annotation tool interfaces (cf. Figure 4.10). It is important to note that one usability problem could violate several heuristics at the same time – therefore, a total number of 408 violations[172] of Nielsen's usability heuristics could be observed. The heuristics that were violated only in few cases are H9 ("Help users recognize, diagnose and recover from errors") and H10 ("Help and documentation).

[172] Note: The average usability problem violates approx. 2 heuristics.

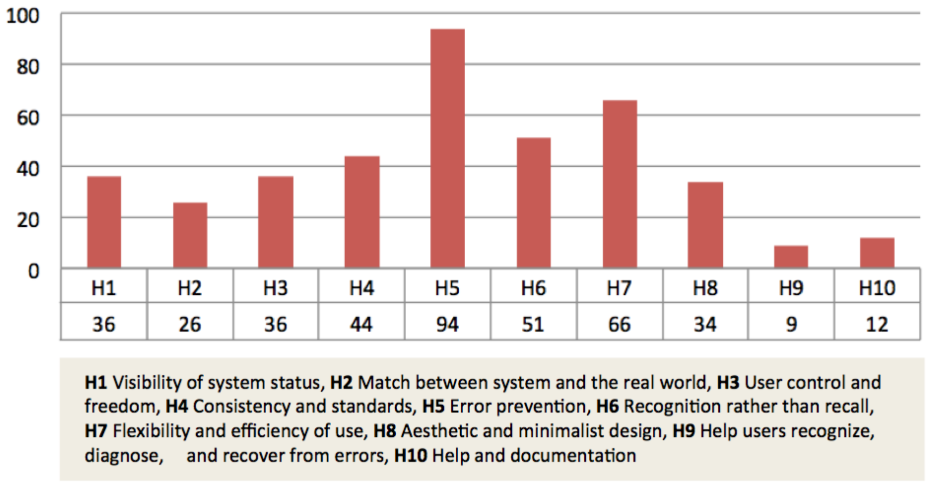


Figure 4.10.: Number of violated heuristics during the series of heuristic walkthroughs for eleven annotation tools.

The most violated heuristic is H5 ("Error prevention"), which indicates that current annotation tool interfaces are more prone to make users produce errors and unintentional effects rather than preventing errors.

4.6.3. Quantitative results: Strengths

Similar to the usability problems, specific strengths of the different tools were documented. A total of 84 strengths was identified and documented for the eleven evaluated annotation tools (cf. Figure 4.11).

Strength count

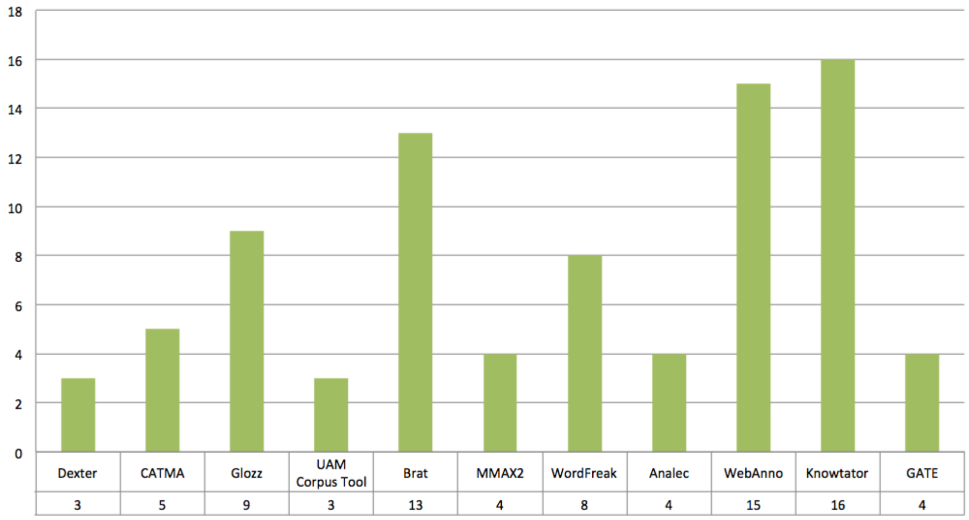


Figure 4.11.: Strengths per tool.

The average number of strengths per tool is around 7.6, with the lowest strengths-per-tool count being at 3, and the highest being at 16. Modern tools such as *Brat*, *WebAnno* and *Knowtator* implement some promising ideas for an improved annotation tool interface. At the same time it is important to note that despite these numerous strengths, the mentioned tools also suffer from a number of usability problems, i.e. they implement many positive but also many negative interface aspects.

Categories

Figure 4.12 shows the identified strengths distributed among the different domain-specific problem areas. It is important to note that at this point of the analysis it is not possible to relate the strengths to concrete weaknesses, i.e. it may be possible that there are various strengths that can be used for one identified usability problem. At the same time, it is possible that the strengths are not even concrete answers to usability problems, but rather give hints on the general design of annotation tool interfaces. A concrete relation between problems and strengths will be established in the next chapter on the identification of usability patterns.

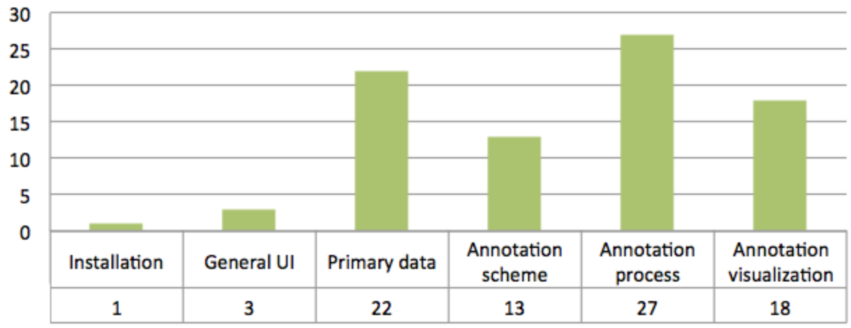


Figure 4.12.: Strengths per domain-specific category.

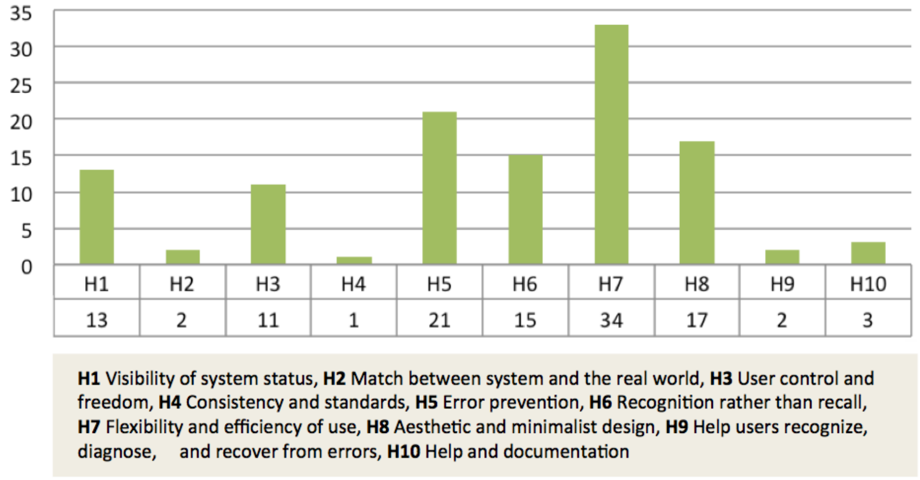
Successfully applied heuristics

Heuristics may not only be used as an instrument to systematically reveal usability problems, but can also be used as positive design guidelines. By looking at heuristics from this perspective, it is possible to document the usability heuristics that were successfully applied by a positive design solution. The results in Figure 4.13 indicate that many identified strengths seem to improve an annotation tool by means of "Flexibility and efficiency of use". A more detailed discussion of the strengths will follow in the next chapters.

4.6.4. From usability problems to usability patterns

Problems and solutions

The results described in the previous section comprise a collection of 207 usability problems and a smaller collection of 84 tool-specific interface strengths. In order to provide hints for the design of user-friendly annotation tools, it is, however, necessary to suggest solutions for the identified problems. According to (Nielsen, 1994c, p. 31), "heuristic evaluation does not provide a systematic



H1 Visibility of system status, **H2** Match between system and the real world, **H3** User control and freedom, **H4** Consistency and standards, **H5** Error prevention, **H6** Recognition rather than recall, **H7** Flexibility and efficiency of use, **H8** Aesthetic and minimalist design, **H9** Help users recognize, diagnose, and recover from errors, **H10** Help and documentation

Figure 4.13.: Number of successfully applied heuristics during the series of heuristic walkthroughs for eleven annotation tools.

way to generate fixes to the usability problems" – yet, he notes that the heuristics used during a heuristic evaluation (or a heuristic walkthrough) do not only help to identify problems, but can also be used as generic guidelines. If for instance a problem is discovered that violates the heuristic "Visibility of system status", the obvious solution is to implement appropriate feedback for the system, to make its status transparent and visible for the user. Nielsen (1994c, p. 31) also mentions that "many usability problems have fairly obvious fixes as soon as they have been identified". In the case of this evaluation study, where a total of eleven linguistic annotation tools was systematically analyzed with the same evaluation approach, it may also be possible to use interface strengths of one tool as a solution for the problem of another tool.

Documentation

Once usability problems and solutions have been identified, it is necessary to document the data in a way it can be shared with software developers. Typically, the results of a usability evaluation are shared as *usability problem reports*. Jeffries (1994, p. 273-274), however, notes that problem reports oftentimes run the risk of being misunderstood or not being taken seriously (*false alarms*) by the developers. In order to find out how such misunderstandings come about, and how they can be avoided, Jeffries (1994, p. 274ff.) analyzed a large collection of usability problem reports that resulted from different evaluation techniques. Jeffries (1994, p. 283) found that "problem reports can be decomposed into four aspects: the problem, the justification for why the current situation is a usability problem, the proposed solution, and the justification for why the proposed solution is better". He concludes that misunderstandings are likely whenever one or more of those aspects are neglected in the problem report.

Usability patterns

Considering the four aspects identified by Jeffries, *usability patterns*, as described by Van Welie (2001), are a promising format to document the results of the heuristic walkthrough evaluation study on linguistic annotation tools: The structure of usability patterns not only integrates very well with the evaluation results (cf. Figure 4.14), but also ensures that the problem and its context, potential trade-offs as well as a rationale for the solution are documented in a consistent and comprehensible form.

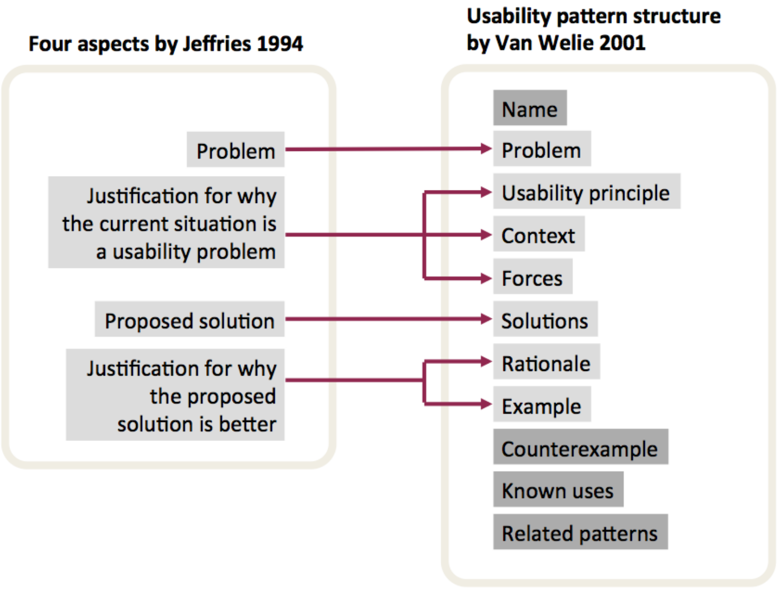

Four aspects by Jeffries 1994
Problem
Justification for why the current situation is a usability problem
Proposed solution
Justification for why the proposed solution is better
Usability pattern structure by Van Welie 2001
Name
Problem
Usability principle
Context
Forces
Solutions
Rationale
Example
Counterexample
Known uses
Related patterns

Figure 4.14.: Overview of the most important aspects of usability reports (Jeffries, 1994, p. 283) and their connecting factors to the usability pattern format as suggested by Van Welie (2001).

4.7. Summary

This chapter has introduced the basic components of usability testing and usability engineering. The usability inspection method *heuristic walkthrough* was chosen as an appropriate method for the evaluation of eleven annotation tools. The results of this series of heuristic walkthroughs – 207 usability problems and 84 tool-specific strengths – were documented alongside with information about the violated usability heuristics and the domain-specific category the data item belongs to. This additional information was used for a first, quantitative interpretation of the data. Beyond the scope of such a mere quantitative approach, it has been proposed to document the content of the usability problems and strengths in a format that enables annotation tool developers to quickly access the relevant information.

Related literature suggests that typical usability reports oftentimes lack relevant, contextual information, making it hard to communicate the essential design knowledge that has been distilled from the usability evaluation to the target audience, the developers. As a solution to this problem, usability patterns have been suggested as a means to document design knowledge in a transparent and comprehensible way. The idea of *design patterns* as a means to document usability knowledge, the *structure of usability patterns*, and the process of *identifying patterns* are described in more detail in the next chapter.

5. Patterns and pattern identification

5.1. Introduction

This chapter introduces *patterns* as a means to document and share design knowledge. More specifically, the pattern format will be used to communicate the main results of the previous evaluation of linguistic annotation tools. After a general discussion of the process of identifying patterns, a systematic pattern identification approach that integrates qualitative data from the heuristic walkthrough study with an existing pattern structure will be presented.

Chapter structure

The chapter is structured as follows: Section 5.2 introduces the original pattern concept, as proposed by Christopher Alexander. As an addition, section 5.3 presents the related concept of *antipatterns* and illustrates why patterns were chosen over antipatterns in this work. Section 5.4 gives an overview of how the pattern concept has been adopted by the HCI community; section 5.5 presents the basic constituents of a pattern as well as common formats that are used to document patterns. Section 5.6 is dedicated to the issue of identifying patterns, which oftentimes is a rather vague process. Therefore, a systematic approach for the identification of usability patterns, which makes use of data from previous usability evaluations, is presented. Section 5.7 illustrates this systematic pattern identification process as it describes a step-by-step guide for the identification of an exemplary usability pattern for annotation tools. Finally, section 5.8 summarizes the whole chapter and leads to the next chapter, which describes the main results of this dissertation: a collection of *usability patterns* for the domain of linguistic annotation tools.

5.2. Christopher Alexander's pattern concept

During the 1970s, the architect and mathematician Christopher Alexander proposed the idea of design patterns for the construction of towns (Alexander et al., 1977; Alexander, 1979). He observed that in architecture there seems to be a *timeless way of building*, in which certain successful – yet rather implicit – design solutions for towns appear over and over again.

Quality without a name

Alexander (1979, chap. 2) suggests to capture such implicit design knowledge, which implements a certain *quality without a name*, in the form of what he calls *patterns*:

> Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. (Alexander et al., 1977, p. x)

Looking for invariants

A pattern is meant to capture the essentials of a problem by gathering "the invariant property common to all places which succeed in solving the problem" (Alexander et al., 1977, p. xiv). According to Alexander, patterns are thus a solution to a recurring problem situation that is formulated in a rather generic and abstract way, so that it can be applied to a wider range of actual scenarios:

> Each solution is stated in such a way that it gives the essential field of relationships needed to solve the problem, but in a very general and abstract way – so that you can solve the problem for yourself, in your own way, by adapting it to your preferences, and the local conditions at the place where you are making it. (Alexander et al., 1977, p. xiii)

Pattern languages

Alexander et al. (1977) present a comprehensive collection of 253 architectural patterns that are interconnected with each other to form a network of patterns, a so called *pattern language*. The authors also suggest different levels of abstraction for patterns, which may nevertheless be connected to each other (Alexander et al., 1977, p. xii). Although the idea of patterns has spread to many areas, and has been applied to many domains, only few pattern collections are nearly as comprehensive as the Alexandrian pattern language. For this reason, patterns that do only address a few aspects of an application domain, and "show almost no relationships among each other and thus do not form a fully interconnected system" (Kruschitz & Hitz, 2009, p. 203) are usually referred to as pattern catalogues or pattern collections. Borchers subsumes Alexander's premises on patterns and pattern languages in the following definition:

> Put simply, a *design pattern* is a structured textual and graphical description of a proven solution to a recurring design problem. A *pattern language* is a hierarchy of design patterns ordered by their scope. High-level patterns address large-scale design issues and reference lower-level patterns to describe their solution. (Borchers, 2001, p. 7)

Patterns as proven solutions?

Although the idea of "patterns as a *proven* [my emphasis] solution to a recurring problem" has become a widely accepted, short definition for software engineering and HCI design patterns, Alexander et al. (1977) did not actually use the word "proven" in their elaboration on patterns (cf. Coplien, n.d.). While on the one hand it is unclear how much actual "proof" is needed in order to make a pattern, and how such proof is to be produced, this additional requirement also seems to be (partly) conflicting with the three basic ways Alexander (1979, p. 258ff.) describes for the identification of patterns. These include the observation of existing, positive examples, which may be interpreted as "proven" solutions, but also the deduction of solutions from negative examples, and even the generation of a pattern, merely on basis of an abstract argument that has no positive or negative proof at all (also cf. section 5.6.1 for a more detailed description of the different pattern identification approaches).

Patterns as a useful format

Even though the idea of *proven* solutions may be hard to implement, this dissertation, nevertheless, utilizes the pattern concept as it has been adapted by the software engineering and HCI communities, i.e. patterns are primarily used as a format to capture design knowledge. More concretely, in this work patterns will be used as a means to document the results from a series of usability evaluations on linguistic annotation tools, and to communicate these results in a structured and comprehensive way to tool developers.

The focus on patterns as a useful format to document user interface design knowledge also seems to be in line with Christopher Alexander's personal assessment of design patterns as used in the software engineering field. In a keynote address at the ACM *Conference on Object-Oriented Programs, Systems, Languages and Applications* (OOPSLA) in 1996, Alexander commented on the application of his pattern concept in the domain of software engineering: Although admitting to be not fully aware of all the relevant work that has been done in the context of software engineering design patterns, Alexander (1996, section A – Pattern theory) observes that patterns seem to be used primarily as an inspiring *format* that supports the documentation and exchanging of fragmentary ideas about programming.

5.3. Antipatterns

Documentation of bad solutions

While the purpose of design patterns is to document successful solutions to a recurring problem, there is also the notion of *antipatterns*, which are meant to create an awareness for bad solutions to a problem that a user might be inclined to use. An antipattern superficially looks like a solution to a recurring problem, but essentially it is not (Koenig, 1998, p. 387), or as Brown et al. (1998, p. 7) put it:

> An AntiPattern is a literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences. The AntiPattern may be the result of a manager or a developer not knowing any better, not having sufficient knowledge or experience in solving a particular type of problem, or having applied a perfectly good pattern in the wrong context. When properly documented, an AntiPattern describes a general form, the primary causes which led to the general form; symptoms describing how to recognize the general form; the consequences of the general form; and a refactored solution describing how to change the AntiPattern into a healthier situation.

In a way, patterns and antipatterns describe *dos and don'ts* during the design process. An antipattern, however, should not only document generic bad solutions, the causes for it, and its consequences, but also a refactored solution (Brown et al., 1998, p. 7). Antipatterns and the idea of *refactoring* (cf. Fowler, 1999) are typically used in the field of software engineering, but are currently also discussed in the context of usability engineering (cf. Garrido et al., 2011).

Patterns vs. antipatterns

In a case study on the effects of first exposure on patterns and antipatterns in the context of HCI, Van Biljon et al. (2004, p. 181) found that positive patterns are better suited for knowledge transfer on the first encounter with a particular design problem. It also showed that pattern novices run the risk of remembering an antipattern (on first encounter with a design problem) as the actual solution to a problem, and that it is hard to replace their mental model with the correct pattern afterwards. While the authors make several suggestions on how to improve antipatterns, they also stress that antipatterns should only be used if the developers are already familiar with positive patterns in the domain of application (Van Biljon et al., 2004, p. 184). On these grounds it seems reasonable to use patterns rather than antipatterns in the context of this work, as there are no existing design patterns in this area of application. Besides, it is unlikely that the target audience will be familiar with the general concept of usability patterns, which is a prerequisite to successfully apply antipatterns.

5.4. Characteristics of HCI patterns

Patterns as knowledge repositories

Although Alexander introduced the idea of patterns in the context of urban architecture, the approach was quickly adopted in other fields as well. Patterns have been particularly popular in the field of software engineering, where they have been used to capture best practices of software design. One of the most influential pattern collections in this field is the "Gang of Four" book (Gamma et al., 1995). Being a means to capture design knowledge, patterns also have found their way into the HCI community, as an addition to existing techniques such as *best practices*, *heuristics* or *guidelines*[173]. Borchers (2001, p. 28) notes that patterns for architectural design, as they were originally described by Alexander, can be transformed to the area of interface design more naturally than to software engineering. The capturing of design knowledge is crucial to document the lessons learned from a project, which help to avoid repeating the same errors (Borchers, 2001, p. 5) and reinventing the wheel (i.e. successful solutions for a problem) over and over again (Kruschitz & Hitz, 2009, p. 202).

Patterns vs. guidelines

Borchers (2001, p. 5) also observes that design guidelines are the most common approach to document such design knowledge. He, however, notes that most guidelines are either to concrete (e.g. Ben Shneiderman's "8 Golden Rules of

[173] As there is no common ground on the exact differentiation of methods to capture design knowledge, in this dissertation *guidelines* are used as an umbrella term for related techniques such as *best practices*, *heuristics*, *golden rules*, *maxims*, *recommendations* etc.

Interface Design" or Jakob Nielsen's "10 Usability Heuristics for User Interface Design") or to specific (e.g. the "Macintosh Human Interface Guidelines"). Van Welie et al. (2001, p. 2) summarize some of the downsides of guidelines as a means for documenting design knowledge, which may be overcome by using patterns instead[174]:

- Guidelines are often too simplistic or too abstract
- Guidelines can be difficult to select
- Guidelines can be difficult to interpret
- Guidelines can be conflicting
- Guidelines often have authority issues concerning their validity

Patterns, with their predefined internal structure that expresses "a relation between a certain context, a problem, and a solution" (Alexander, 1979, p. 247), seem to be an appropriate instrument to make up for the downsides of design guidelines[175]:

> As an alternative to guidelines, we propose patterns as a solution to some of the problems using guidelines. Patterns explicitly focus on context and tell the designer when, how and why the solution can be applied. The solutions are concrete so that they can be applied directly. Because of such a different focus, patterns can potentially be more powerful than guidelines as tools for designers. (Van Welie, 2001, p. 93)

Benefits of using patterns

The benefits of patterns as a means to document design knowledge are also underlined by an online survey that was conducted by Kruschitz & Hitz (2010). The motivation of the study was to empirically proof that HCI design patterns are actually used in academic and industrial environments, and that they improve the design process. It shows that approx. 60% of all survey participants (n=286) report that they have used design patterns in their work (Kruschitz & Hitz, 2010, p. 712). Approx. 71% of those who use patterns confirm that it does improve the overall design process:

> Patterns are improving the design process because patterns can be used as checklists, patterns are more contextualized than other reuse concepts (e.g. guidelines), they help to avoid common pitfalls, and patterns help maintain consistency in larger projects. Furthermore, patterns provide access to proven and well-documented solutions and they are providing a better overall user experience. These statements are a clear indicator that patterns bring more quality to UIs. (Kruschitz & Hitz, 2010, p. 713)

Definition: HCI design pattern

The idea of patterns as a means to document design knowledge from the domain of HCI and usability engineering goes back to the mid-1990s. An important milestone in the evolution of HCI design patterns was a workshop named "Usability

[174] Also cf. Pemberton (2000) for an overview of various tools and representations for the documentation of design knowledge and a discussion about the potential of design patterns.

[175] For an extensive discussion on the differences between guidelines and patterns cf. Van Welie (2001, p. 93).

Pattern Language: creating a community", which was held as part of the *INTERACT* conference in 1999[176]. The participants agreed upon the following goals for HCI pattern languages:

> The goals of an HCI Pattern Language are to share successful HCI design solutions among HCI professionals, and to provide a common language for HCI design to anyone involved in the design, development evaluation, and use of interactive systems. (Borchers, 2001, p. 39)

At the same time, Van Welie (2001, p. 98) highlights that interaction patterns are *successful* if they achieve a high level of usability on the interface level. Besides these rather generic definitions of the goals HCI patterns are trying to achieve, there is also a discussion about different flavors of patterns, and how they can be distinguished. Bayle et al. (1998) note that there are at least two basic types of patterns: (1) *design patterns*, which claim to present a proven solution to a recurring problem, and (2) *activity patterns*, which just document existing interaction behaviors. Although most HCI patterns are of the first type, there are some examples for the application of activity HCI patterns: Martin et al. (2001) report an approach, where the results from ethnographic studies have been transformed into *patterns of cooperative interaction*, with the main goal being to bridge a communicative gap between fieldworkers and designers. The authors found it hard to apply the classical, problem-oriented design patterns to their domain of knowledge, and rather chose to use descriptive patterns to document "recurrent phenomena . . . without necessarily making judgements" about the success of these activities (Martin et al., 2001, p. 47).

Design patterns vs. descriptive patterns

Another way to distinguish patterns is by their scope and their granularity (Alexander et al., 1977, p. xii). Mahemoff & Johnston (1998) suggest four levels of abstraction to distinguish HCI patterns:

Distinguishing patterns by scope

UI elements Example pattern: "Scrollbar" (context: user's complete working area cannot be displays at one time)

Tasks Example pattern: "Open Existing Document" (context: user needs to open a document inside a software)

Entire systems Example pattern: "Document Manipulator" (context: user needs to arrange a set of elements in a document)

Users Example pattern: "Intermediate User, Domain Expert" (context: a domain expert who regularly uses a tool is likely to evolve from a novice user to an intermediate user)

Borchers (2000, p. 9-10) picks up the idea of different levels of abstraction and identifies *large-scale patterns* (complete task), *small-scale patterns* (style of certain interactions) and *low-level patterns* (UI objects). By adding a *functional dimension* (perception / interface output, manipulation / interface input, navigation) and a

[176] `www.it.bton.ac.uk/staff/rng/UPLworkshop99/`

physical dimension (spatial layout, sequence of discrete events, continuous time), Borchers suggests a comprehensive taxonomy for HCI patterns.

Types of usability patterns

As has been described in the previous chapter, usability is a complex concept that can be broken down into several attributes, such as *learnability* and *efficiency* (cf. Nielsen, 1993, p. 26). Correspondingly, Casaday (1997, p. 289-290) suggests to differentiate usability patterns according to the number of different usability attributes that can be observed in the pattern:

Simple patterns One usability attribute dominates; solutions are fairly easy.

Intrinsic patterns Multiple usability attributes are involved; solutions are more difficult.

Circumstantial patterns Usability attributes cannot easily be achieved due to external factors.

Wording: usability patterns

Krischkowsky et al. (2013, p. 67) note that HCI patterns are known under a wide variety of names, such as *interaction patterns*, *user interface patterns*, *usability patterns*, *web design patterns*, etc., but that they all share the common goal to "provide solutions to commonly occurring usability problems in interaction and interface design". Throughout this work, the term *usability patterns* (or simply patterns) will be used to indicate a close relation to the identified usability problems. This relation will also be made clear in the structure of the pattern, which is described in more detail in the following section.

5.5. Structure of patterns

Alexandrian format

In order to describe and discuss the structure and composition of patterns, it is helpful to go back to the Alexandrian antetype of a pattern: All of Alexander's architectural patterns follow a common, implicit structure, which is realized by means of formatting and typography. Borchers (2001, p. 18ff.) identifies nine basic characteristics of the patterns proposed by Christopher Alexander:

1. **Name** of the pattern should convey the idea of the pattern in a few words.
2. **Ranking** and validity of the pattern (scale: 0-2).
3. **Picture** as an example of its application, i.e. photographs of an actual realization of the pattern.
4. **Context** in which it is used. Also relation to larger-scale patterns: How does this pattern help to implement larger-scale patterns?
5. **Short problem statement** which summarizes the general situation and illustrates the existing problem or conflict.

6. **Detailed problem description** with empirical background; description of "competing forces" (e.g. build higher walls for more privacy vs. build lower or no walls for a more public feeling) and discussion of existing solutions.

7. **Central solution** of the pattern describes a general solution to a problem at hand. This solution is clear but also to some degree generic, so that it can be applied in varying situations.

8. **Diagram** should illustrate the solution in a generic way.

9. **References** to other patterns (patterns are ultimately meant to form an interconnected pattern language).

HCI pattern structures

When designing his own set of patterns for HCI, Borchers leaves out Item 4, and subsumes Items 5 and 6. He also introduces a new item "Examples", which shows real world solutions to the problem at hand, illustrating how the forces are balanced (Borchers, 2001, p. 52-54). Despite the efforts to standardize the structure of design patterns by means of an XML-based *Pattern Language Markup Language* (PLML, cf. Fincher, 2004), there exists a plethora of different pattern formats in the area of HCI, (cf. for instance Obrist et al., 2010; Tidwell, 2011; Van Welie, 2001)[177], whose least common denominator are the three basic parts as described by Alexander:

> Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution. As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occur repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves. (Alexander, 1979, p. 247)

Van Welie pattern structure

The structure for usability patterns proposed by Van Welie et al. (2001) is similar to the Alexandrian format as described by Borchers (2001, p. 18ff.), but also shows some modifications toward the documentation of specific *usability-related* information (cf. Figure 5.1). It is important to note that the first version of the usability pattern structure (Van Welie & Van der Veer, 2000; Van Welie et al., 2001) was slightly modified in Version 2, Van Welie's PhD thesis (Van Welie, 2001). While only the second version of the usability pattern structure will be used in this dissertation, the first version is an important intermediate step in the genesis of Van Welie's usability pattern format.

First of all, the usability pattern format abandons the idea of a generic validity ranking for patterns, but rather introduces a more concrete section named *usability impact*. According to Van Welie et al. (2001, p. 6), each usability pattern must improve a concrete usage indicator such as *learnability* or *memorability* (also cf. Van Welie's layered model of usability in chapter 4.2), or it is not a usability pattern. In addition, a section named *usability principle* documents the higher-level ergonomic principle that is the basis of the pattern, while the section *rationale* is

[177] For a short review of different pattern formats (not only for HCI) cf. Kruschitz & Hitz (2009, p. 203-204).

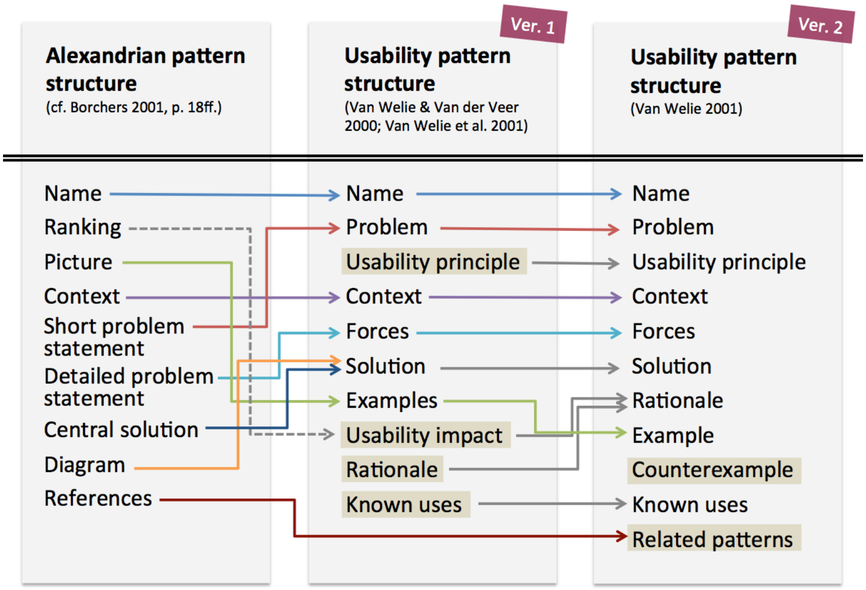


Figure 5.1.: Comparison of the Alexandrian pattern structure (cf. Borchers, 2001, p. 18ff.) and usability pattern structures as proposed by Van Welie & Van der Veer (2000), Van Welie et al. (2001) and Van Welie (2001)

meant to explain how the principle was used and why it leads to an improvement of the usage indicators (Van Welie et al., 2001, p. 6). Alexander's *short problem statement* corresponds to the *problem* section, while *detailed problem statement* corresponds to the *forces* section. The *picture* of the pattern in actual use is part of the *examples* section proposed by Van Welie et al. (2001), whereas the *diagram* may be part of the *solution* section. Van Welie et al. (2001) introduce a new section named *known uses*, which describes further examples that implement the described pattern. In the second version, Van Welie (2001, p. 104-105) suggests two more sections *counterexamples* and *related pattern*. While the latter corresponds to Alexander's *references* section, *counterexamples* is an optional section that may describe systems that suffer from usability problems that could have been avoided by using the described pattern. Another adjustment in the second version of the usability pattern structure is the inclusion of the *usability impact* into the *rationale* section.

For the context of this work, the pattern structure (Version 2) proposed by Van Welie (2001, p. 102ff.) was chosen to capture usability knowledge for the design of linguistic annotation tools. Van Welie's pattern structure is well documented and also integrates very well with the qualitative data gathered from the heuristic walkthrough evaluation. The following list shows the complete template with detailed descriptions for each section (cf. Van Welie, 2001, p. 102ff.).

Name A pattern should have a short, catchy name that either relates to the problem or the solution (Van Welie, 2001, p. 102).

Problem This section describes a (typically task-related) user problem the pattern is trying to solve, and the objectives it is trying to achieve (Van Welie, 2001, p. 102-103). In this work, wherever applicable, there will be a reference to concrete problems identified during the series of heuristic walkthroughs.

Usability principle The solution described in usability patterns is usually based on some existing, higher-level usability principle (Van Welie, 2001, p. 103). As Van Welie notes that there is no complete list of all usability principles, for this dissertation it was decided to use the principles described in the heuristics by Nielsen (1994a), which integrate very well with the heuristic walkthrough inspection method (cf. chapter 4.5.2).

Context Contextual information on when to use a certain pattern is one of the big advantages over other forms of documenting design knowledge, such as guidelines, which oftentimes assume that a design solution can be generally applied. This section describes the context as well as the conditions under which the previously stated problem occurs, and helps to determine when the pattern can be applied successfully (Van Welie, 2001, p. 103).

Forces Typically, a problem will be complex and intricate, i.e. in most cases there is no silver bullet that solves a problem, but rather trade-offs have to be made in order to find a solution. A good pattern should try to balance existing forces and conflicts in an ideal way (Van Welie, 2001, p. 102-103). Forces may be seen as an extension of the contextual information given in the section before.

Solution Van Welie (2001, p. 103) suggests to summarize the solution in an introductory sentence, and then describe in more detail how the problem is solved by means of prose, and optionally diagrams and sketches. Van Welie also underlines that the solution should be formulated in a rather abstract, generic way, so it can be applied and implemented for various different scenarios (that match the contextual conditions described in the *context* section). The solution describes the visible structure as well as the behavior of the pattern (Van Welie, 2001, p. 103).

Rationale The rationale makes clear why a pattern works by providing "a reasonable argumentation for the specified impact on usability" (Van Welie, 2001, p. 104). This argumentation can be based on psychological, sociological, ergonomic or organizational aspects. Van Welie also suggests to describe the impact on the following aspects of usability: *performance speed*, *learnability*, *memorability*, *satisfaction*, *task completion* and *errors*.

Example Concrete examples (typically a screenshot) help to understand the pattern and underline that the solution has been proven to work in existing applications (Van Welie, 2001, p. 104). In this work, examples of annotation tools that successfully implement the described pattern will be described. If no examples are available in the evaluated tools, self-created prototypes will be provided.

Counterexample This optional section gives examples of applications that do not use the pattern, and underlines the case for using the pattern (Van Welie, 2001, p. 104).

Known uses This section is a list of further examples of applications that implement the pattern, but not as detailed as in the example section (Van Welie, 2001, p. 104).

Related patterns In the last section of the pattern structure, other patterns that address related problems may be referenced (Van Welie, 2001, p. 105). Relations may either be identified inside the collection of annotation tool patterns, or for existing, external HCI pattern collections.

5.6. Identifying patterns

The previous section has laid out in great detail how patterns are structured, and how they can be used to document design knowledge. However, the issue of how to actually *identify* patterns is rather delicate, and deserves a section of its own.

5.6.1. Different approaches for the identification of patterns

Alexander (1979, p. 258ff.) describes three basic approaches for identifying patterns that may be characterized as being more *inductive* or more *deductive*, or a *mix* of both modes of reasoning (also cf. E-Teaching.org Redaktion, 2011).

Inductive approach

Inductive pattern identification means to observe and analyze a set of positive examples (Alexander, 1979, p. 258), and to extract the *invariant* that solves a particular problem of these concrete examples. The inductive approach, which seems to be widespread in the field of software engineering, is also described by DeLano (1998, p. 87), who notes that "patterns are present in the artifacts that already exist". The process of extracting patterns from artifacts that implement a proven solution (cf. Borchers, 2001, p. 7) to a problem is often called *pattern mining*. DeLano (1998, p. 88) explains why the metaphor of *mining* is much better suited than for instance *pattern hunting* (implies randomness) or *pattern harvesting* (patterns cannot be planted systematically, and they are not just there for the taking):

> The mined elements need to be removed as gingerly as a fossil or artifact. The elements must be further processed before it [sic!] become useful. After refinement – cutting, polishing, smelting, molding – we are left with a useful product.

DeLano (1998, p. 90ff.) describes three basic approaches for finding artifacts that are suited for inductive pattern mining:

Individual contributions Pattern mining based on one's own expertise / experience; method: creative process (DeLano, 1998, p. 90)

Second-hand contributions Pattern mining based on other experts knowledge; method: interview (DeLano, 1998, p. 91)

Pattern mining workshops Pattern mining based on the knowledge of a group of domain experts; method: focus group (DeLano, 1998, p. 93-94)

Combined approach

If there are no existing, positive examples, from which patterns can be extracted, Alexander (1979, p. 258-259) suggests a combined approach of induction and deduction: First, a number of negative examples are observed and analyzed (induction). Second, solutions are created based on the previously identified problems (deduction). It, however, remains unclear how the solutions can be created systematically, and how an actual pattern can be generated based on the observation of negative examples.

Deductive approach

As a third approach, Alexander (1979, p. 259) mentions a purely deductive pattern identification process, which does not start from the concrete observation of positive or negative examples, but rather builds on an abstract argument. By allowing the identification of patterns through concrete observation only, Alexander (1979, p. 259) notes that it would be "impossible to find patterns which do not already exist in the world already . . . and would therefore imply a claustrophobic conservatism".

5.6.2. Practical problems of pattern identification

Despite the manifold approaches for discovering patterns, there remains a gap between writing patterns, which is well described in literature (cf. for instance Wellhausen & Fiesser, 2012), and missing information on how to *systematically* identify them:

> While considerable literature exists on documenting patterns of different types little is said in the pattern community about the genesis of patterns. It is unclear how patterns come into existence and how they should be generated. The core of most descriptions is that a series of "pattern workshops" are held where patterns are identified and expressed using some form of pattern language. (Martin et al., 2001, p. 43)

Usability pattern engineering

Along the same lines Krischkowsky et al. (2013, p. 66) note that in the field of HCI very little is known about a structured process for pattern creation, but that pattern creation rather relies on the implicit knowledge of experienced designers. This notion seems to be conflicting with the idea of *pattern mining*, which implies a more or less structured mining process. It also stands in contrast to the fundamental presupposition that usability is not a "quality without a name", but rather can be tested and engineered systematically by applying a wide set of available methods (cf. chapter 4.3). On these grounds, Krischkowsky et al. (2013, p. 68ff.) present a "step-by-step pattern generation guidance" for domain experts who have conducted some sort of empirical UX study, and who typically have little or no experience in pattern writing. The five-step process is mainly intended to introduce HCI experts to the concept of patterns and to make them reflect on the key findings from their previous study by providing various checklists. While the process is well structured and easy to follow, it remains unclear what kind of *key findings* the authors have in mind, and how they can be systematically transformed into a pattern.

5.6.3. Toward a systematic pattern identification process

Systematic pattern identification

In order to make the process of identifying and documenting usability patterns more transparent and comprehensible, a systematic approach that combines the heuristic walkthrough evaluation method with a usability pattern structure for documenting the evaluation results is proposed. This approach solves some of the problems and open questions that were introduced in the previous section, as many components of the heuristic walkthrough method integrate very well (cf. Figure 5.2) with the pattern structure as proposed by Van Welie (2001, p. 102ff.).

Categories

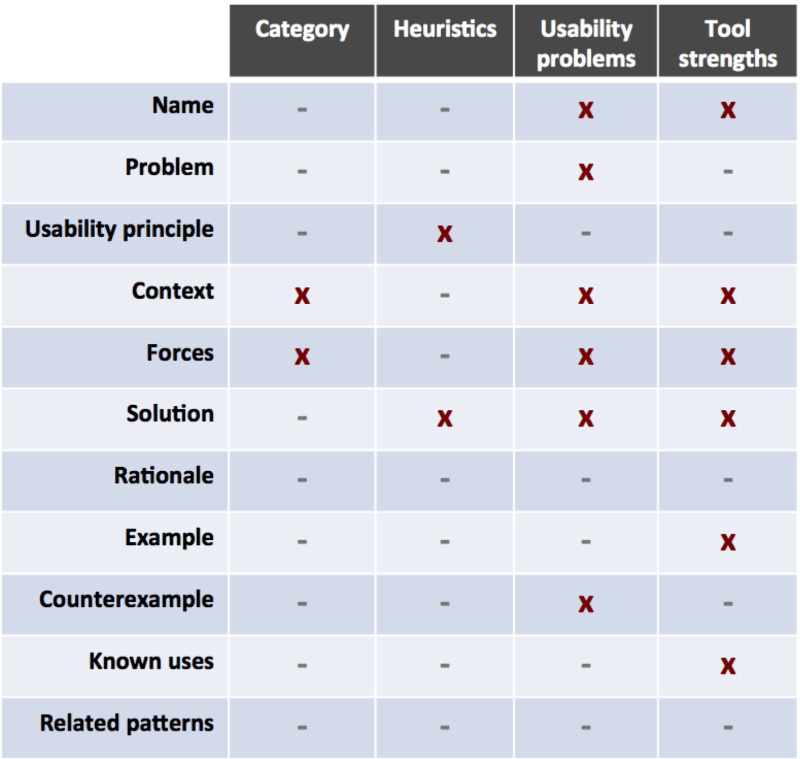
The domain-specific categories (cf. chapter 4.5) that were used to systematize the problems and strengths during the heuristic walkthrough can be used as generic input for the *context* and *forces* sections, which both contain contextual information on how and when the pattern can be applied.

Heuristics

Another key element of the heuristic walkthrough, the heuristics, also fit into the pattern structure very well: While heuristics on the one hand facilitate the systematic discovery of usability problems (cf. chapter 4.5.2), they can also be used as positive usability principles that help to formulate a solution for the problem. For this reason, heuristics are used as an explicit reference in the *usability principle* section, but also as implicit input in the *solution* section.

Usability problems

The usability problems that were identified during the series of heuristic walkthroughs (cf. chapter 4.6.2) are an essential input for several fields of the pattern: First of all, a problem can be used to find an adequate *name* for a pattern, as the name should always relate to the problem or the solution. Second, one or more problems will be paraphrased in the *problem* section. Additionally, all associated usability problems will be listed alphabetically at the end of the problem section, to make the process more transparent. Third, usability problems can be used to describe the *context* in which the patterns can be used. They also can serve as input for the extended context section, the *forces*. As "many usability problems have fairly obvious fixes as soon as they have been identified" (Nielsen, 1994c,



	Category	Heuristics	Usability problems	Tool strengths
Name	-	-	x	x
Problem	-	-	x	-
Usability principle	-	x	-	-
Context	x	-	x	x
Forces	x	-	x	x
Solution	-	x	x	x
Rationale	-	-	-	-
Example	-	-	-	x
Counterexample	-	-	x	-
Known uses	-	-	-	x
Related patterns	-	-	-	-

Figure 5.2.: The matrix shows how some of the core elements of the heuristic walkthrough can be integrated with the usability pattern structure.

p. 31), they may also be used as input for the formulation of an adequate *solution*. Finally, usability problems can help to describe tools that qualify as a *counterexample* for the respective pattern.

Tool strengths

Similar to the usability problems, the identified tool strengths (cf. chapter 4.6.3) can be used as input to formulate an adequate *name* for the pattern. Much like the usability problems, specific strengths of a tool can help to elaborate the context of application, which is documented in the *context* and *forces* sections. Obviously, strengths of a tool can be generalized in the *solution* section of a pattern. If appropriate, they will also be documented in the *example* (screenshot and short description) or *known uses* (mention of tool name) section.

Clustering of problems and strengths

The systematic identification of usability patterns for linguistic annotation tools starts by examining the results of the previously conducted series of heuristic walkthroughs. As was described before (cf. chapter 4.3.7), the heuristic walkthrough method allows the evaluator to reveal usability problems as well as particularly good solutions for certain problems. Conducting an evaluation of eleven annotation tools, however, leads to a large number of identical or at least thematically similar problems and strengths. The basic idea for the pattern identification process is to create clusters of similar problems that recur for different annotation tools. If there are strengths that can be used to solve these very problems, they are also included in the cluster. Such thematic clusters are then used as input for the usability pattern structure described in Figure 5.2. In order to facilitate the creation of clusters, all of the 207 usability problems as well as the 84 strengths were printed out as small cards. The front side of the card contained the ID, name and description of the problem, the back side contained the heuristics, the severity ranking and the task category (cf. Figure 5.4).

Similarities to content analysis

The clustering approach is similar to the qualitative method of *content analysis* (cf. Krippendorff, 2013), which is summarized by Elo & Kyngäs (2008, p. 108) as follows: "Through content analysis, it is possible to distill words into fewer, content-related categories". The technical process of analyzing and distilling qualitative data is typically referred to as *coding* (Lazar et al., 2010, p. 289). According to Stemler (2001, para. 12ff.), *categories* may either be created in an ad hoc fashion during the coding process (*emergent coding*), or be established before the actual analysis (*a priori coding*). One big difference between the clustering approach described in this work, and the content analysis described by Krippendorff (2013) and others, is the number of coders. While content analysis typically involves multiple persons who code the data (*reliability* of different coders as a quality criterion), the analysis of the results of a single-evaluator heuristic walkthrough is performed by only one person, and thus is a rather subjective approach, that nevertheless is trying to achieve similar goals as content analysis.

Clustering phases

The actual clustering process was conducted in two phases: In the first phase, usability problems and strengths were clustered according to the domain-specific category they were tagged with during the heuristic walkthrough (*a priory coding* approach; cf. Figure 5.3). The same categories were also used to categorize the patterns later on (cf. chapter 6). In the second phase, the cards were analyzed

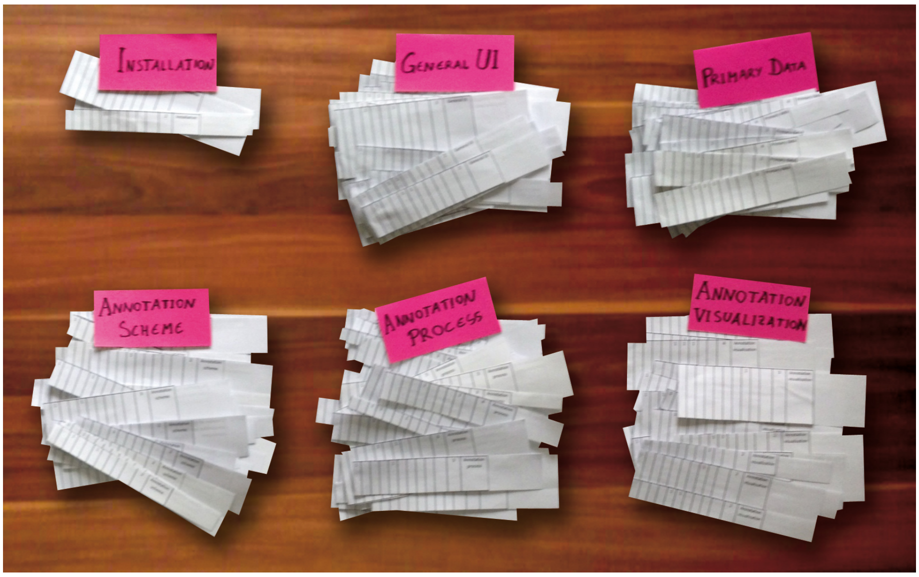


Figure 5.3.: Photograph of the category cluster phase (cards are displayed from their back side).

with regard to their content. Problems and strengths that describe a similar issue (e.g. "visualization of parallel annotations") were grouped to form clusters of their own (*emergent coding* approach). During this clustering phase, three basic scenarios occurred:

1. **Existence of problems and appropriate solutions** – Ideally, a solution to a recurring problem can be identified in one of the test objects (inductive approach), which may then be used to relate both, the problem categories and the good solutions in one common usability pattern.
2. **Existence of solutions without concrete problems** – Some patterns (P3.3, P3.6, P5.1, P5.3, P5.5, P5.6, P5.9) were created merely on the basis of good solutions, without explicitly documented usability problems (inductive approach). It was, however, easy to derive the problem situation that is improved by those strengths retrospectively.
3. **Existence of problems without concrete solution** – In a number of cases, a group of similar problems could be identified without having a concrete solution in the heuristic walkthrough data. In these cases, new solutions were created based on the usability problems and the heuristics that are violated when the problem occurs (inductive-deductive approach).

It showed that not all of the identified problems and strengths qualify for the derivation of a usability pattern. These cases are nonetheless documented and discussed in chapter 6. The next section describes a step-by-step documentation of the genesis of an exemplary pattern, illustrating how the results from the heuristic walkthroughs are used to derive patterns, and which other input is used to fill in the usability pattern structure described above.

5.7. A step-by-step guide for the creation of usability patterns

This section describes the process of creating an exemplary pattern as a number of sequential steps. Steps 1 and 2 describe the two phases of the clustering process. Steps 3 to 13 explain how the heuristic walkthrough data can be used as input for the usability pattern structure, and also show the respective section of the exemplary pattern at the end of each step (set in smaller type). All patterns described in chapter 6 were identified and documented according to this 13-step approach. The rather static *print view* of the patterns in the next chapter was derived from a wiki[178] that serves as the central repository for the documentation

Annotation usability wiki

[178] The wiki is titled "Annotation Usability Wiki - Usability Patterns for Linguistic Annotation Tools" (cf. `http://www.annotation-usability.net`) and was realized by using the *PmWiki* (cf. `http://www.pmwiki.org/`) wiki software.

and curation of the pattern collection. The main advantages of the wiki format are summarized in the following list:

Distribution As the ultimate goal of this work is to share usability patterns with the community of annotation tool designers, an online wiki seems to be a legitimate way for disseminating the patterns beyond the scope of this dissertation.

Interactivity The wiki can be used to link different patterns to each other in a hypertext fashion, which greatly facilitates the navigation and practical usage of the pattern collection. In addition, the wiki provides an easy way to include HTML prototypes as interactive examples (cf. Step 10 in this section) in the patterns.

Change log Changes made to a pattern are automatically stored in a "page history file". This is useful for keeping track of different versions and refinements of a pattern, not only during the course of the initial pattern identification process, but also for a later point in time, when feedback from the community might lead to further modifications of the patterns.

5.7.1. Clustering according to categories (Step 1)

All problems and strengths were clustered according to their domain-specific category. This category was identified during the heuristic walkthrough, and is documented on the back of the printed cards (cf. Figure 5.4).

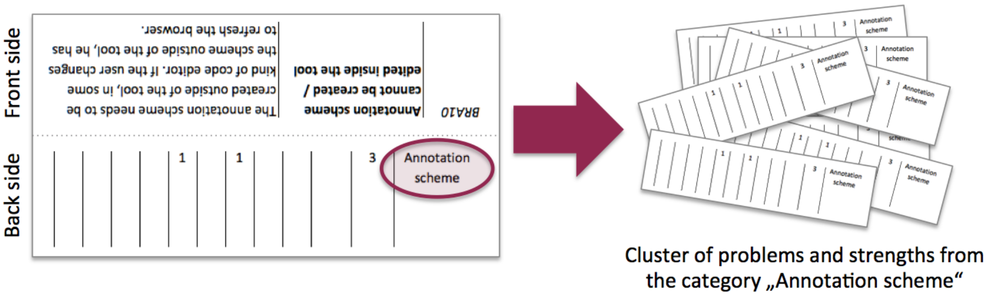
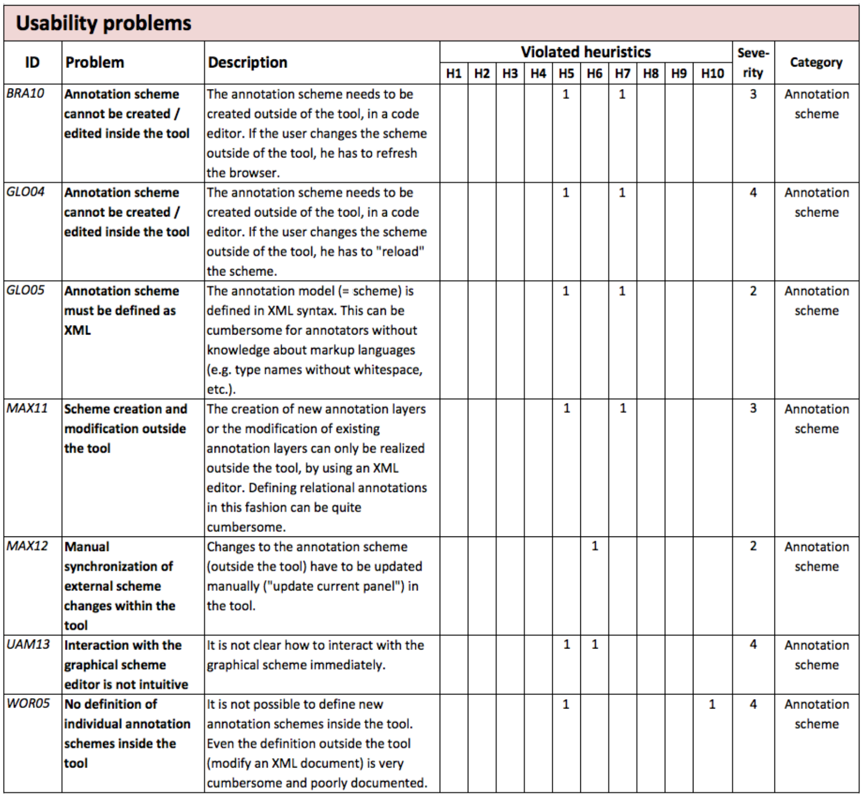

Front side
Back side
BRA10
Annotation scheme cannot be created / edited inside the tool
The annotation scheme needs to be created outside of the tool, in some kind of code editor. If the user changes the scheme outside of the tool, he has to refresh the browser.
1
1
3
Annotation scheme
Cluster of problems and strengths from the category „Annotation scheme“

Figure 5.4.: Example: Clusters of problems and strengths according to their category.

5.7.2. Clustering according to similar content (Step 2)

For this exemplary pattern, problems and strengths from the "Annotation scheme" category were analyzed and clustered according to their *description* content. The problems shown in Figure 5.5 are all concerned with the issue that the annotation scheme cannot be created inside the annotation tool, but rather has to be created in an external code editor, by means of XML markup. At the same time, there are several tools that implement an integrated annotation scheme editor. These strengths are summarized in Figure 5.6 .



Usability problems

ID	Problem	Description	Violated heuristics										Seve-rity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
BRA10	**Annotation scheme cannot be created / edited inside the tool**	The annotation scheme needs to be created outside of the tool, in a code editor. If the user changes the scheme outside of the tool, he has to refresh the browser.					1		1				3	Annotation scheme
GLO04	**Annotation scheme cannot be created / edited inside the tool**	The annotation scheme needs to be created outside of the tool, in a code editor. If the user changes the scheme outside of the tool, he has to "reload" the scheme.					1		1				4	Annotation scheme
GLO05	**Annotation scheme must be defined as XML**	The annotation model (= scheme) is defined in XML syntax. This can be cumbersome for annotators without knowledge about markup languages (e.g. type names without whitespace, etc.).					1		1				2	Annotation scheme
MAX11	**Scheme creation and modification outside the tool**	The creation of new annotation layers or the modification of existing annotation layers can only be realized outside the tool, by using an XML editor. Defining relational annotations in this fashion can be quite cumbersome.					1		1				3	Annotation scheme
MAX12	**Manual synchronization of external scheme changes within the tool**	Changes to the annotation scheme (outside the tool) have to be updated manually ("update current panel") in the tool.						1					2	Annotation scheme
UAM13	**Interaction with the graphical scheme editor is not intuitive**	It is not clear how to interact with the graphical scheme immediately.					1	1					4	Annotation scheme
WOR05	**No definition of individual annotation schemes inside the tool**	It is not possible to define new annotation schemes inside the tool. Even the definition outside the tool (modify an XML document) is very cumbersome and poorly documented.					1					1	4	Annotation scheme

Figure 5.5.: Cluster of usability problems that describe a similar issue.

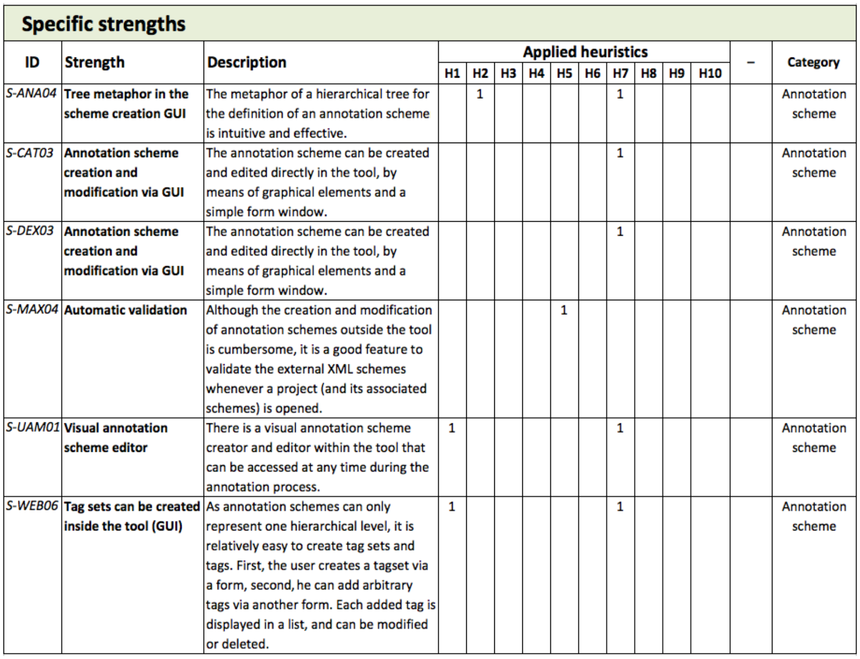
The cluster of problems and strengths that all address the issue of "annotation scheme creation" is then used as input for the different sections of the pattern structure.

5.7.3. Pattern name (Step 3)

In this case, the name of the pattern is closely related to the solution of the problem, which in short is to provide an integrated annotation scheme editor. Accordingly, the pattern is named as follows:

Name: P4.1 - Integrated annotation scheme editor

The pattern ID "P4.1" indicates that it is the *first* pattern in the *fourth* category (*Annotation scheme*).



Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-ANA04	Tree metaphor in the scheme creation GUI	The metaphor of a hierarchical tree for the definition of an annotation scheme is intuitive and effective.		1					1					Annotation scheme
S-CAT03	Annotation scheme creation and modification via GUI	The annotation scheme can be created and edited directly in the tool, by means of graphical elements and a simple form window.							1					Annotation scheme
S-DEX03	Annotation scheme creation and modification via GUI	The annotation scheme can be created and edited directly in the tool, by means of graphical elements and a simple form window.							1					Annotation scheme
S-MAX04	Automatic validation	Although the creation and modification of annotation schemes outside the tool is cumbersome, it is a good feature to validate the external XML schemes whenever a project (and its associated schemes) is opened.					1							Annotation scheme
S-UAM01	Visual annotation scheme editor	There is a visual annotation scheme creator and editor within the tool that can be accessed at any time during the annotation process.	1						1					Annotation scheme
S-WEB06	Tag sets can be created inside the tool (GUI)	As annotation schemes can only represent one hierarchical level, it is relatively easy to create tag sets and tags. First, the user creates a tagset via a form, second, he can add arbitrary tags via another form. Each added tag is displayed in a list, and can be modified or deleted.	1						1					Annotation scheme

Figure 5.6.: Cluster of strengths that describe a similar issue.

5.7.4. Problem description (Step 4)

The problem description paraphrases and summarizes the problems described on each of the cards (cf. Figure 5.5). In addition, all related problems are listed by means of their ID (alphabetic order). The respective *problem description* section for pattern P4.1 looks like this:

> **Problem description**: The creation of an annotation scheme that defines different levels of annotation as well as concrete annotation items on each level is a crucial task in any annotation project. Typically, annotation schemes are defined by means of document grammars known from markup languages like XML or SGML. Users without technical knowledge about markup languages will have difficulties in creating a scheme in XML syntax. At the same time, many tools require to define an annotation scheme outside the annotation tool, which makes the task even more challenging for markup novices.
>
> Related problems: BRA10, GLO04, GLO05, MAX11, MAX12, UAM13, WOR05

5.7.5. Usability principle (Step 5)

As was described before, Nielsen's heuristics cannot only be used to discover usability problems, but also to derive fixes for identified problems. As the heuristics were already used during the heuristic walkthrough, it suggests itself to use the same set of heuristics to describe the usability principles the solution of the pattern is based on. If the heuristics documented for the problems and strengths of the respective cluster are very heterogeneous, only the most important, predominant heuristics are mentioned in the pattern section. In this example, the two predominant heuristics that are violated by the identified usability problems, and that in turn can be used to fix these problems, are "error prevention" and "flexibility and efficiency of use": Markup novices, who have to create and modify an annotation scheme in an external XML editor are likely to produce errors during this process. At the same time, having to switch between the actual annotation tool and an external editor slows down the user and thus reduces his flexibility and efficiency.

Usability principle: Error prevention, flexibility and efficiency of use

5.7.6. Context (Step 6)

The context of a pattern describes in which scenarios and under which conditions it can be used. While a basic, rather generic hint can be derived from the category that is associated with the cluster (in this case *Annotation scheme*), concrete usability problems as well as concrete strengths help to derive a more specific context of use. The context for this example is to provide users who are markup novices with an intuitive and efficient way to formulate annotation schemes.

Context: This pattern can be used to facilitate the creation of annotation schemes for users without technical knowledge about markup languages and document grammars.

5.7.7. Forces (Step 7)

This section of the pattern structure can be seen as an extension of the previous context section, as it describes some conflicting forces that may require tradeoffs when the pattern is applied. The sources of input are the same as in the context section, but they are interpreted in a way that reveals potential conflicts and helps to balance existing contradictions. In this example, the main conflict is that on the technical side, annotation schemes are typically realized by means of a markup language, which is the ideal instrument for this kind of task, as it can be easily processed by machines. On the side of the user, who may not be familiar with the concept of markup languages, this may be a rather irritating and cumbersome way to create an annotation scheme.

Forces

- Annotation schemes are typically defined by means of document grammars (DTDs or XMLSchemas).
- Annotation schemes are typically defined outside of the annotation tool, in a text editor that facilitates the creation of document grammars.
- For editing existing schemes during the annotation process, it is impractical to switch between the annotation tool and an external text editor.

- It is vital to provide a synchronization mechanism that ensures that changes made to the annotation scheme are updated and applied to the current, ongoing annotation process (cf. S-MAX04).

5.7.8. Solution (Step 8)

This is an integral part of the pattern, as it describes a solution to the previously introduced problem. In this exemplary pattern it is possible to derive a solution from many positive examples (strengths) of other annotation tools. Essentially, the solution is to provide an integrated annotation scheme editor that allows the user to create schemes without having to use XML or some other markup syntax.

> **Solution**: The tool integrates a scheme editor that allows the user to define and edit annotation schemes inside the annotation tool. By providing a graphical user interface for the scheme editor it is also possible to hide technical details of the storage format from the user. Such an interface should utilize well-known metaphors for the creation of hierarchical structures, such as file-trees (cf. S-ANA04) or ordered lists. It can also make use of established input elements, such as forms and input fields (cf. S-WEB06). It must be made clear via the interface which annotation items belong to which level of annotation, i.e. typically the annotation levels are at the highest hierarchical level of the scheme, while concrete annotation items can be subordinate to those different levels.

5.7.9. Rationale (Step 9)

The rationale section does not use input from the heuristic walkthrough data, but rather relies on a set of usability aspects (also cf. Van Welie's layered model of usability in chapter 4.2) that help to explain why and how the pattern improves the usability of an annotation tool. The five usability aspects that are used in this pattern collection were introduced by Nielsen (1993, p. 26ff.): *learnability*, *efficiency*, *memorability*, *error rate*, and *satisfaction*. In this example the suggested solution improves the usability of an annotation tool as it helps to increase the overall performance speed (*efficiency*), but also helps to quicker learn (*learnability*) the tool and to avoid errors (*error rate*) when using it productively.

> **Rationale**: As ad hoc modifications of the annotation scheme are part of the typical annotation process, an integrated editor for annotation schemes speeds up the overall annotation process (*efficiency of use*). At the same time, the availability of a GUI for the creation and modification of annotation schemes increases the *learnability* of the annotation tool and decreases the number of potential *errors* that may occur when novices are forced to translate linguistic annotation schemes into formal markup languages.

5.7.10. Examples and prototypes (Step 10)

In their meta-study on the usability of usability recommendations, Molich et al. (2007, p. 178) come to the conclusion that it is most important to "do as you preach" and to "show a good example by making your usability recommendations useful and usable". Accordingly, this section of the pattern contains screenshots that illustrate the solution as a concrete example in an existing tool. It is important to note that these examples do not present the pattern's solution in an isolated way, but rather as part of the overall tool interface. Therefore, an example that illustrates a good solution may, nevertheless, be accompanied by usability problems that are related to other interface aspects.

The caption beneath the exemplary screenshot shows the name of the tool and provides a brief description of the screenshot as well as an explicit reference to the documented strength by means of its ID (e.g. S-ANA04). A number of screenshots has already been taken during the heuristic walkthrough. It is also possible to create specific screenshots during the process of pattern creation directly from the tools (all tools can still be accessed on the test system) or by using the detailed video documentations of the previous heuristic walkthroughs. For some patterns there may be several screenshots from different tools that illustrate slightly different implementations of the suggested solution. In some cases patterns may also contain several partial solutions, which are then illustrated by several examples as well.

Prototypes as a means of communication

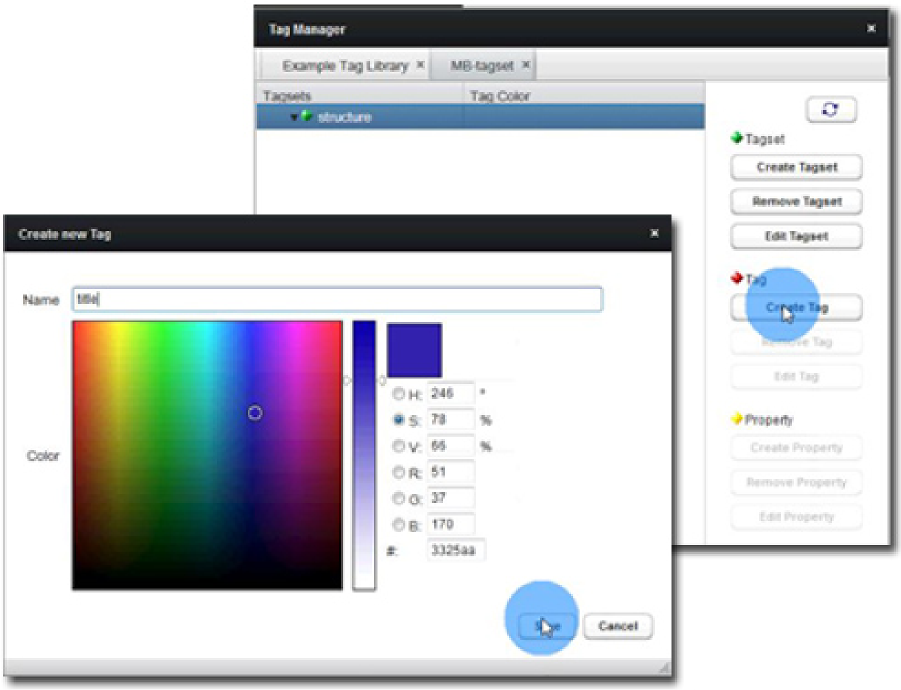
In the few cases (cf. patterns P3.1 and P6.3) where there is no positive example available in the tested tools, an interactive prototype that implements the proposed solution and that may serve as an example will be provided. In his practitioner's guide to prototyping, Warfel (2009, p. 3) emphasizes the power of prototypes to document and communicate design ideas: "If a picture is worth a thousand words, then a prototype is worth 10.000". For this reason, interactive prototypes are created as an addition to the static screenshot examples whenever it is technically feasible and whenever it brings additional benefit in understanding how the solution works. As the primary repository for the patterns is an online wiki, the prototypes are realized by means of HTML and JavaScript libraries (primarily jQuery / jQuery UI and in a few examples D3.js). Interactive HTML prototypes are the most elaborate and most expressive type of prototypes as compared to other types, such as *paper prototypes*, *Powerpoint / Keynote*, *Visio*, *Firework* or *Axure prototypes* (Warfel, 2009, p. 148)[179]. Warfel (2009, p. 151-152) describes HTML prototypes as the "Holy Grail" of prototyping due to a number of strengths:

[179] The comprehensive "Methods & Tools Comparison Matrix" is also available online on the Rosenfeld Media Flickr account: `https://www.flickr.com/photos/rosenfeldmedia/4000307751/in/set-72157622384497663/`

- Platform independency / portability
- Free to use / lots of free frameworks
- Modularity / reusable code
- Unlimited potential

Interactive annotation tool prototypes

A complete list of all interactive prototypes can be found in appendix E. The appendix also contains detailed information about the JavaScript libraries and plugins that were used for the implementation of each prototype. For this example pattern, a number of tools can be identified that implement the suggested solution of an integrated, non-XML annotation scheme editor (cf. Figure 5.7)[180]. As an addition, an interactive prototype for this pattern's solution illustrates the basic interaction steps of the pattern (cf. Figure 5.8).



Name: title

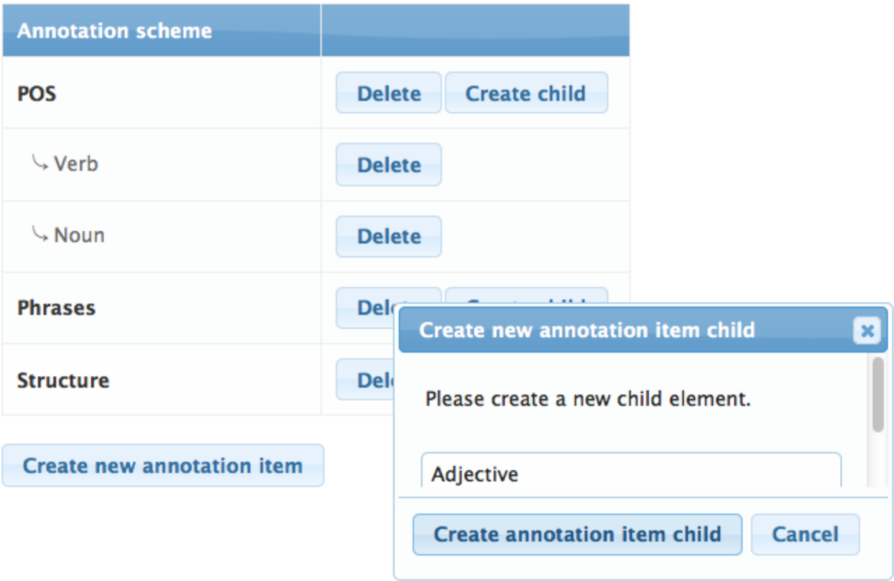
○ H: 246 °
● S: 78 %
○ V: 66 %
○ R: 51
○ G: 37
○ B: 170
#: 3325aa

Figure 5.7.: *CATMA* – Integrated annotation scheme editor (S-CAT03).

5.7.11. Optional counterexamples (Step 11)

The optional section counterexamples was left out in most patterns, as the information is largely redundant with the alphabetic list of concrete usability problems that is provided in the problem description section: All tools that are doc-

[180] For lack of space, only one of the five examples is described here. The complete collection of examples can, however, be found in the description of pattern *P4.1 – Integrated annotation scheme editor*.


Annotation scheme
POS
Delete
Create child
Verb
Delete
Noun
Delete
Phrases
Del
Structure
Del
Create new annotation item
Create new annotation item child
Create annotation item child
Cancel

Please create a new child element.

Adjective

Figure 5.8.: Prototype P4.1 – The prototype allows the user to create and delete annotation items on two different hierarchical levels.

umented with a usability problem for a specific pattern may as well serve as a counterexample. There are, however, a few cases where the documentation of counterexamples is reasonable. One of these cases is pattern P2.1 ("Easy installation and setup"): Three counterexamples were documented for this pattern, which were not part of the full evaluation procedure due to installation and setup problems.

5.7.12. Known uses (Step 12)

This section can be seen as an extension of the *examples* category, as it lists all tools that implement the solution of the pattern, but have not been mentioned in the examples section for means of reduced redundancy. In some case – though not in this example pattern – the *known uses* section may also contain examples for tools that have not been part of the heuristic walkthrough evaluation, but that are known from previous studies or from hints in related literature.

> **Known uses**:
> GATE, Knowtator

5.7.13. Related patterns (Step 13)

One of the strengths of the pattern format is the possibility to relate different patterns to each other, and thus create an interconnected repository for design knowledge. Relations may either point to a pattern from an external collection of HCI patterns, or to other patterns in the same collection.

External HCI pattern collections

In the domain of HCI, there are numerous pattern collections[181] that describe good design on various levels of abstraction, e.g. *concrete interaction elements*, *general user behaviors*, etc. As many of the patterns described in these collections are either redundant, or have a strong focus on web-based interfaces (cf. e.g. Van Duyne et al., 2006), the patterns presented in this dissertation only relate to two other pattern collections that are quite popular throughout the HCI community: The first collection is authored by Van Welie & Trætteberg (2000), who describe 20 "interaction patterns in user interfaces"[182]. The second collection was created by Tidwell (2011), who describes 125 "patterns for effective interaction design". Although most of these external patterns address rather generic usability issues (navigation, information architecture, etc.), some of them can be related to the domain-specific patterns for annotation tools (e.g. the "Wizard" or "Preview" patterns).

Relation types

According to Conte et al. (2002), pattern relations in general can be of the following four types[183]: P$_A$ *uses* P$_B$, P$_A$ *refines* P$_B$, P$_A$ *requires* P$_B$, P$_A$ is an *alternative* for P$_B$. All relations in the pattern collection will be created according to these basic types. For the example pattern P4.1, the following relations to internal as well as external patterns can be identified:

Related patterns

- Internal relations: This pattern uses P4.2
- External relations: This pattern uses the "Structured format" pattern (Tidwell 2011) and the "Input hints" pattern (Tidwell 2011)

5.8. Summary

This chapter has introduced patterns as a format for documenting design knowledge in a transparent and comprehensible way. While the identification of usability patterns is a rather vague process, it has been shown that the results of heuristic walkthrough evaluation studies integrate very well with the pattern format. The suggested pattern identification approach not only benefits from the heuristic walkthrough results, but also helps to avoid some of the problems

[181] On his website, Van Welie gives an overview of many other collections of HCI patterns (cf. `http://www.welie.com/index.php`).

[182] Van Welie also authors another popular collection of patterns that is available online at `http://www.welie.com/patterns/index.php`. This pattern collection, however, was not used in this work, as it focuses on web sites and web applications (navigation, site types, etc.).

[183] Van Welie & Van der Veer (2003, p. 2) also suggest three basic types of pattern relations that are inspired by relations known from the domain of OOP: *aggregation*, *specialization* and *association*. As the pattern collection described in this work does not contain any *aggregation* relations, the typology of Conte et al. (2002) was chosen to be more appropriate.

of traditional usability reports (cf. Jeffries, 1994). A detailed description of all patterns that could be identified on basis of the heuristic walkthrough data will be presented in the next chapter.

6. Usability patterns for annotation tools

6.1. Introduction

This chapter presents a collection of twenty-six usability patterns for the domain of linguistic annotation tools. The patterns were created according to the step-by-step guide described in the previous chapter, and are based on the heuristic walkthrough data that was presented in chapter 4. All data items that could not be integrated into a pattern will be discussed independently as well.

Chapter structure

The chapter is structured as follows: Section 6.2 describes the organization of the pattern collection and of the remaining evaluation results that could not be integrated into a pattern. Sections 6.3 (*General UI*), 6.4 (*Installation*), 6.5 (*Primary data*), 6.6 (*Annotation scheme*), 6.7 (*Annotation process*), and 6.8 (*Annotation visualization*) each describe specific patterns as well as remaining strengths and problems for the previously identified categories. Section 6.9 summarizes the whole chapter and leads over to the final chapter, which provides a summary and discussion of the main *contributions* of this work as well as an outlook to *future work*.

6.2. Preliminaries: Organization of the results

Pattern organization

In order to facilitate the selection of applicable patterns for a specific design scenario it is important to organize the patterns in a systematic and meaningful way (Van Welie & Van der Veer, 2003, p. 1). While there are several examples for the organization of HCI-patterns according to their level of detail (cf. Mahemoff & Johnston, 1998; Van Welie & Van der Veer, 2003), many pattern collections are organized by *function*[184] or *problem similarity* (Van Welie & Van der Veer, 2003, p. 6). The patterns in this work are also organized according to their function, or more specifically, according to the six domain-specific categories that were introduced in chapter 4.5.3. The following sections describe characteristics and challenges of these categories and relate to the identified patterns whenever appropriate.

[184] One of the most prominent examples for a functional organization of patterns can be found in the collection provided by Tidwell (2011).

Overall, a total of twenty-six usability patterns for linguistic annotation tools could be identified (cf. Table 6.1). These patterns are presented throughout the rest of this chapter, but can also be accessed online, via the *annotation usability wiki*[185]. In the following, patterns will be referenced by an abbreviated form such as *P3.2*. This abbreviation indicates that the referenced pattern is the *second* pattern from the *third* category (*Annotation scheme*).

Category	**Number of patterns**	**Abbreviation**
General UI	1	P1.1
Installation	1	P2.1
Primary data	7	P3.1 – P3.7
Annotation scheme	4	P4.1 – P4.4
Annotation process	10	P5.1 – P5.10
Annotation visualization	3	P6.1 – P6.3

Table 6.1.: Overview of all patterns according to their domain-specific category.

Remaining problems and strengths

Not all of the 207 usability problems and the 84 strengths identified during the heuristic walkthrough are suitable for the deduction of a pattern (cf. Figure 6.1)[186]. Many identified problems and strengths were too specific to be formulated as a generic design pattern that can be applied to a recurring problem situation. Nonetheless, these remaining problems (count: 125) and strengths (count: 22) are documented as well. Any heuristic walkthrough data item that is not part of a design pattern will be briefly discussed. For each category, the remaining items are structured in the following way:

Remaining strengths Individual strengths that could be observed for specific tools, which do not really solve serious usability problems, but which may be considered as nice-to-have features that might be worth implementing in a tool anyway. It is also possible to refine these strengths into patterns at a later point in time, when they can be related to a concrete problem context. Mostly, the remaining strengths are discussed as individual items. There are, however, some examples for similar strengths that will be discussed as a cluster.

[185] The patterns presented in this dissertation were exported from the HTML source of the wiki. Despite the creation of a specific print stylesheet, the formatting and appearance is still not optimal with regards to line breaks and text alignment. If possible, it is recommended to view the patterns in the wiki: `http://www.annotation-usability.net`.

[186] Note: P6.3 "Alternative visualizations of relational annotations" has not been incorporated in this figure, as it is the only pattern that was identified exclusively by using input from existing literature, i.e. no strengths or problems from the heuristic walkthrough evaluation were used as input for this particular pattern.

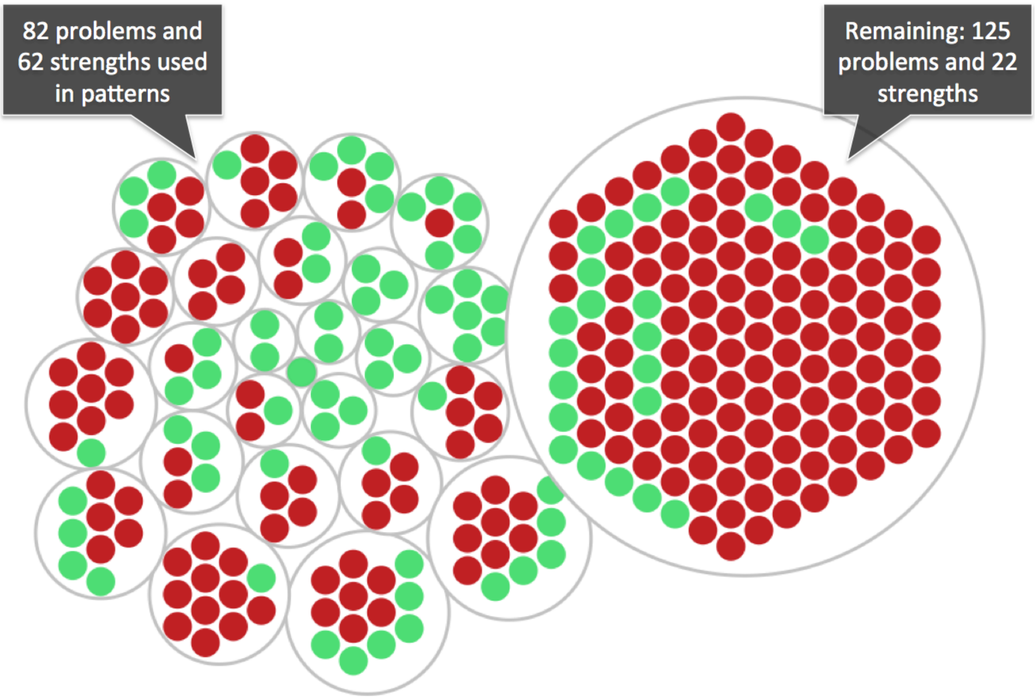


Figure 6.1.: Overview of all 207 problems (red circles) and 84 strengths (green circles), and their suitability as input for a usability pattern (small circles). The items that could not be used as input for a pattern are displayed in the large circle on the right side of the image.

Remaining problem clusters Problems that can be generalized, and that therefore are relevant for the whole domain of linguistic annotation tools, but which could not be resolved in a pattern. Typically, these problems are discussed as clusters of similar problems.

Tool-specific problems Finally, there is a large number of usability problems that cannot be generalized into a pattern, as they are rather tool-specific. Nevertheless, it is important to document these problems, as they allow the developers of a specific tool to work on individual improvements of their tool. At the same time, future tool developers can scan though this list, to get an overview of potential errors that can be made when designing an interface for a linguistic annotation tool. This type of remaining problems is discussed as tool-specific clusters (e.g. remaining problems for UAM Corpus Tool).

In the following sections, the concrete usability problems and strengths will be referenced by an abbreviated form consisting of a prefix that describes the annotation tool and a running index. Table 6.2 gives an overview of the abbreviations. A comprehensive list of all problems and strengths can be found in appendix D.

Annotation tool name	Prefix for problem	Prefix for strength
Aanalec	ANA	S-ANA
Brat	BRA	S-BRA
CATMA	CAT	S-CAT
Dexter	DEX	S-DEX
GATE	GAT	S-GAT
Glozz	GLO	S-GLO
MMAX-2	MAX	S-MAX
Knowtator	KNW	S-KNW
UAM Corpus Tool	UAM	S-UAM
WebAnno	WEB	S-WEB
WordFreak	WOR	S-WOR

Table 6.2.: Overview of abbreviated forms that are used for referencing usability problems and strengths.

The following two examples illustrate how the abbreviations come about:

Example A *ANA02* refers to the *second* usability problem that has been documented for the annotation tool *Analec*.

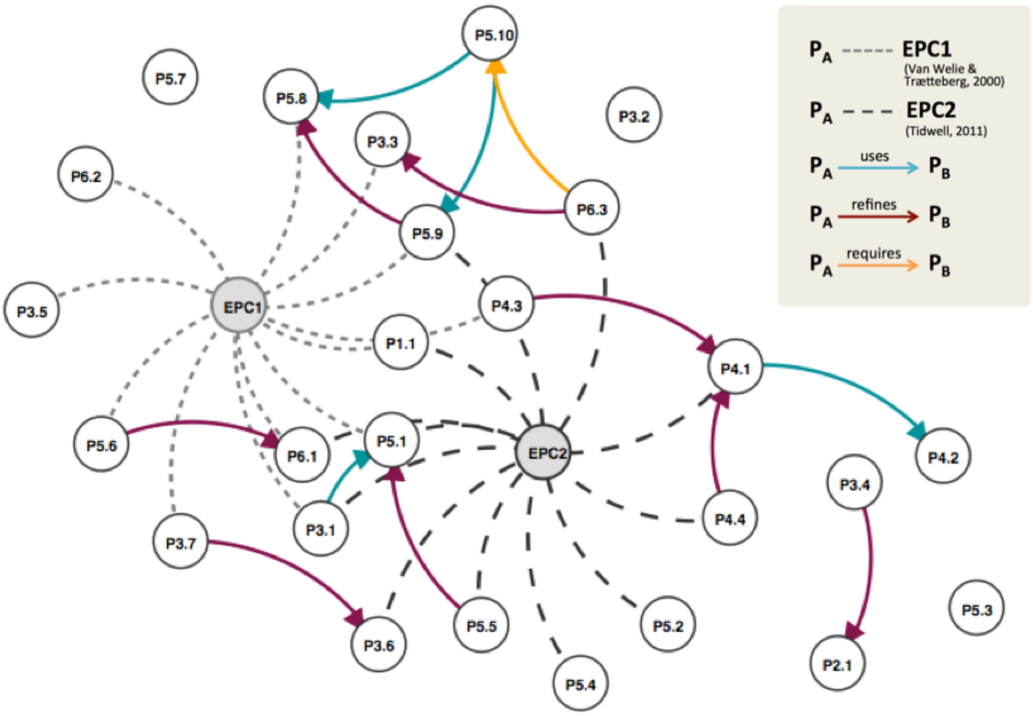
Example B *S-KNW05* refers to the *fifth* tool-specific *strength* that has been documented for the annotation tool *Knowtator*.

Relations between patterns

So far, the pattern collection has been organized according to different functional categories. While these categories provide a more structured, hierarchical access to the patterns, it is also possible to navigate through the collection by means of the *relations* that are defined between different patterns. In the wiki, relations are realized as hyperlinks, to create a pattern collection that can be experienced as an interconnected hypertext. As was described in chapter 5.7.13, there may be relations between patterns inside the collection of annotation tool patterns, but also to the external pattern collections of Van Welie & Trætteberg (2000) and Tidwell (2011). Figure 6.2 shows the internal relations between the different patterns by means of colored, unidirectional arrows. Each color represents a particular type of relation[187] (cf. the legend at the top of Figure 6.2).

As it would bloat the figure to view the exact relations between internal patterns and patterns from the two external collections, relations are only displayed for the whole external pattern collections (EPC1 and EPC2). A more detailed description of the relations to concrete, external patterns, will be given in the respective patterns.

[187] Note: The relation type "alternative" is not used in this pattern collection: If multiple, alternative solutions exist for the same problem, these solutions are summarized in one pattern. There are also no alternative patterns in the external collections, as these patterns are too generic to be used directly in the specific domain of linguistic annotation tools.



Pattern category: General UI
P1.1 Help for domain-specific functions

Pattern category: Installation
P2.1 Easy installation and setup

Pattern category: Primary data
P3.1 Guided pre-processing of primary data
P3.2 Separated import of documents and mapping to corpora
P3.3 Easy navigation between multiple primary data documents
P3.4 Automatic creation of annotation files
P3.5 Mode for editing primary data
P3.6 Support for reading primary data
P3.7 Tailored display of primary data

Pattern category: Annotation scheme
P4.1 Integrated annotation scheme editor
P4.2 Organization of annotation scheme
P4.3 Tailored display of annotation schemes
P4.4 Facilitated distinction of multiple annotation items

Pattern category: Annotation process
P5.1 Semi-automatic creation of anchors
P5.2 Conventionalized actions for creation of anchors
P5.3 Feedback for creation of anchor scope
P5.4 Accelerators for selecting and applying annotations
P5.5 Accelerators for annotating multiple anchors
P5.6 Access to existing annotations
P5.7 Assistance for modification of anchors
P5.8 Integrated delete annotation function
P5.9 Mass deletion of annotations
P5.10 Support annotation of relations

Pattern category: Annotation visualization
P6.1 Visualization of parallel annotations
P6.2 Tailored display of annotations
P6.3 Alternative visualizations of relational annotations

Figure 6.2.: Relations between patterns.

6.3. General UI

The heuristic walkthrough evaluation was intended to reveal usability problems that are specific for the domain of annotation tools. For this reason, general usability problems were largely excluded from the evaluation. There are, however, some problems that influence the overall user experience with the tool, with no explicit reference to any particular stage in the annotation workflow. These problems are collected in the category "General UI". It becomes obvious that many identified problems are related to the issue of providing rather domain-specific functions that are not known to the user from previous experience with other tools (P1.1).

Most of the remaining problems could be subsumed to larger problem clusters (e.g. "bad wording" or "redundant controls"). There are no generic solutions for these issues that could be expressed in the form of a pattern specific to linguistic annotation tools. These problems are nevertheless discussed in the following paragraphs. Solutions for this class of general UI problems can be found in existing pattern collections by Tidwell (2011) and Van Welie & Trætteberg (2000). The following selection of patterns from Tidwell (2011, chap. 1) seems to be particularly helpful to fix general UI problems for annotation tools, as the patterns describe typical characteristics and behaviors of tool users:

Safe exploration "Let me explore without getting lost or getting into trouble."

Instant gratification "I want to accomplish something now, not later."

Satisficing "This is good enough. I don't want to spend more time learning to do it better."

Changes in midstream "I changed my mind about what I was doing."

Deferred choices "I don't want to answer that now; just let me finish!"

Incremental construction "Let me change this. That doesn't look right; let me change it again. That's better."

Habituation "That gesture works everywhere else; why doesn't it work here, too?"

Spatial memory "I swear that button was here a minute ago. Where did it go?"

Prospective memory "I'm putting this here to remind myself to deal with it later."

Streamlined repetition "I have to repeat this how many times?"

Keyboard only "Please don't make me use the mouse."

6.3.1. Usability patterns

Only one usability pattern could be identified for this category:

P1.1 Help for domain-specific functions

Overview of the usability problems and tool-specific strengths that were used as input for the pattern:

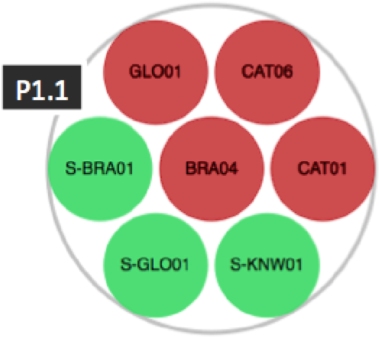


Figure 6.3.: Usability problems and strengths used as input for patterns in category "General UI".

P1.1 - Help for domain-specific functions

Problem description

Linguistic annotation tools come with many domain-specific functions that are largely unknown to novice users. The tool needs to provide help for novice users, so they can easily understand and learn these functions.

Related problems: BRA04, CAT01, CAT06, GLO01

Usability principle

Error prevention, recognition rather than recall

Context

This pattern can be used to provide help and explanation for domain-specific functions that are not known by novice users and that are not understood intuitively. The solutions described in this pattern can be used for functions with different degrees of complexity.

Forces

While annotation tools make use of many functions that are already known from other types of software (e.g. *import document*, *save document*, etc.), there is also a great number of domain-specific functions (e.g. *create coreference annotation*) that is not known to novice users. Being *domain-specific*, these functions cannot build on conventions and previous experience with other tools. It may, however, be necessary to use domain-specific wording and metaphors to describe these functions in the interface, although they are hard to grasp for unexperienced users.

Solution

This pattern describes three solutions that are not meant to be mutually exclusive.

Solution A: Provide a short explanation for domain-specific functions inside the tool. This can be realized by means of tooltips, if the function is fairly easy to explain. In case the explanation of a function is more complex, a short introductory dialog opens when the function is first started. If the function requires multiple substeps, user guidance by means of a wizard is provided.

Solution B: If there are several functions that build on each other, it might also be helpful to provide a quick interactive tutorial once the tool is started for the first time.

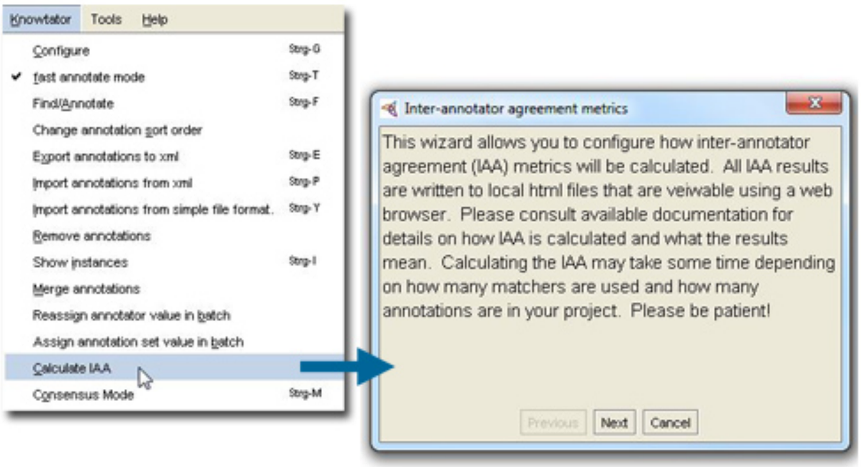
Solution C: If a function requires prerequisites before it can be used, providing a "sandbox" with existing examples (that meet those prerequisites) is a good way to convey how that function works, and what can be achieved with it.

Rationale

Providing help for rather unintuitive, domain-specific functions, increases the *learnability* of the tool. This is particularly important for novice users, who may also be lacking knowledge about the domain of linguistic annotation tools.

Example

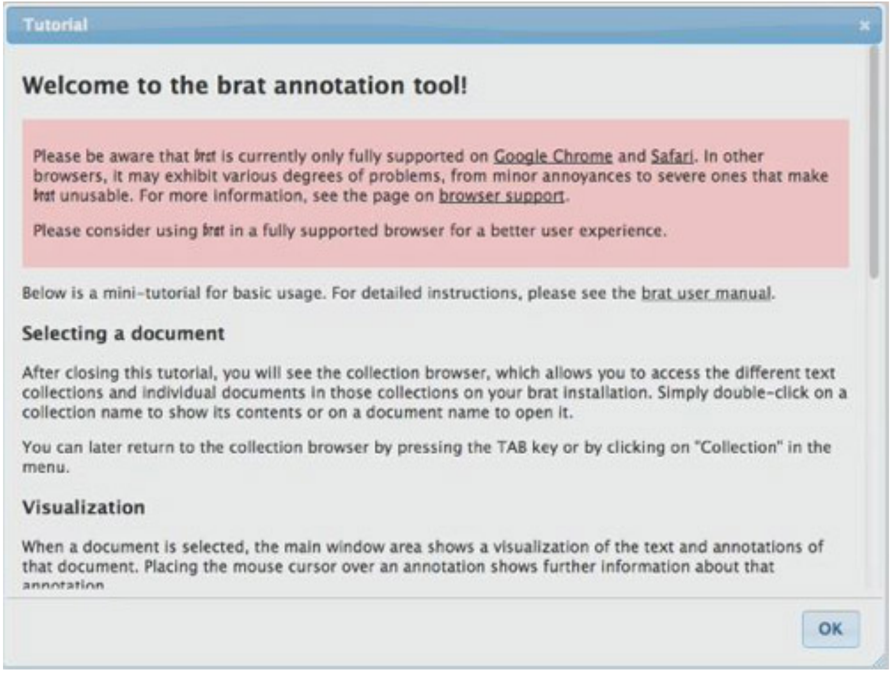
Example for solution A


Knowtator
Tools
Help
Configure Strg-G
✔ fast annotate mode Strg-T
Find/Annotate Strg-F
Change annotation sort order
Export annotations to xml Strg-E
Import annotations from xml Strg-P
Import annotations from simple file format. Strg-Y
Remove annotations
Show instances Strg-I
Merge annotations
Reassign annotator value in batch
Assign annotation set value in batch
Calculate IAA
Consensus Mode Strg-M
Inter-annotator agreement metrics
Previous
Next
Cancel

This wizard allows you to configure how inter-annotator agreement (IAA) metrics will be calculated. All IAA results are written to local html files that are veiwable using a web browser. Please consult available documentation for details on how IAA is calculated and what the results mean. Calculating the IAA may take some time depending on how many matchers are used and how many annotations are in your project. Please be patient!

Knowtator: Complex functions are explained by means of a short introductory dialog that is followed up by a wizard (S-KNW01).

Example for solution B



Tutorial

Welcome to the brat annotation tool!

Please be aware that brat is currently only fully supported on Google Chrome and Safari. In other browsers, it may exhibit various degrees of problems, from minor annoyances to severe ones that make brat unusable. For more information, see the page on browser support.

Please consider using brat in a fully supported browser for a better user experience.

Below is a mini-tutorial for basic usage. For detailed instructions, please see the brat user manual.

Selecting a document

After closing this tutorial, you will see the collection browser, which allows you to access the different text collections and individual documents in those collections on your brat installation. Simply double-click on a collection name to show its contents or on a document name to open it.

You can later return to the collection browser by pressing the TAB key or by clicking on "Collection" in the menu.

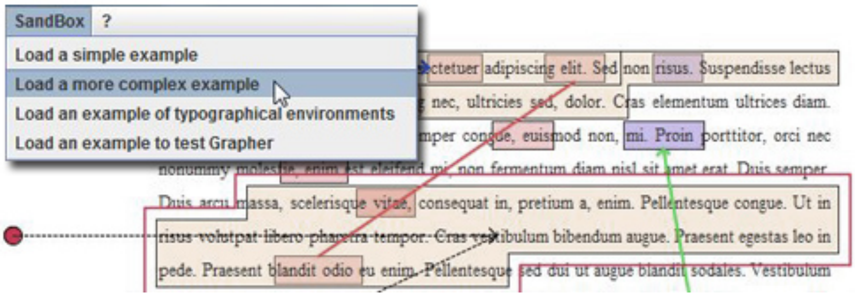
Visualization

When a document is selected, the main window area shows a visualization of the text and annotations of that document. Placing the mouse cursor over an annotation shows further information about that annotation

OK

Brat: The most important functions are explained when the tool is first used (S-BRA01).

Example for solution C


SandBox
?
Load a simple example
Load a more complex example
Load an example of typographical environments
Load an example to test Grapher

Glozz: A sandbox allows the user to explore different functions on the basis of existing examples (S-GLO01).

Related patterns

- Internal relations: -
- External relations: This pattern uses the "Wizard" pattern (Van Welie & Trætteberg 2000; Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns /pmwiki.php?n=PmWiki.HelpForDomain-specificFunctions
Page last modified on September 15, 2014, at 09:02 PM EST

6.3.2. List of remaining problem clusters

Cumbersome wording and metaphors The overall wording of functions and controls of many linguistic annotation tools does not adhere to established conventions, but rather uses technical or domain-specific terms that are not easily understood by novice users (BRA05, GAT02, GAT03, UAM04). If possible, conventionalized terminology and theory-neutral, easy-to-grasp metaphors should be used (also cf. P1.1).

Unconventional user interface design Some tools do not adhere to basic design conventions, such as layout of the interface (MAX02), or design of interaction elements, such as buttons (UAM01). Unconventional interface designs slow down the learnability and comprehensibility of an annotation tool. A more specific example of an unconventional UI design can be found with *Brat* and *WebAnno*: While typically, the main navigation, which holds most of the important functions, is visible and easy to access at all times, *Brat* does hide the navigation bar in the top of the screen by default (BRA03). The navigation should rather be visible by default, with an optional feature to hide it, in case it is not needed. In order to access the main navigation in *WebAnno* (WEB03), the user has to switch from an annotation screen to a screen that holds the main navigation items. While switching screens by means of hyperlinks or Browser navigation controls may be a common interaction behavior for web pages, it is not intuitive and efficient for a web-based annotation tool. The main navigation should rather be positioned as a horizontal menu structure on top of the screen.

Redundant controls Some annotation tools try to make up for cumbersome interface design by providing certain functions and controls redundantly. *UAM Corpus Tool* for instance provides multiple help buttons on different positions of the interface, which are all visible at once (UAM02). This is rather irritating for the user, as he cannot be sure if different help buttons contain different help content. Another example for redundant functions that are rather irritating than helping the user are to be found in *WordFreak* (WOR02).

No explicit save action *Brat* and *WebAnno* are both realized as web-based tools that are accessed via a web browser. As a result, there is no way to explicitly save any progress during the annotation process explicitly, each action is rather stored automatically (WEB02, BRA02). While this seems to be a useful feature at first glance, novice users may be irritated about their progress being stored automatically, as there is no explicit feedback from the system. The irritation gets even bigger when users switch or reload a screen (cf. WEB03), as they may be afraid that their progress is lost. In order to avoid these uncertainties, the tool should provide a manual save button (while still saving progress in the background), or indicate the ongoing automatic saving by means of an unobtrusive message text or animation at the bottom of the screen.

Insufficient documentation Due to a number of domain-specific functions, good documentation is an important requirement for any annotation tool. While some tools do not have a manual at all (ANA03), other tools provide only insufficient documentation that does not cover all relevant issues (BRA08, WOR01). Another related problem in this area are documentations and help functions that make use of advanced terminology, i.e. for novice users it is not possible to look up specific issues in the documentation, because they are lacking the technical terms (UAM03).

6.3.3. List of tool-specific problems

Analec Some functions in *Analec* appear in the tool's menu structure, but are not actually available (ANA01): When the user clicks on such a function, a message appears, stating that this function is not implemented yet. Another issue with *Analec* is, that it is only available in French language (ANA02). There should be a possibility to adjust the language settings of the tool to English.

Brat While tooltips are generally useful to provide a short explanation or hint for an existing function, the user may be irritated if tooltips are used inconsistently, i.e. if they are not available for all functions (BRA06). Another problem that is also related to the area of "user feedback / help" is present in the error messages that appear on the bottom of the screen, but disappear too quickly to be read carefully (BRA07). Seeing an error message, but not being able to read its whole content, may leave the user in a state of uncertainty. The little blue button that is labeled "i" does not convey the function that it can be used to show those messages again (BRA07). Also, the interaction behavior of "hovering" over this button-like element is irritating, as buttons are typically clicked to perform some specified action (BRA07).

CATMA Although *CATMA* is a web-based tool, it is trying very hard to look like a desktop application. The multiple-windows interface design (each module opens in a new window) is the most obvious desktop metaphor. However, this kind of interface does not feel natural for a web application, as the user is still aware that he is using a web browser to display the annotation tool. This becomes particularly noticeable when the windows are dragged or resized, as such an action can cause significant delays (CAT02). Another minor bug is related to the double clicking behavior for documents and tag sets: while a double click is intended to open the items, it does not work reliably (CAT05).

On some occasions small, textual hints on how to proceed with the tool appear in the bottom left corner. Although the idea of guiding the user with short message texts is good, the position of the messages is not in the focus of the user. Also, the messages disappear very fast, which makes it hard to read the whole message (CAT03). Similar to the problem described for *Brat* (BRA07), such messages tend to create insecurity on the user side instead of guiding the user through the annotation process. Along the same lines, *CATMA* provides a number of small icons that are labeled with a question mark. These icons are very small and they are scattered among the whole interface, which makes them hard to perceive. While they look like small buttons, they cannot be clicked, but rather reveal generic hints when the user hovers over them. Such hints should be more specific, and they should be associated with a specific UI element, i.e. the hint should explain the function of a specific UI element.

GATE The datastore metaphor and its purpose, which is to save a project in an external datastore, is not intuitive (GAT01). Also, the process of creating a datastore is unclear. The user typically would expect to be able to save the current project via a "Save" option in the file menu.

Knowtator Whenever the user changes settings or properties of a current *Knowtator* project, this is done via a dialog window. It is, however, not possible to explicitly apply the changes by means of a "confirm" button or the like, but rather the settings are applied when the dialog window is closed (KNW02). The window is closed by clicking on a red "x" icon, which is positioned in the top left corner. The color and label of this button does not convey the impression that the settings are saved; there should rather be some sort of explicit feedback or confirmation mechanism, to avoid uncertainties on the user side. As *Knowtator* is realized as a plugin for the ontology modeling platform *Protégé*, it makes use of existing *Protégé* concepts such as "classes" and "slots". It is likely that these concepts are hard to grasp and hard to differentiate for the standard linguistic annotator (KNW03).

Like many other tools, *Knowtator* allows the user to customize the font size of the primary data (cf. P3.7). However, the font size changes are applied globally to all menus and controls in the whole tool, which may be irritating for the user (KNW04). Another issue is concerned with the visibility of UI elements: The *Protégé* logo on the right covers some of the controls when the main window is displayed in full screen mode (test setting: 1280×1024 pixels); the arrow controls for navigating between annotations are hardly visible (KNW05). These arrow controls entail another problem: In total, there are three different pairs of arrow controls, which all look very similar, but in fact have quite different functions (KNW06). The first is for switching between primary data documents, the second is for switching between visualization filters, and the third is for navigating between annotations. The icons should be designed in a way the user can differentiate their respective function more clearly.

MMAX2 *MMAX2* is started as a Java program. After it is started, a console window opens that gives feedback about the actions executed in the GUI. This window may not be closed; the only way to avoid it is to put it somewhere in the background (MAX01). This can be very annoying and irritating for the user.

WebAnno The "open documents" dialog in *WebAnno* can be resized by means of rather small, red arrows. These arrows are hardly visible and do not adhere to conventionalized interaction behaviors for resizing a window (WEB05). This is an important issue, as resizing the window is mandatory in order to solve the problem described in WEB09 (cf. chapter 6.5).

6.4. Installation

An easy-to-install tool is vital, keeping in mind that the user group that was identified for this evaluation scenario (cf. chapter 4.4.1) typically has no experience with command line tools, programming languages, client-server architectures, database configuration, etc. While most of the tested tools were rather easy to install and setup (P2.1), there remains a number of tools that were excluded from the evaluation because of the cumbersome setup process (cf. chapter 4.5.5).

6.4.1. Usability patterns

Only one usability pattern could be identified for this category:

P2.1 Easy installation and setup

Overview of the usability problems and tool-specific strengths that were used as input for the pattern:

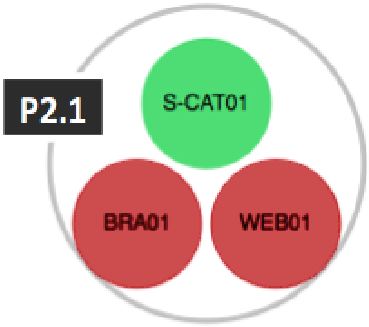


Figure 6.4.: Usability problems and strengths used as input for patterns in category "Installation".

P2.1 - Easy installation and setup

Problem description

The installation and setup of some annotation tools requires advanced technical skills, which are typically not available for linguistic annotators with a *traditional* humanities background.

Related problems: BRA01, WEB01

Usability principle

Error prevention

Context

This pattern can be used to provide an easy-to-accomplish installation and setup of an annotationt tool that does not require advanced technical skills.

Forces

Some annotation tools require basic technical infrastructure such as a database, a web server, or a configured version of a programming language (e.g. *Python*). Although the use of an annotation tool facilitates the work of linguists in the long term, the initial setup of the tool is oftentimes hard to achieve for users with only basic technical skills, and can thus be seen as an entry obstacle for *traditional humanists*.

Solution

This pattern describes two solutions that are not meant to be mutually exclusive:

Solution A: Provide a one-click-installer solution for multiple platforms (Windows, Linux, Mac). This can be best achieved by aggregating the necessary program files into a platform-independent *Java Archive* (JAR), as Java can be assumed to be available on most systems. However, make sure to add a hint in the readme file or documentation that decribes how to obtain Java from the web. Alternatively, *batch* (*.bat) or *executable* (*.exe) files can be used to realize a one-click-installation. Avoid databases as a means to store annotations, as they are difficult to set up for novice users. Rather store annotations in an XML-based file format.

Solution B: Design the annotation tool as a web service that can be used by multiple clients. Although a web-based approach most likely requires a web server and a database, the user does not have to worry about the technical infrastructure, as he can use the tool as a service.

Rationale

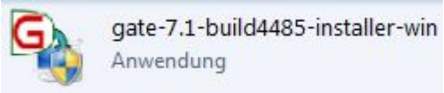
By providing a ready-to-use / one-click-installation tool, the potential of *errors* that might occur during the installation and setup process is reduced significantly.

Example

Examples for solution A

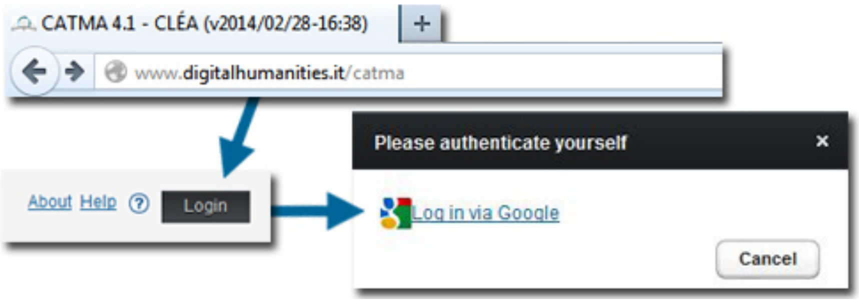
WordFreak: JAR installer.

Glozz: Batch installer.



GATE: Executable installer.

Example for solution B


CATMA 4.1 - CLÉA (v2014/02/28-16:38)
+
www.digitalhumanities.it/catma
About
Help
Login
Please authenticate yourself
×
Log in via Google
Cancel

CATMA: Annotation tool as a web service (authentification required) (S-CAT01).

Counterexample

Anafora, Djangology, Slate (these tools were not part of the full evaluation procedure due to installation and setup problems)

Known uses

Solution A: Analec, Dexter, Knowtator, MMAX2, UAM Corpus Tool

Related patterns

- Internal relations: -
- External relations: -

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.EasyInstallationAndSetup
Page last modified on June 20, 2014, at 06:05 PM EST

6.4.2. List of tool-specific problems

Knowtator *Knowtator* is not designed as a stand-alone tool, but rather as a plugin for an existing framework, i.e. *Protégé* needs to be installed before *Knowtator* can be set up. While the setup process of both, the framework and the plugin, are documented very well, there is an issue with "activating" *Knowtator*: Any plugin needs to be activated in the *Protégé* framework before it can be actually used. However, the *Knowtator* plugin cannot be activated unless the user has created and saved a *Protégé* project in advance (KNW01). This interaction behavior is counterintuitive, and may irritate the user, as he does not know why the plugin cannot be activated.

6.5. Primary data

Before the annotation process starts, the primary data documents have to be imported to the annotation tool. Typically, annotation tools require some kind of pre-processing of that data, e.g. tokenization or sentence splitting. An annotation tool should guide novice users through the process of setting the pre-processing parameters and also make clear how the settings affect the primary data (P3.1). The primary data that is about to be annotated is usually spread across multiple documents, which are composed as a corpus. The creation of a corpus, as a larger meta-structure for single text documents, should be part of the data import process (P3.2). At the same time, the user should be able to navigate between specific primary data documents during the annotation process (P3.3). Once the primary data has been imported to the tool, an annotation file has to be created that allows the user to store any annotations that are applied to the primary data (P3.4). As many tools use character positions to link externally stored annotations (cf. the stand-off annotation approach as described in chapter 3.4.1) to the primary data, editing the primary data during the annotation process may have technical consequences for the storage of annotations (P3.5). The two last aspects are concerned with the appearance of the primary data in the annotation tool: During the annotation process, the textual primary data is typically rather scanned than read sequentially from beginning to end. An annotation tool should present the primary data in a way that facilitates this type of reading (P3.6). The annotator should also be able to tailor the display of the primary data according to his personal reading preferences (P3.7).

6.5.1. Usability patterns

Seven usability patterns were identified for this category:

P3.1 Guided pre-processing of primary data

P3.2 Separated import of documents and mapping to corpora

P3.3 Easy navigation between multiple primary data documents

P3.4 Automatic creation of annotation files

P3.5 Mode for editing primary data

P3.6 Support for reading primary data

P3.7 Tailored display of primary data

Overview of the usability problems and tool-specific strengths that were used as input for the patterns:

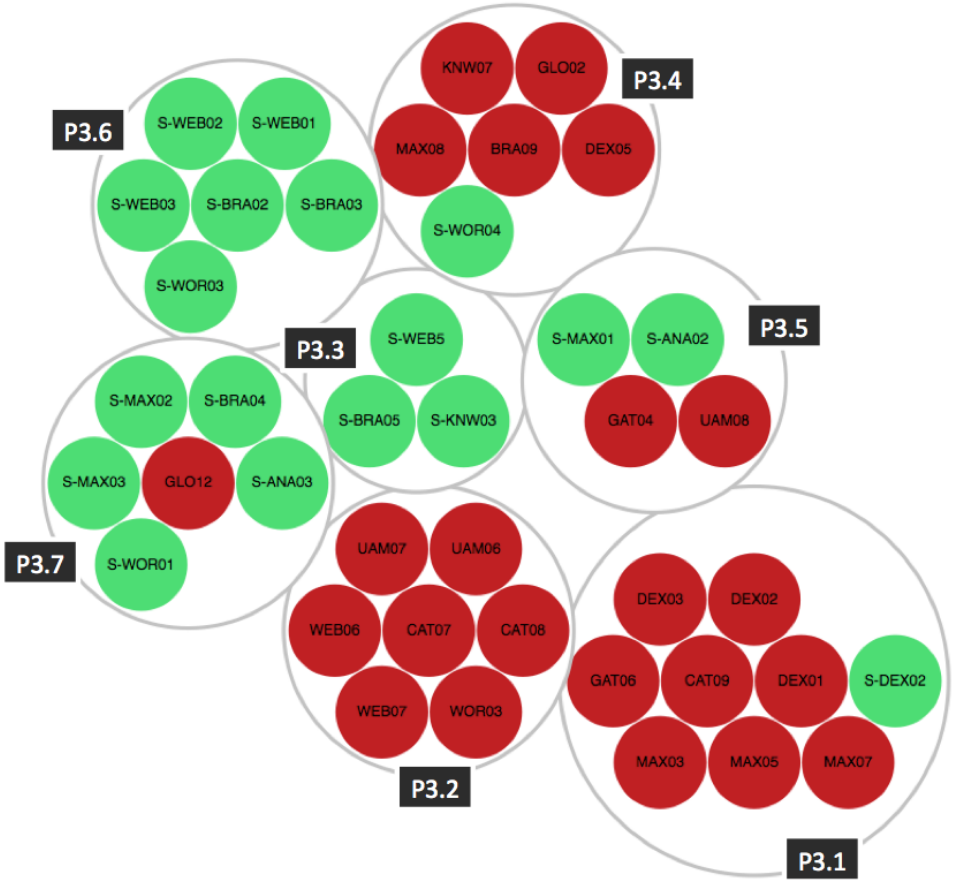


Figure 6.5.: Usability problems and strengths used as input for patterns in category "Primary data".

P3.1 - Guided pre-processing of primary data

Problem description

The user wants to import a text document in order to annotate it, but the tool requires to pre-process the data first. Users without technical knowledge about document markup may not know all available pre-processing parameters, and how they influence the primary data and the later annotation process.

Related problems: DEX01, DEX02, DEX03, CAT09, GAT06, MAX03, MAX05, MAX07

Usability principle

Error prevention, recognition rather than recall

Context

This pattern can be used to facilitate the pre-processing of the primary data. Many annotation tools require the primary data to be in a specific file format or structure. In this case, there should be a tool component that guides the user through the necessary pre-processing steps, which may be very specific and complex.

Forces

- The tool requires the primary data to be in a specific format in order to create annotations for the document.
- Users are not familiar with the required file format (especially if it is a proprietary format).
- Users are not familiar with the required document structure or segmentation (tokenization, sentence splitting, etc.)
- Users are not interested in pre-processing the data, because they are not aware of the benefits.
- Users are unsure whether a pre-processing component manipulates the primary data in an inadvertent way.
- Users want to pre-process the primary data in a way the tool does not support.

Solution

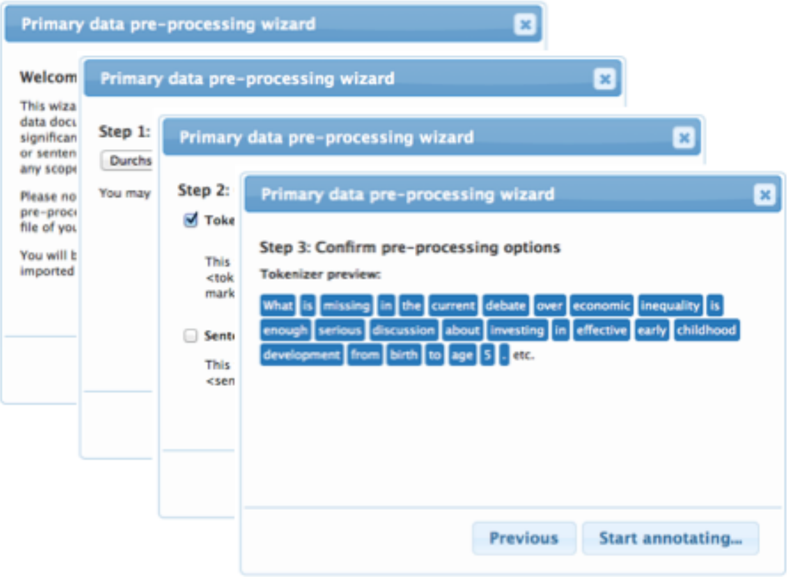
If a tool requires pre-processing of the primary data, it provides a guided import dialog with several steps: After an existing primary data document has been imported, the tool makes clear why pre-processing of the primary data is necessary, and how it influences the primary data and the actual annotation process in general. The tool provides a set of options from which the user can choose from, and also makes clear how each of these options influences the primary data and the actual annotation process. The options are formulated in an intuitive, theory-neutral way. After the options have been set, the tool previews the data in a way the user recognizes the effect of the pre-processing (cf. S-DEX02). Finally, the user can choose if he wants to edit the pre-processing options once again, or if he wants to import the pre-processed data into the annotation tool. As users are guided through the pre-processing of primary data, the pre-processing options and their effect on the data are transparent for the user.

Rationale

Guided pre-processing of primary data helps to prevent errors (*error rate*) and increases the *learnability* of a tool by giving explicit feedback on the available options and their consequences.

Example

Interactive prototype



Step 3: Confirm pre-processing options

Tokenizer preview:

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5 . etc.

Prototype P3.1 - The prototype implements a basic wizard that guides the user through the import and pre-processing process of primary data documents.

Known uses

Cf. UAM Corpus Tool and CATMA for tools that pick up the idea of a wizard-like, guided process for importing and pre-processing primary data

Related patterns

- Internal relations: This pattern uses P5.1
- External relations: This pattern uses the "Wizard" pattern (Van Welie & Trætteberg 2000; Tidwell 2011) and the "Preview" pattern (Van Welie & Trætteberg 2000; Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns/pmwiki.php?n=PmWiki.GuidedPreprocessingOfPrimaryData
Page last modified on September 12, 2014, at 05:47 PM EST

P3.2 - Separated import of documents and mapping to corpora

Problem description

A typical annotation project consists of multiple single documents, which are usually composed as a corpus. The novice user may not think about such a larger structural unit when importing the first, single document, but the tool may require it to be able to store and manage imported primary data properly.

Related problems: CAT07, CAT08, UAM06, UAM07, WEB06, WEB07, WOR03

Usability principle

User control and freedom, error prevention

Context

This pattern can be used when the annotation tools provides a macro structure, such as a *corpus* or a *project*, to combine multiple primary data documents.

Forces

For means of tool exploration, novice users typically want to add only a single document to the tool at first. At the same time, many annotation tools require the definition of a larger macro structure (like a corpus or a project), to which single documents have to be assigned. The necessity of a larger macro structure is not always obvious for the user at the beginning of an annotation project. In fact, users oftentimes will realize the necessity of a "corpus" macro unit only after they have imported and annotated several documents.

Solution

The import of one or more primary data documents and their assignment to a *corpus* or *project* macro structure are implemented as independent processes. The user can import and annotate arbitrary documents in the tool, and may assign them to a corpus at any time. A user-friendly annotation tool offers several mechanisms to assign documents to a corpus:

- When the user creates a new corpus, he can choose which documents he wants to include initially from a list of all primary data documents that already have been imported to the tool.

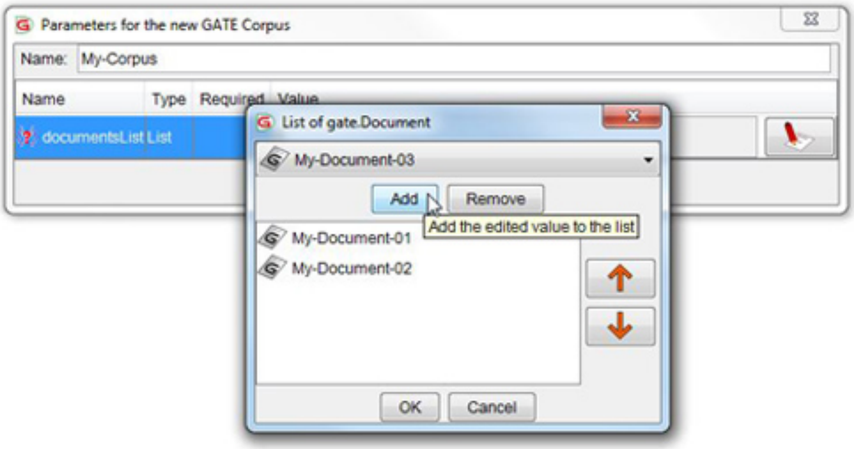
- The user can add or delete documents from the corpus or assign them to another corpus at any time.

Rationale

By separating the import of primary data documents and their assignment to a corpus structure, the user can freely choose when to map a document to a corpus. The overall *error potential* is reduced, as there is no necessity to create a corpus or project structure in advance.

Example

Example 1



Parameters for the new GATE Corpus

Name: My-Corpus

Name	Type	Required	Value
documentsList	List		

List of gate.Document

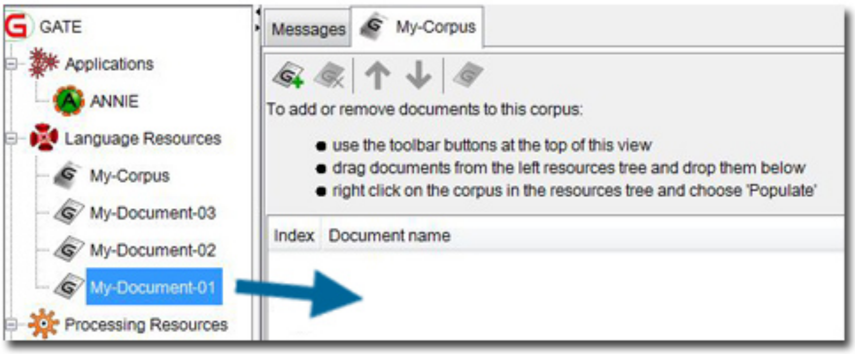
My-Document-03

Add the edited value to the list

My-Document-01
My-Document-02

GATE: Primary data documents can be added when the corpus is created.

Example 2



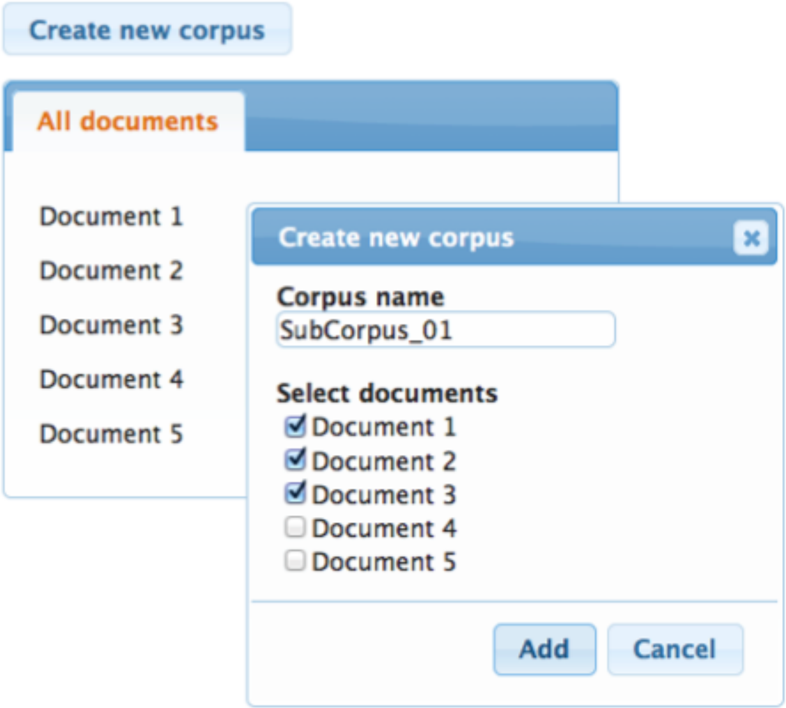
To add or remove documents to this corpus:

- use the toolbar buttons at the top of this view
- drag documents from the left resources tree and drop them below
- right click on the corpus in the resources tree and choose 'Populate'

Index	Document name

GATE: Primary data documents can be added to an existing corpus by means of drag-and-drop or by using an "add" dialog.

Interactive prototype



Corpus name
SubCorpus_01

Select documents
☑ Document 1
☑ Document 2
☑ Document 3
□ Document 4
□ Document 5

Prototype P3.2 - The prototype allows the user to create and delete corpora, and to relate a set of five existing documents to the corpus when it is created.

Related patterns

- Internal relations: -
- External relations: -

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns /pmwiki.php?n=PmWiki.SeparatedImportOfDocumentsAndMappingToCorpora
Page last modified on September 12, 2014, at 05:55 PM EST

P3.3 - Easy navigation between multiple primary data documents

Problem description

During the annotation process, the user wants to annotate different primary data documents. Therefore, it is necessary to be able to switch between different documents during the annotation process without losing the specific annotation progress for each text.

Related problems: -

Usability principle

User control and freedom, flexibility and efficiency of use

Context

This pattern can be used to provide a mechanism that allows users to switch between different primary data documents in an effective, unobstrusive way.

Forces

- An annotation tool can only display one primary data document at a time.
- For some annotation tasks, the user needs to switch between different primary data documents during the annotation process (inter- vs. intra-document annotations).

Solution

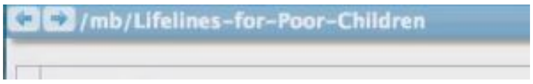
The tool provides direct access to multiple primary data documents during the annotation process. All documents are visible in a list view that can be opened with a single click. The user can navigate between different documents, either by means of arrow buttons (jump to previous or next document), or by selecting the desired document directly from the list view. If the user selects another primary data document, the annotations of the previously edited document are saved automatically. When the user goes back to the previous document, it is displayed in the same fashion as before (same line position, same annotations, etc.).

Rationale

By providing an efficient mechanism to switch between different primary data documents, the overall *efficiency* of the annotation tool is increased.

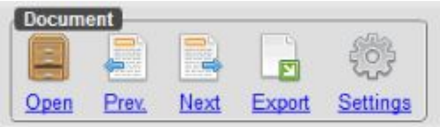
Example

Example 1



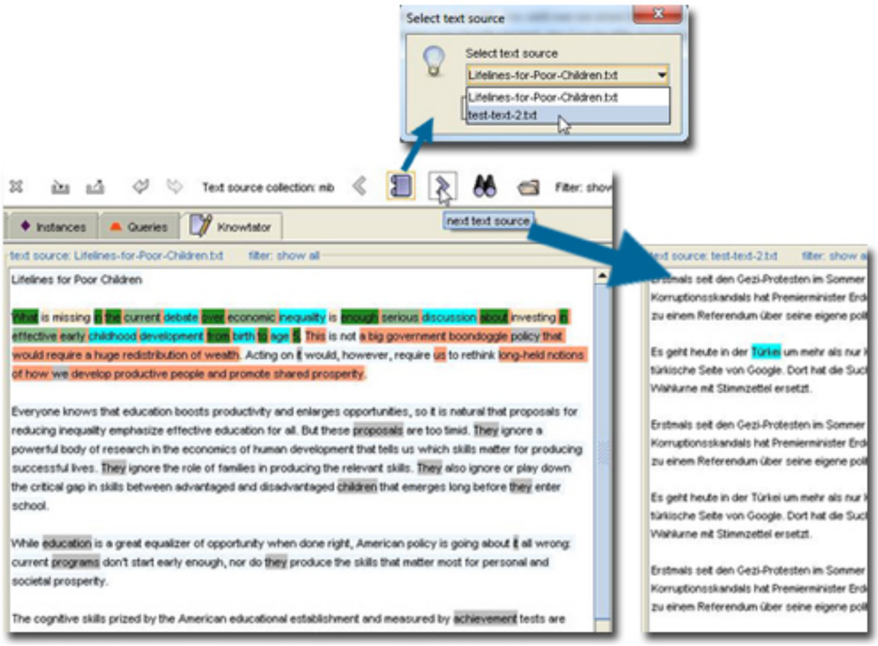
Brat: Arrow controls can be used to switch between different primary data documents (S-BRA05).

Example 2



WebAnno: Arrow controls (*Prev.* / *Next*) can be used to switch between different primary data documents (S-WEB05).

Example 3



text source: Lifelines-for-Poor-Children.txt filter: show all

Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

The cognitive skills prized by the American educational establishment and measured by achievement tests are

text source: test-text-2.txt filter: show a

stmals seit den Gezi-Protesten im Sommer
Korruptionsskandals hat Premierminister Erd
zu einem Referendum über seine eigene poli

Es geht heute in der Türkei um mehr als nur K
türkische Seite von Google. Dort hat die Suc
Wahlurne mit Stimmzettel ersetzt.

Erstmals seit den Gezi-Protesten im Sommer
Korruptionsskandals hat Premierminister Erd
zu einem Referendum über seine eigene poli

Es geht heute in der Türkei um mehr als nur K
türkische Seite von Google. Dort hat die Suc
Wahlurne mit Stimmzettel ersetzt.

Erstmals seit den Gezi-Protesten im Sommer
Korruptionsskandals hat Premierminister Erd
zu einem Referendum über seine eigene poli

Knowtator: Arrow controls can be used to switch between different primary data documents. In addition, specific documents can be selected from a drop-down list ("document" icon) (S-KNW03).

Related patterns

- Internal relations: -
- External relations: This pattern uses the "Navigation spaces" pattern (Van Welie & Trætteberg 2000) and the "List browser" pattern (Van Welie & Trætteberg 2000)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.EasyNavigationBetweenMultiplePrimaryDataDocuments
Page last modified on September 15, 2014, at 09:07 PM EST

P3.4 - Automatic creation of annotation files

Problem description

Annotation tools typically add markup to a primary text document in stand-off annotation format, i.e. the annotations are stored in separate documents. Oftentimes, there are also files that manage the relation between primary data and annotation documents. The manual creation of these files that are essentially important for the technical realization and storage of the annotation is an abstract task for novice users.

Related problems: BRA09, DEX05, GLO02, KNW07, MAX08

Usability principle

Error prevention, match between system and the real world

Context

This pattern can be used when technical files for the realization of annotation storage need to be created.

Forces

- A novice user typically wants to import primary data and start annotating it.
- The annotation tool needs to specify annotation files first, in order to store any annotations made by the user in stand-off format.
- The file formats are oftentimes proprietary and therefore hard to grasp by novice users.
- The role of the files and their implications for the annotation process (especially the storage) are not clear to the novice user.

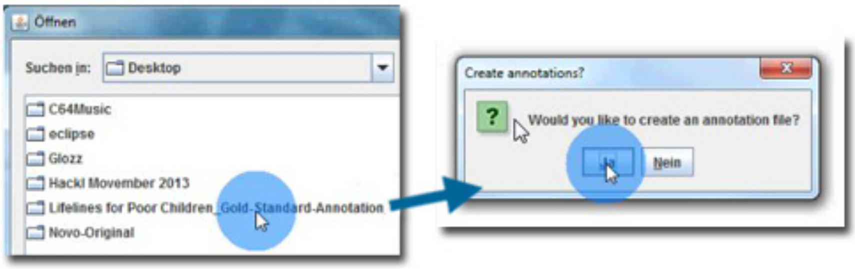
Solution

Create the files that are necessary for the technical representation and storage of the annotations automatically and do not bother the user with the task. Also provide a short hint on the storage location of the files at the beginning of the annotation process. More elaborate information on the technical details of the storage mechanism may be described in the manual, in order to allow advanced users to configure the files to their specific needs.

Rationale

By hiding technical details from the novice user, the annotation process is accelerated (*efficiency of use*). At the same time, errors that may result from faulty or incomplete user input for specifying the storage files can be prevented (*error rate*).

Example



Suchen in: Desktop

C64Music
eclipse
Glozz
Hackl Movember 2013
Lifelines for Poor Children_Gold-Standard-Annotation
Novo-Original

Would you like to create an annotation file?

WordFreak: Annotation files can be created automatically after the primary data has been imported (S-WOR04).

Known uses

UAM Corpus Tool

Related patterns

- Internal relations: This pattern refines P2.1
- External relations: -

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns /pmwiki.php?n=PmWiki.AutomaticCreationOfAnnotationFiles
Page last modified on September 15, 2014, at 09:11 PM EST

P3.5 - Mode for editing primary data

Problem description

During the annotation process, the user has to interact with the primary data in order to select anchors and apply annotations. Accidental manipulation (e.g. deletion) of the primary data during the annotation process may result in annotation inconsistencies that affect both, the annotator and the storage format of the annotation tool.

Related problems: GAT04, UAM08

Usability principle

Error prevention

Context

If an annotation tool allows the user to edit the primary data after it has been imported, this pattern can be used to prevent accidental manipulation of the text by making sure the user is aware that he is about to edit the primary data.

Forces

- Some annotation tools allow the user to manipulate the primary data during the annotation process.
- Typical actions to manipulate primary data may conflict with actions that are used for annotating the data.
- Mixing up commands for text manipulation and text annotation creates cognitive overhead for the tool user.

Solution

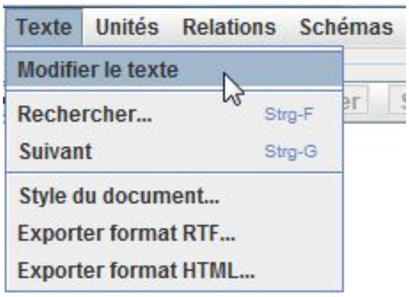
In most cases, the manipulation and annotation of primary data are not parallel tasks. In order to distinguish those tasks on the level of tool interactions, the tool provides two explicit modes for *annotation* and for *manipulation* of primary data documents. As the primary data typically is edited and normalized before it is imported to the tool, manipulation of the text data is a task that occurs less often than *annotation*. Therefore, the standard mode is the annotation mode, and a "data manipulation" mode is available on demand. It is recommended to implement a "shield" dialog that requires the user to explicitly confirm that he wants to switch to a mode where he can edit the primary data. This dialog also provides a hint on the consequences that changes in the primary data might have on existing annotations.

Rationale

By differentiating annotation and manipulation of primary data in two different modes, the potential to produce accidental text manipulations (*errors*) is reduced.

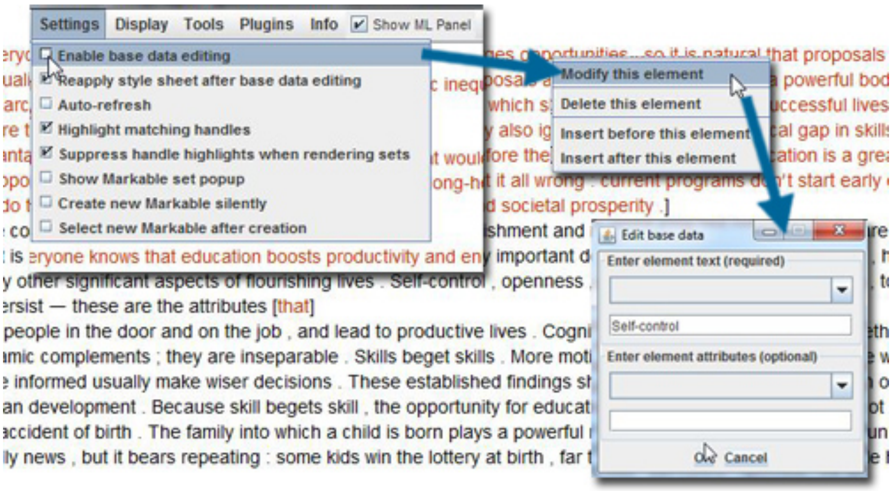
Example

Example 1



Analec: Mode for editing primary data has to be enabled (S-ANA02).

Example 2


Settings
Display
Tools
Plugins
Info
☑ Show ML Panel
Modify this element
Delete this element
Insert before this element
Insert after this element
Edit base data
Enter element text (required)
Enter element attributes (optional)
OK
Cancel

☐ Enable base data editing
☑ Reapply style sheet after base data editing
☐ Auto-refresh
☑ Highlight matching handles
☑ Suppress handle highlights when rendering sets
☐ Show Markable set popup
☐ Create new Markable silently
☐ Select new Markable after creation

Self-control

MMAX2: Mode for editing primary data has to be enabled. Primary data can be edited via a context menu (S-MAX01).

Related patterns

- Internal relations: -
- External relations: This pattern uses the "Shield" pattern (Van Welie & Trætteberg 2000)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns /pmwiki.php?n=PmWiki.ModeForEditingPrimaryData
Page last modified on June 05, 2014, at 04:23 PM EST

P3.6 - Support for reading primary data

Problem description

During the annotation process, the primary data is typically not read sequentially from beginning to end, but rather scanned for certain text fragments that can be used as an anchor for a specific annotation. The standard display of primary data often does not support such episodic scanning and reading.

Related problems: -

Usability principle

Flexibility and efficiency of use, aesthetic and minimalist design

Context

This pattern can be used to support the readibility of primary data for the purpose of linguistic annotation.

Forces

- The imported primary data document tends to be rather long, but has to be displayed in an adequate way in the annotation tool.
- Reading primary data in order to fulfill different annotation tasks is different from standard reading scenarios, where a text is read systematically, from beginning to end.

Solution

There are several features that can be implemented to enhance the readibility of primary data. This pattern describes four solutions that are not meant to be mutually exclusive:

Solution A: Numbered lines facilitate the navigation through the primary data document.

Solution B: The use of two different colors helps to distinguish alternating lines from each other (also known as *row striping*).

Solution C: The *page* metaphor allows the user to break down very long documents in smaller units that are familiar to the user.

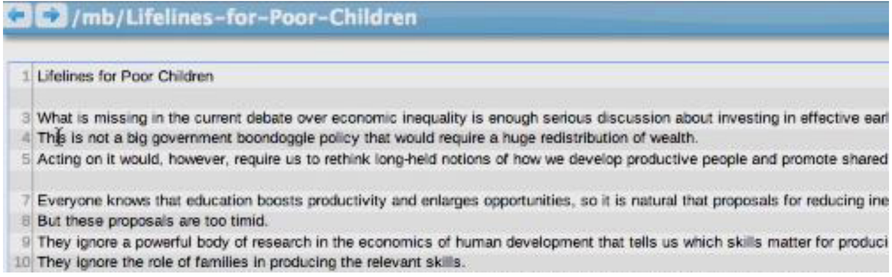
Solution D: Alternative views (e.g. table or tree view) enhance the readibility of the primary data.

Rationale

This pattern increases the readibility of the primary data and thus accelerates the annotation process (*efficiency of use*).

Example

Example for solutions A and B



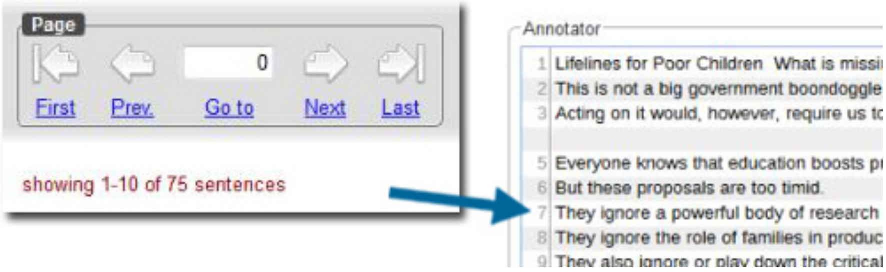
1 Lifelines for Poor Children

3 What is missing in the current debate over economic inequality is enough serious discussion about investing in effective earl
4 This is not a big government boondoggle policy that would require a huge redistribution of wealth.
5 Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared

7 Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing ine
8 But these proposals are too timid.
9 They ignore a powerful body of research in the economics of human development that tells us which skills matter for produci
10 They ignore the role of families in producing the relevant skills.

Brat: Primary data is displayed with numbered lines and row striping, for increased readibility (S-BRA02, S-BRA03, also cf. S-WEB01, S-WEB02).

Example for solution C



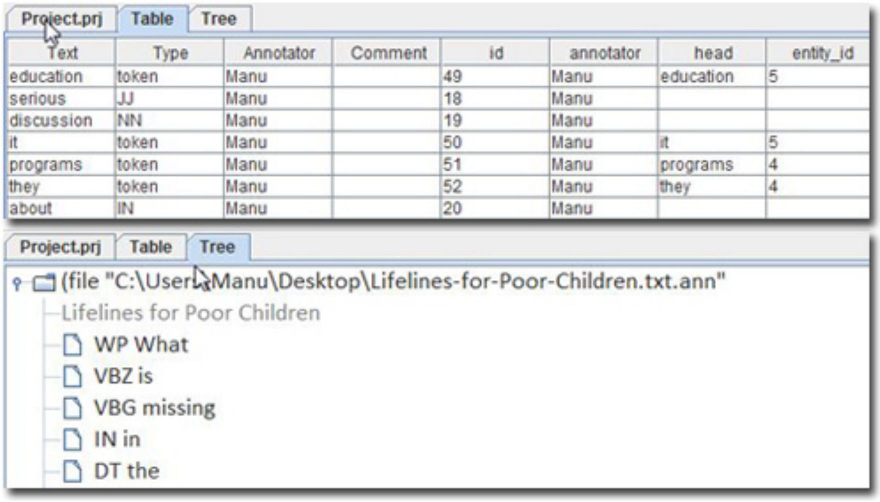
showing 1-10 of 75 sentences

1 Lifelines for Poor Children What is missi
2 This is not a big government boondoggle
3 Acting on it would, however, require us tc

5 Everyone knows that education boosts pı
6 But these proposals are too timid.
7 They ignore a powerful body of research
8 They ignore the role of families in produc
9 They also ignore or play down the critical

WebAnno: Longer primary data documents are displayed as single pages that can be browsed by means of arrow controls. The user can also configure the number of sentences to be displayed per page (S-WEB03).

Example for solution D



Text	Type	Annotator	Comment	id	annotator	head	entity_id
education	token	Manu		49	Manu	education	5
serious	JJ	Manu		18	Manu		
discussion	NN	Manu		19	Manu		
it	token	Manu		50	Manu	it	5
programs	token	Manu		51	Manu	programs	4
they	token	Manu		52	Manu	they	4
about	IN	Manu		20	Manu		

(file "C:\Users\Manu\Desktop\Lifelines-for-Poor-Children.txt.ann"
Lifelines for Poor Children
WP What
VBZ is
VBG missing
IN in
DT the

WordFreak: Besides the standard text view, there are several alternative views for displaying the primary data, e.g. a table view or a tree view (S-WOR03).

Related patterns

- Internal relations: -
- External relations: This pattern uses the "Row striping" pattern (Tidwell 2011) and the "Pagination" pattern (Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.SupportForReadingPrimaryData
Page last modified on September 15, 2014, at 09:13 PM EST

P3.7 - Tailored display of primary data

Problem description

Users have diverse preferences when it comes to the rendering of text data. At the same time, different annotation tasks may require different kinds of text display.

Related problems: GLO12

Usability principle

User control and freedom

Context

This pattern can be used to allow users to tailor the display of primary data according to their personal preferences, or to support the achievement of a specific annotation task.

Forces

- Annotation tools display text in a standarized way (default view).
- Annotators have individual preferences when it comes to the appearance of text (*font size*, *font family*, *line spacing*, etc.).
- Different annotation tasks require different display of primary data (e.g. larger font size if single words are annotated, smaller font size if sentences are annotated).

Solution

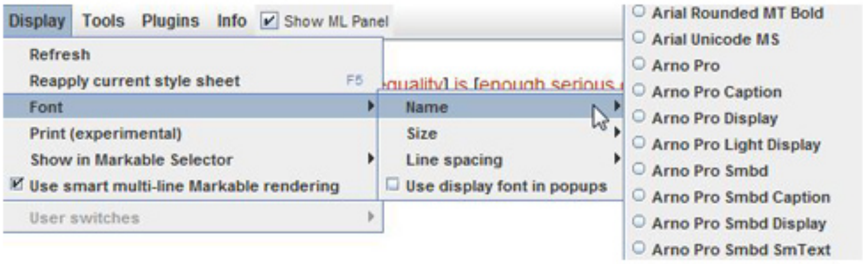
The tool provides an accessible menu with the following parameters that allow the users to tailor the display of primary data according to their needs:

- font-size
- font-family
- line spacing
- letter spacing
- margin spacing
- customized linebreaks after certain types of annotated anchors (e.g. linebreak after "title" annotation; cf. S-MAX03)

Rationale

A tailored display of primary data increases the overall annotation speed of the annotator (*efficiency of use*).

Example



MMAX2: Adjust font family, font size and line spacing (S-MAX02).

Known uses

Analec (S-ANA03), Brat (S-BRA04), WordFreak (S-WOR01)

Related patterns

- Internal relations: This pattern refines P3.6
- External relations: This pattern uses the "Preferences" pattern (Van Welie & Trætteberg 2000)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.TailoredDisplayOfPrimaryData
Page last modified on September 15, 2014, at 09:15 PM EST

6.5.2. List of remaining strengths

Metadata for primary data documents Some annotation projects may comprise a number of different text documents – in such cases, it may be helpful to be able to add metadata such as *author*, *source*, *description*, etc. to each of the documents (S-CAT02). Such metadata not only provides additional insights for later analysis, but also facilitates the navigation between different primary data documents during the annotation process.

Batch import of multiple primary data documents If a number of (short) single primary data documents is to be imported to the annotation tool, a batch import function that allows the user to import multiple documents at once can speed up the workflow significantly (S-KNW02).

Modification of encoding of primary data There is a number of different encoding formats (UTF-8, ISO 8859-1, etc.) for textual data. If the annotation tool fails to recognize the correct encoding format, the data may be displayed erroneously. Therefore, it is helpful for the user to be able to (optionally) adjust the encoding format manually (S-ANA01, S-WOR02).

6.5.3. List of remaining problem clusters

Cumbersome import of primary data The import of a "data file" into a tool is a recurring task that is known from many other applications (*MS Word*, *Adobe Photoshop*, *Audacity*, etc.). Typically, files are imported via a menu that is usually titled "file" or "project", and that is positioned in the top left corner of the tool. Some annotation tools, however, implement rather unorthodox mechanisms for importing documents: *GATE* requires the user to create a "Language Resource" (GAT05). This wording is not clearly understandable without previous knowledge about the tool. Furthermore, language resources comprise not only of primary data documents, but also of corpora and annotation schemes. *MMAX2* allows the user to import documents only in the initial "project wizard", but not at a later point in time (MAX04). *UAM Corpus Tool* uses terms like "Project" and "Import" in an unconventional fashion that is very different from other software contexts (UAM05).

6.5.4. List of tool-specific problems

Brat Even though the primary data is displayed with line breaks right after its import, lines are collapsed into one long line if an anchor scope spans multiple lines. The user has to scroll horizontally in order to display the primary data, which is very annoying (BRA22).

Glozz Once the dialog for importing the primary data is finished, there is no feedback that a document has been imported to the tool, i.e. the text is not displayed in the tool, and there is also no file list available that displays the newly imported document (GLO03). The user rather has to open the primary data manually via the file menu. This step seems to be unnecessary, as it irritates the user and slows down the import process. Another problem that was identified for *Glozz* seems to be a bug that is concerning the display of primary data: At some point during the evaluation of the tool, the document was "reloaded"; after this reload, the display of the primary data was completely broken, i.e. the text was displayed in different font sizes, with some paragraphs centered and also with arbitrary line breaks after single characters, words and phrases (GLO14).

Dexter *Dexter* has a separate conversion tool that pre-processes and converts primary data into a specific format. The conversion tool, however, seems to have problems parsing documents that contain double quotation marks, which leads to an error during the conversion process (DEX04).

Knowtator In *Knowtator*, primary data can only be imported by means of a small "folder" icon (KNW08). The icon does not really convey its purpose. The icon is positioned beyond the main menu structure, which runs the risk of not being seen by the user, who typically presumes the action to "import a primary data document" somewhere in the main navigation "File" menu.

MMAX2 While it is possible to create an annotation project and set the relevant parameters (e.g. name and color of annotation levels, project name, etc.) by means of a project wizard GUI, it is not possible to adjust these parameters at a later point in time with the same wizard (MAX06). After the initial creation of a project, the parameters may only be changed outside the tool, by editing the multiple project files in an XML editor.

WordFreak The process for displaying primary data in WordFreak involves a number of different steps, which are cumbersome to achieve (WOR04). (1) First, the primary data needs to be added and loaded. (2) Next, a viewer has to be selected from the "Viewer" menu. (3) The data is only displayed in the viewer if an annotation scheme has been selected via the "Annotation" menu. There should be default values for the viewer and the annotation scheme that allow the user to display the primary data in a standardized way after it has been imported.

WebAnno Although *WebAnno* is generally a tool with many good ideas for an improved, more user-friendly interface, it also suffers from some bugs[188] that influence the user experience in a negative way: (1) The "Upload tar.gz", a function in the "Data" dialog, does not work (WEB08). (2) The

[188] Note: *WebAnno* was used in the "StandAlone" version (cf. `https://code.google.com/p/webanno/wiki/WebAnnoStandalone`. The bugs documented in this section may not be an issue if *WebAnno* is set up with a tailored database and server.

"Open" button (as well as the "Cancel" button) in the "Annotations" window are not fully visible in the small dialog window (WEB09); the user has to resize the window to view the controls. (3) It took several seconds to open and display a test document that comprised about 75 sentences (WEB10). (4) If the user wants to re-open a previously annotated document, he has to use the generic "Open" dialog, as the default "Open document" dialog does not seem to work (WEB11).

6.6. Annotation scheme

The annotation scheme is an essential element of any annotation project, as it specifies all the annotation values that can be applied to anchors in the primary data. These values are oftentimes structured by means of different levels of annotation (cf. chapter 3.4.3). On the technical side, an annotation scheme is based on markup languages and document grammars (cf. chapter 3.3), which allow the user to define a list of elements and attributes that may be used for the annotation of a text document. XML seems to be the obvious choice for implementing linguistic annotation schemes; many tools, however, require the user to create the scheme beforehand, which implies knowledge about markup languages, and the XML syntax respectively. Only few tools allow the user to create annotation schemes inside the actual annotation tool (P4.1). From the interaction perspective, the user needs to be able to access the annotation scheme during the annotation process in order to select concrete values that can be applied to a previously selected anchor. Most tools display the annotation scheme throughout the whole annotation process, a few tools only display the scheme after an anchor has been selected (cf. ANA09, BRA13, GAT13, WEB14). When it comes to how the individual annotation values are organized in the scheme (P4.2.), two main categories can be observed:

1. All values are displayed at once. This is typically achieved by means of lists or file trees (cf. *Dexter*, *Glozz*, *MMAX2*, *GATE*, *Brat*, *CATMA*, *UAM Corpus Tool*) or separate windows (cf. *WordFreak*).
2. Only some values are displayed at once, i.e. usually the user has to open a drop-down menu to select the appropriate annotation value (cf. *ANA08*, *WEB16*, *GATE*).

In order to avoid annotation schemes that are crammed with too many values, the user needs to be able to customize the annotation scheme and the display of its values according to his specific needs (P4.3). Another critical factor that may have great influence on the overall annotation process, is the easy distinction of multiple annotation values in the annotation scheme (P4.4).

6.6.1. Usability patterns

Four usability patterns were identified for this category:

P4.1 Integrated annotation scheme editor

P4.2 Organization of annotation scheme

P4.3 Tailored display of annotation schemes

P4.4 Facilitated distinction of multiple annotation items

Overview of the usability problems and tool-specific strengths that were used as input for the patterns:

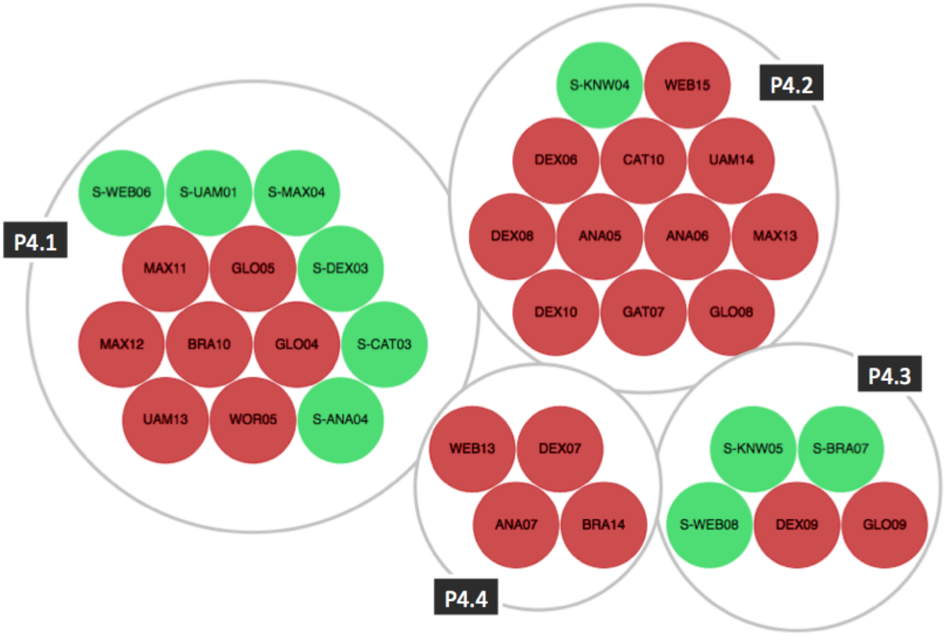


Figure 6.6.: Usability problems and strengths used as input for patterns in category "Annotation scheme".

P4.1 - Integrated annotation scheme editor

Problem description

The creation of an annotation scheme that defines different levels of annotation as well as concrete annotation items on each level is a crucial task in any annotation project. Typically, annotation schemes are defined by means of document grammars known from markup languages like XML or SGML. Users without technical knowledge about markup languages will have difficulties in creating a scheme in XML syntax. At the same time, many tools require to define an annotation scheme outside the annotation tool, which makes the task even more challenging for markup novices.

Related problems: BRA10, GLO04, GLO05, MAX11, MAX12, UAM13, WOR05

Usability principle

Error prevention, flexibility and efficiency of use

Context

This pattern can be used to facilitate the creation of annotation schemes for users without technical knowledge about markup languages and document grammars.

Forces

- Annotation schemes are typically defined by means of document grammars (DTDs or XMLSchemas).
- Annotation schemes are typically defined outside of the annotation tool, in a text editor that facilitates the creation of document grammars.
- For editing existing schemes during the annotation process, it is impractical to switch between the annotation tool and an external text editor.
- It is vital to provide a synchronization mechanism that ensures that changes made to the annotation scheme are updated and applied to the current, ongoing annotation process (cf. S-MAX04).

Solution

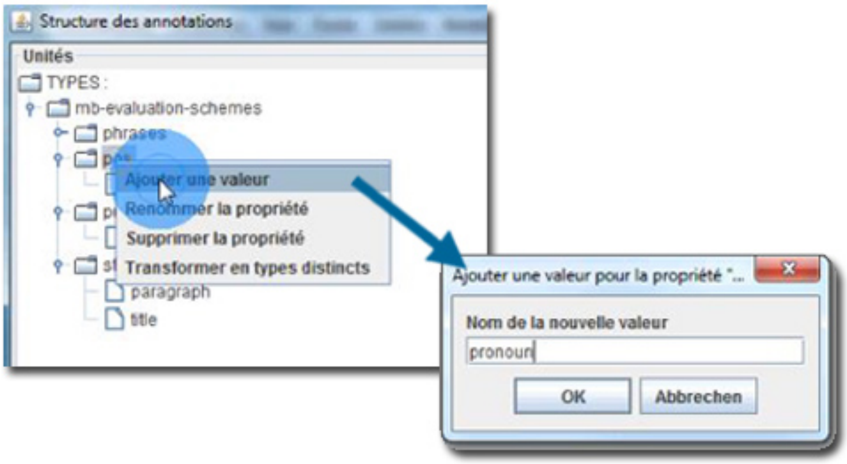
The tool integrates a scheme editor that allows the user to define and edit annotation schemes inside the annotation tool. By providing a graphical user interface for the scheme editor it is also possible to hide technical details of the storage format from the user. Such an interface should utilize well-known metaphors for the creation of hierarchical structures, such as file-trees (cf. S-ANA04) or ordered lists. It can also make use of established input elements, such as forms and input fields (cf. S-WEB06). It must be made clear via the interface which annotation items belong to which level of annotation, i.e. typically the annotation levels are at the highest hierarchical level of the scheme, while concrete annotation items can be subordinate to those different levels.

Rationale

As ad hoc modifications of the annotation scheme are part of the typical annotation process, an integrated editor for annotation schemes speeds up the overall annotation process (*efficiency of use*). At the same time, the availability of a GUI for the creation and modification of annotation schemes increases the *learnability* of the annotation tool and decreases the number of potential *errors* that may occur when novices are forced to translate linguistic annotation schemes into formal markup languages.

Example

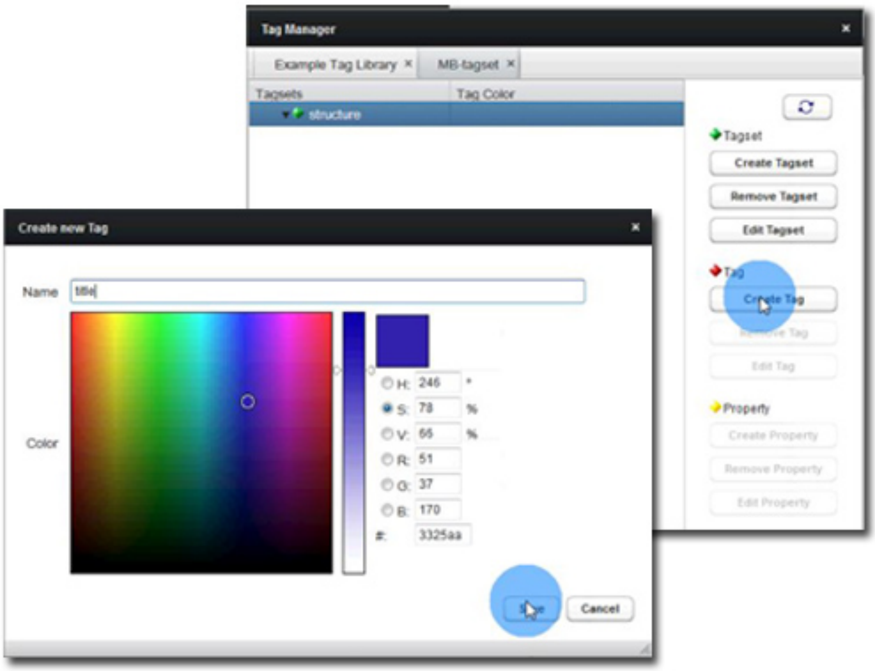
Example 1



Nom de la nouvelle valeur: pronoun

Analec: Integrated annotation scheme editor (S-ANA04).

Example 2

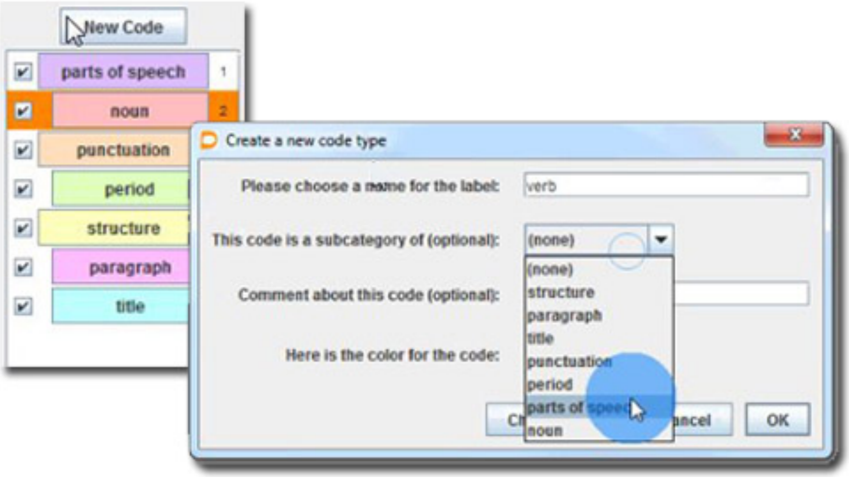


Name: title

H: 246 °
S: 78 %
V: 66 %
R: 51
G: 37
B: 170
#: 3325aa

CATMA: Integrated annotation scheme editor (S-CAT03).

Example 3



Please choose a name for the label: verb

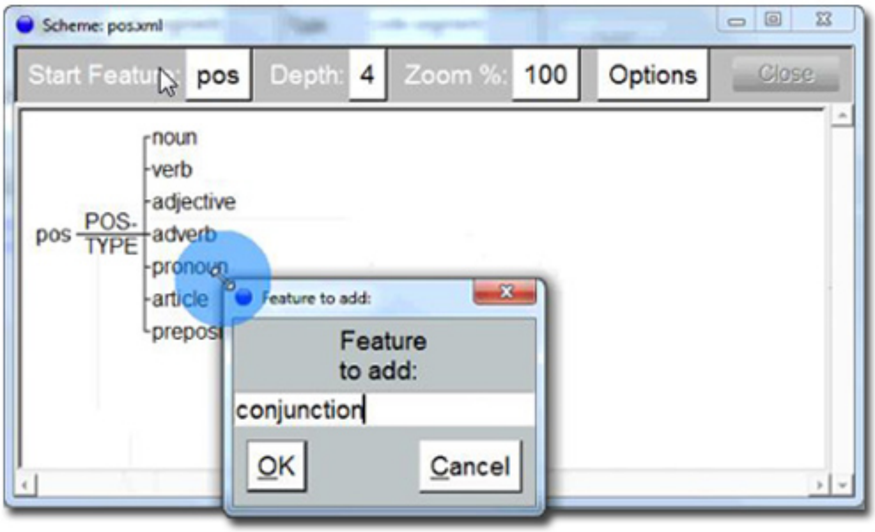
This code is a subcategory of (optional): (none)

Comment about this code (optional):

Here is the color for the code:

Dexter: Integrated annotation scheme editor (S-DEX03).

Example 4



Scheme: pos.xml

Start Feature: pos

Depth: 4

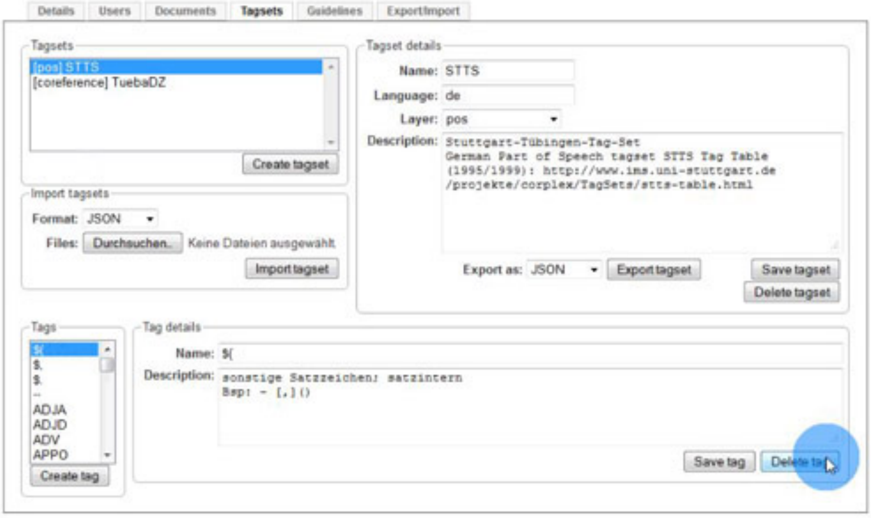
Zoom %: 100

Feature to add:

Feature to add: conjunction

UAM Corpus Tool: Integrated annotation scheme editor (S-UAM01).

Example 5



Tagsets
[pos] STTS
[coreference] TuebaDZ

Tagset details

Name: STTS

Language: de

Layer: pos

Description:
```
Stuttgart-Tübingen-Tag-Set
German Part of Speech tagset STTS Tag Table
(1995/1999): http://www.ims.uni-stuttgart.de
/projekte/corplex/TagSets/stts-table.html
```

Import tagsets

Format: JSON

Files: Durchsuchen... Keine Dateien ausgewählt

Export as: JSON

Tags
$(
$,
$.
--
ADJA
ADJD
ADV
APPO

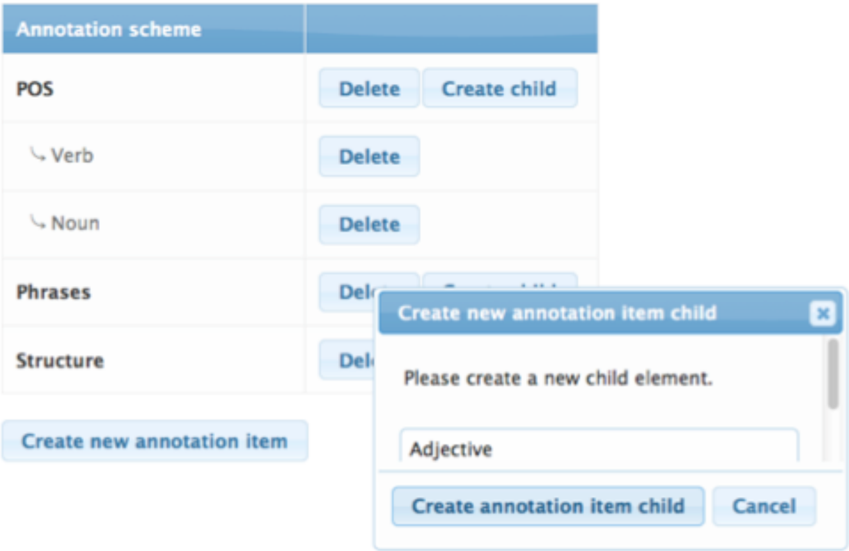
Tag details

Name: $(

Description:
```
sonstige Satzzeichen; satzintern
Bsp: - [,]()
```

WebAnno: Integrated annotation scheme editor (S-WEB06).

Interactive prototype


Annotation scheme
POS
Delete
Create child
Verb
Delete
Noun
Delete
Phrases
Del
Structure
Del
Create new annotation item
Create new annotation item child
Create annotation item child
Cancel

Please create a new child element.

Adjective

Prototype P4.1 - The prototype allows the user to create and delete annotation items on two different hierarchical levels.

Known uses

GATE, Knowtator

Related patterns

- Internal relations: This pattern uses P4.2
- External relations: This pattern uses the "Structured format" pattern (Tidwell 2011) and the "Input hints" pattern (Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns /pmwiki.php?n=PmWiki.IntegratedAnnotationSchemeEditor
Page last modified on September 12, 2014, at 06:46 PM EST

P4.2 - Organization of annotation scheme

Problem description

Annotation schemes typically consist of multiple levels of annotation and a number of annotation items on each of those levels. The user needs an effective mechanism that allows him to organize different elements of an annotation scheme.

Related problems: ANA05, ANA06, CAT10, DEX06, DEX08, DEX10, GAT07, GLO08, MAX13, UAM14, WEB15

Usability principle

Error prevention, aesthetic and minimalist design

Context

This pattern can be used to facilitate the organization and visualization of an annotation scheme inside the annotation tool. The pattern basically sums up important user-interactions with the elements of an annotation scheme.

Forces

- Typically, annotation schemes consist of multiple levels of annotation (e.g. "parts of speech", etc.).
- Each level of annotation comprises several annotation items (e.g. "noun", "verb", etc.).
- The user wants to organize (order and hierarchy) the different elements of the scheme in an effective way.
- The user wants a clear overview of the whole annotation scheme that shows how different levels and their subelements relate to each other.

Solution

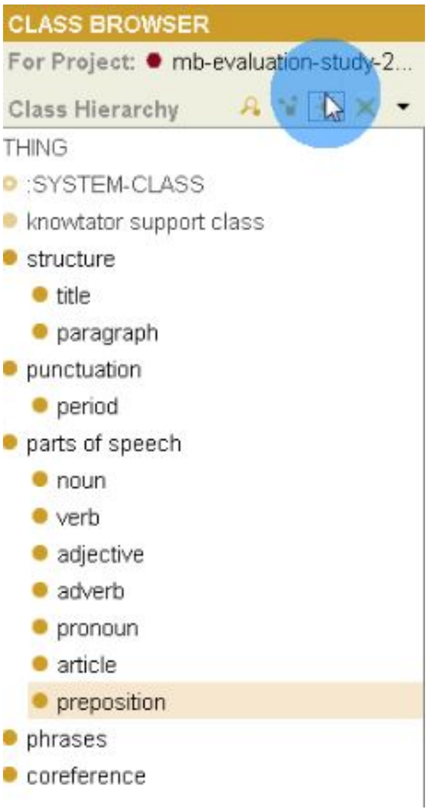
The tool provides a mechanism that allows the user to create an unambiguous hierarchy inside the annotation scheme, where the different levels of annotation are positioned on the highest level. Users are able to move elements from one level to another in an easy and intuitive way. Also, the hierarchical structure of the scheme must be visualized adequately. This can be achieved by using known metaphors such as nested trees, indented lists and tabs. Another important feature to support the organization of the annotation scheme is the possibility to change the order of levels and items.

Rationale

By providing a way to organize and visualize the annotation scheme in a hierarchical fashion, the user can quickly navigate through the annotation scheme and speed up the entire annotation process. This increases the tool's *efficiency of use* as well as the *memorability* of the position of specific items in the annotation scheme. Allowing the user to change the order of levels and items adds to these two aspects of usability.

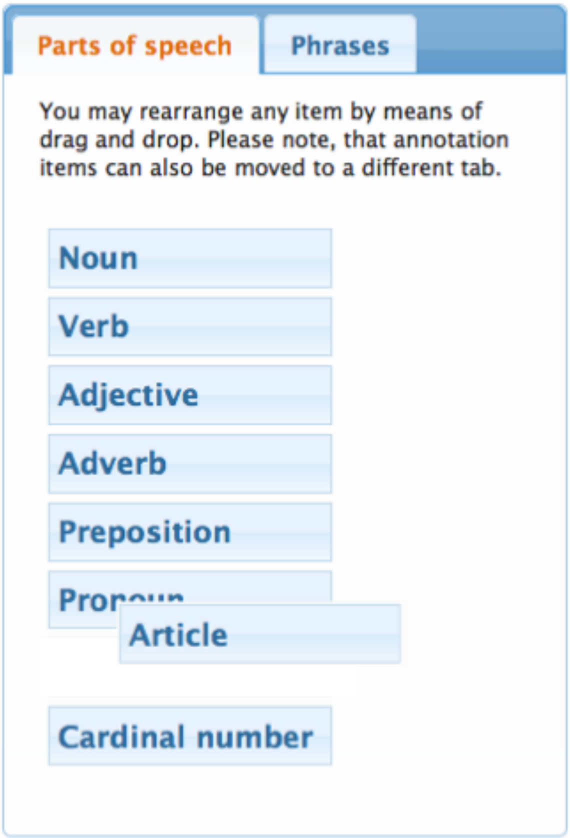
Example

Example



Knowtator: The annotation scheme is displayed as a hierarchical tree in which nodes can be modified by means of drag-and-drop (S-KNW04).

Interactive prototype



You may rearrange any item by means of drag and drop. Please note, that annotation items can also be moved to a different tab.

Prototype P4.2 - The prototype allows the user to rearrange a set of predefined annotation items by means of drag-and- drop. Items can also be moved from one annotation level to the other.

Related patterns

- Internal relations: -
- External relations: -

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.OrganizationOfAnnotationScheme
Page last modified on September 12, 2014, at 06:59 PM EST

P4.3 - Tailored display of annotation schemes

Problem description

Annotation schemes that contain many different annotation levels and annotation items can quickly become crammed, and will obfuscate the user during the annotation process.

Related problems: DEX09, GLO09

Usability principle

Aesthetic and minimalist design

Context

This pattern can be used to allow the user to tailor the display of the levels and items defined in the annotation scheme.

Forces

- During an annotation project, many different levels of annotation with multiple annotation items on each level are used.
- The display of too many annotation items can result in a crammed interface that makes it hard to find the appropriate item during the annotation process.
- Typically, the annotator does not use all annotation items from all annotation levels at the same time during the annotation process, but rather annotates from level to level.

Solution

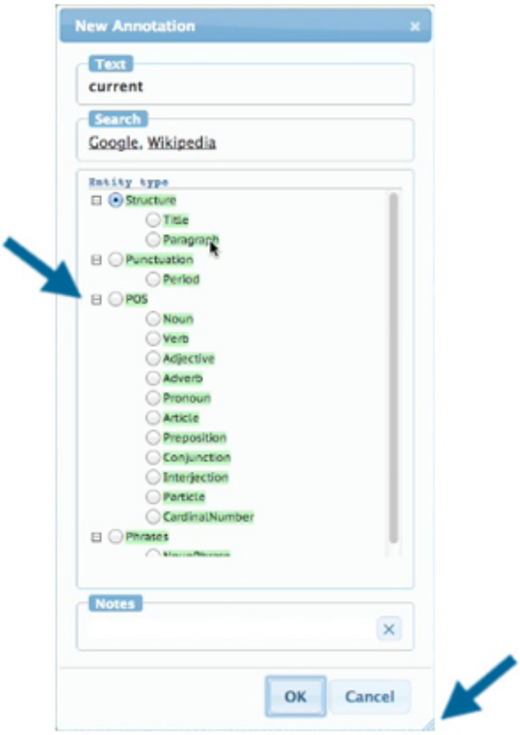
The tool provides an easy-to-use mechanism that allows the user to tailor the display of annotation levels and items by showing / hiding them from the annotation scheme that is displayed during the annotation process. It is also possible to hide higher level units and all their subordinate elements (e.g. hide the whole annotation level "parts of speech").

Rationale

By providing a tailored view on items that are displayed in the annotation scheme, the cognitive overhead is reduced significantly. The user may hide items that are not needed during the current annotation task and thus reduce the number of options he needs to select from. This helps to prevent *errors* and increases the tool's *efficiency of use*.

Example

Example 1



New Annotation ×

Text: current

Search: Google, Wikipedia

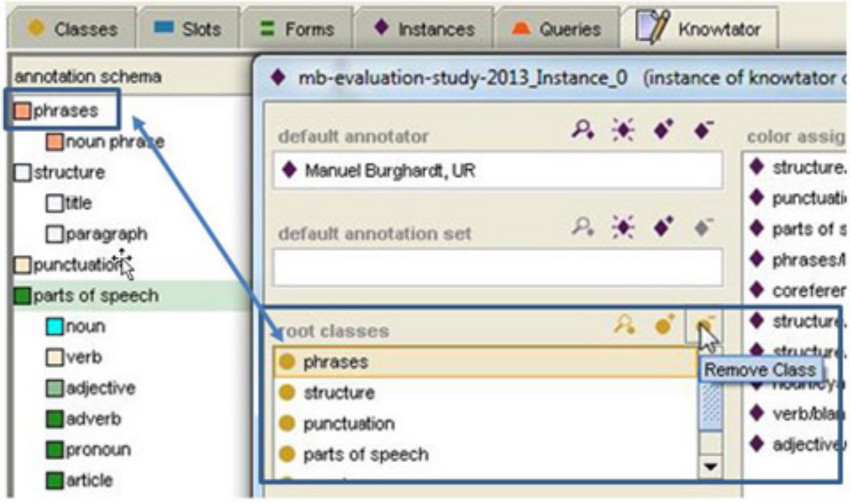
Entity type

- ⊟ ● Structure
  - ○ Title
  - ○ Paragraph
- ⊟ ○ Punctuation
  - ○ Period
- ⊟ ○ POS
  - ○ Noun
  - ○ Verb
  - ○ Adjective
  - ○ Adverb
  - ○ Pronoun
  - ○ Article
  - ○ Preposition
  - ○ Conjunction
  - ○ Interjection
  - ○ Particle
  - ○ CardinalNumber
- ⊟ ○ Phrases

Notes

Brat: Annotation levels can be extended or collapsed. The annotation scheme window can be resized, to show more or less annotation items (S-BRA07).

Example 2

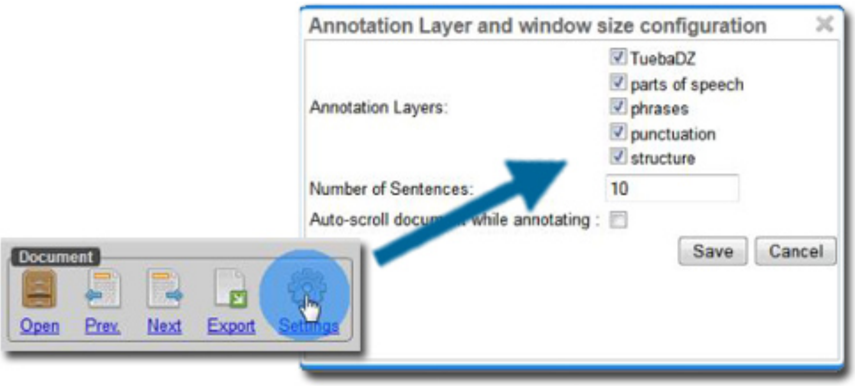


default annotator: Manuel Burghardt, UR

root classes: phrases, structure, punctuation, parts of speech

Knowtator: Annotation levels or single annotation items can be displayed or hidden during the annotation process (S-KNW05).

Example 3



Annotation Layer and window size configuration

Annotation Layers: ☑ TuebaDZ ☑ parts of speech ☑ phrases ☑ punctuation ☑ structure

Number of Sentences: 10

Auto-scroll document while annotating : □

Save Cancel

WebAnno: Annotation levels can be displayed or hidden during the annotation process (S-WEB08).

Related patterns

- Internal relations: This pattern refines P4.1
- External relations: This pattern uses the "Preferences" pattern (Van Welie & Trætteberg 2000) and refines the "Favorites" pattern (Van Welie & Trætteberg 2000)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.TailoredDisplayOfAnnotationSchemes
Page last modified on June 20, 2014, at 07:09 PM EST

P4.4 - Facilitated distinction of multiple annotation items

Problem description

During the annotation process, the user has to choose from a multidude of different annotation items. The selection of the right item at the right moment can be cumbersome, as multiple items may obfuscate the annotator.

Related problems: ANA07, BRA14, DEX07, WEB13

Usability principle

Recognition rather than recall, aestetic and minimalist design

Context

This pattern can be used to distinguish multiple items of an annotation scheme during the annotation process.

Forces

- An annotation scheme tyically contains several levels of annotations, each with a number of different annotation items.
- During the annotation process, the user has to select a specific item from the annotation scheme in justifiable time; this selection becomes more challenging with every additional item in the scheme.

Solution

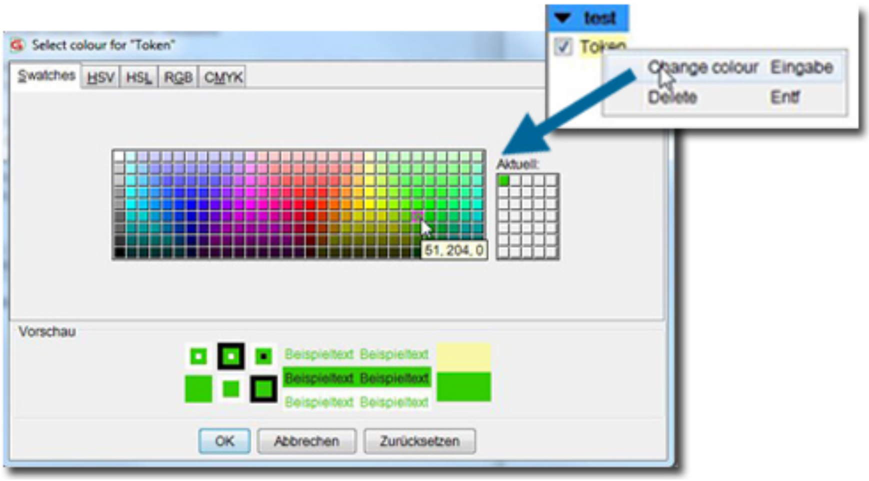
Users have a hard time distinguishing different annotation items by their textual pattern. The tool uses different colors as an aditional means to differentiate the annotation items in the scheme. The tool provides default colors for each item, but also allows the user to modify the color for specific items according to his personal preferences. By providing default colors, the user is saved the trouble of finding distinctive colors for all items. The user may, however, have a mental model that links certain colors to certain entities, in which case it is desirable to set some color values individually via a *color picker*. Also, the set of colors should not be limited to a predefined palette, but rather span the whole spectrum of possible colors. The colors that are used to differentiate items in the annotation scheme are also used to visualize annotations in the primary data (underlines or highlights).

Rationale

By providing different colors to distinguish multiple annotation items in the scheme, the user can increase the overall *efficiency* of the annotation process, as appropriate annotation items can be selected more quickly.

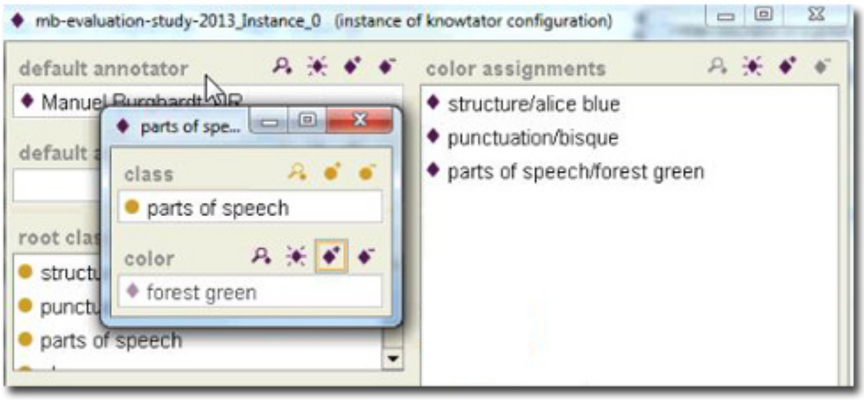
Example

Example 1



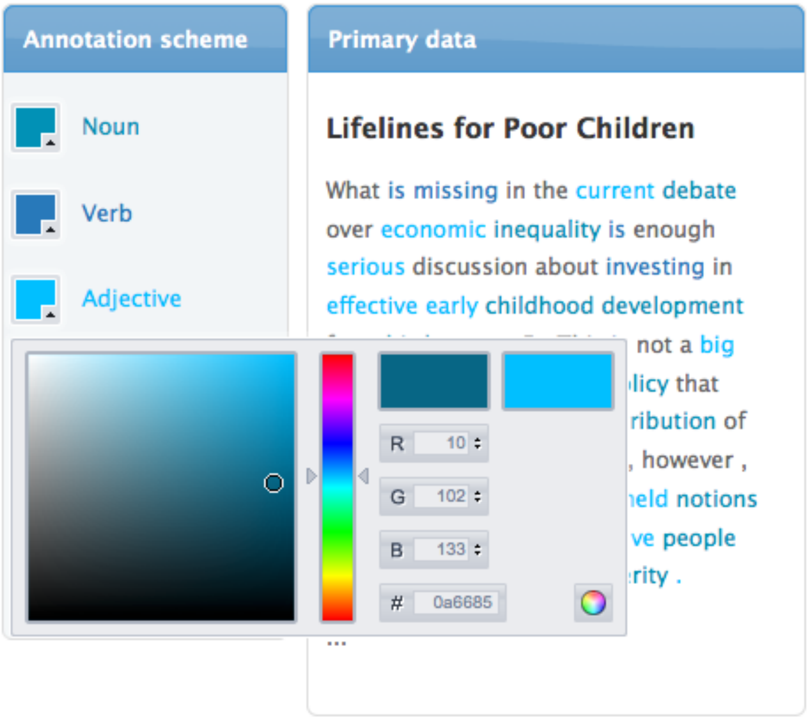
GATE: GATE provides default colors for every annotation value, but also allows the user to set colors individually.

Example 2



Knowtator: Knowtator provides a list of color names from which the user may select. It is also possible to assign a color to a whole annotation level (e.g. parts of speech). There are, however, no predefined colors for the annotation items (default: no color).

Interactive prototype



Lifelines for Poor Children

What is missing in the current debate
over economic inequality is enough
serious discussion about investing in
effective early childhood development
not a big
licy that
ribution of
, however ,
held notions
ve people
rity .
...

R 10
G 102
B 133
0a6685

Prototype P4.4 - The prototype illustrates the basic interaction behavior for setting individual colors for different annotation items by means of a color picker.

Known uses (with minor limitations)

CATMA (no predefined colors), Dexter (only limited color palette), Glozz (no predefined colors)

Related patterns

- Internal relations: This pattern refines P4.1
- External relations: This pattern uses the "Dropdown chooser" pattern (Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.FacilitatedDistinctionOfMultipleAnnotationItems
Page last modified on September 12, 2014, at 07:15 PM EST

6.6.2. List of remaining strengths

Easy import and export of annotation schemes *WebAnno* (S-WEB04) allows the user to export tag sets that were created inside the tool in structured JSON format. These tag sets – or other tag sets defined according to the *WebAnno* JSON structure – can also be imported to the tool, and are then displayed in the internal annotation scheme view. This feature is very helpful, as it allows the user to share annotation schemes with others.

Availability of example tag sets *WebAnno* (S-WEB07) provides an exemplary tag set for each of the annotation layer types that are available in the tool (cf. WEB12). These default tag sets can be deleted or modified very easily, and at the same time help to understand what types of annotation scheme can be implemented with *WebAnno*.

Intuitive syntax for the creation of annotation schemes If there is no integrated scheme editor (cf. P4.1) available, most tools require the user to create an annotation scheme outside the tool, by means of XML markup. As the usage of XML may be cumbersome for users without technical knowledge about markup languages, a more intuitive syntax, that does not require angle brackets, can facilitate annotation scheme creation (S-BRA06).

6.6.3. List of remaining problem clusters

Visibility and accessibility of annotation scheme While most tools display the annotation scheme throughout the whole annotation process, other tools require a specific action (e.g. the selection of an anchor scope) to display the scheme (ANA09, BRA13, GAT13, WEB14). Seeing the annotation values in an annotation scheme at all times facilitates the anchor selection, as the user will know in advance which value he is going to apply. In the referenced cases, anchor selection and annotation value selection are not integrated, but rather two independent steps.

Scrolling in the annotation schemes Some tools display the values of the annotation scheme as a drop-down list that only shows a limited number of items at once (ANA08, WEB16). To facilitate quick selection of values, they should be all available at once. If the scheme contains a great number of annotation values, the user should be able to customize the view of the items in the scheme.

Creation of an empty annotation level required Some tools (cf. UAM10, MAX09) implement the creation of an empty annotation level and the creation of concrete annotation values on that level as two distinct steps. This is very unintuitive, as typically the user wants to create values on a newly defined annotation level right away.

Wording and metaphors There are many examples for tools that use unconventional wording and metaphors for functions and actions necessary to create an annotation scheme. As was described in chapter 2.4, the predominant term for the linguistic markup of text documents seems to be *annotation*, i.e. *annotation* and *annotation scheme* should be used preferably over terms such as *markup / markup scheme*, *coding / coding scheme*, etc. There are also some tool-specific wording particularities that are not understood intuitively (CAT11). Another branch of problems can be found in the use of specific models for the implementation of an annotation scheme (ANA04, GLO07, UAM12)

Default tags Some tools (GAT08, GAT09, UAM11) create default tags whenever a new scheme is created. It is, however, not clear where these tags are coming from, or if they are important for the creation of an individual scheme. These tags also do not facilitate the comprehension of the annotation scheme building process, but are rather irritating.

6.6.4. List of tool-specific problems

Brat Although *Brat* does not use XML syntax for the creation of its annotation schemes, it does not allow for whitespace characters in the annotation values (BRA11). In *Brat*, there are four basic annotation types: *entities*, *relations*, *events* and *attributes*. If any of those types is not used directly in the scheme, the user nevertheless has to define a placeholder for those unused types (BRA12). Although there is a hint in the configuration tutorial, this is very unintuitive, and may easily lead to errors in the scheme definition.

CATMA Relating newly created tag sets to a primary data document by means of drag-and-drop is not an intuitive interaction in the context of a web application (CAT12).

GATE At the beginning of the annotation process, there is an empty, unnamed annotation level (GAT10). It is not clear where this annotation level is coming from, and what its purpose is. The creation of new annotation levels is achieved by clicking on a small "New" button, in the bottom right corner (GAT11). The button can be easily overlooked. The renaming and deletion of an existing layer can be achieved via a right click and a context menu; it would be more consistent to move the creation of new annotation levels into this context menu as well. Once an annotation value has been created, it cannot be renamed, but rather has to be deleted (all previous annotations with this value will be deleted as well) and created anew (GAT12). It is not clear how annotation values that are created ad hoc, during the annotation process, can be assigned to an existing annotation level (GAT15). Finally, the annotation window, that pops up when hovering over an anchor, contains some irritating parameters, whose function is not clear to the user (GAT17).

Glozz Some annotation values that have been specified in the scheme cannot be used during the annotation process (GLO06). It is not clear why these values cannot be used.

MMAX2 The "Validate" prompt after the initial creation of a project is irritating, as at the beginning of the annotation process there are no annotations that could be validated (MAX10).

UAM Corpus Tool There are too many unexplained options and parameters in the project wizard, which might irritate novice users (UAM 09). This is even more problematic, as the project wizard has to be used to create an annotation project at the very beginning of the annotation process, which means the user has no experience with the tool that could be used to interpret the options. In the annotation scheme window, there is only limited space for the display of annotation items (UAM17).

WebAnno In *WebAnno*, annotation levels cannot be created arbitrarily, but rather have to conform to one of the six predefined, basic layer types (WEB12); the predefined layer types do not cover all annotation scenarios. Another restriction is that it is not possible to define multiple annotation layers of the same annotation layer type.

WordFreak In *WordFreak* it is only possible to display one specific annotation level at a time (WOR06). This is inappropriate if the user wants to annotate one anchor with parallel annotations from different annotation levels. While *WordFreak* provides the user with a number of predefined annotation levels, they are poorly documented, i.e. it is not clear how the specific annotation values are to be used correctly (WOR07). There is also no comprehensive list that explains the various abbreviations for the tags.

6.7. Annotation process

The main goal of the annotation process is to apply an annotation value from the annotation scheme to a target anchor segment in the primary data. This overall goal implies a number of different subtasks and interaction steps with the annotation tool, which may be hard to differentiate by the annotator. Also, the subtasks may be conflicting on the interaction level, i.e. similar actions lead to different results. By splitting up the annotation process into single subtasks, it becomes more transparent and allows us to find solutions for an efficient and user-friendly interface. The following subtasks can be identified for the annotation process:

Create anchor: First, a span of text has to be selected from the primary data, which is then used as the anchor for an annotation value. As the manual selection of anchors can be a laborious task, the tool ideally provides a mechanism for the semi-automatic creation of anchors (P5.1). In any case, established selection-related gestures for the creation of anchors should be supported (P5.2). The tool should also give some kind of (visual) feedback for the selected anchor scope (P5.3). Also, the selection process should be integrated in an efficient way with the next step, the selection of annotation values (cf. MAX16, GLO11, ANA12). Some tools require the user to confirm each anchor creation explicitly (cf. MMAX2, Analec) or they even demand to switch to a specific "anchor creation" mode (cf. Glozz), which is both very annoying, and therefore should be avoided.

Select annotation: Annotation values are typically selected by single clicking on the respective item from the annotation scheme (cf. UAM16 for negative examples). One tool requires the user to click on a small colored button, next to the annotation value (cf. CAT13).

Apply annotation: After an annotation value has been selected, it has to be applied to an anchor. Most tools require the user to explicitly assign a value to an anchor by confirming the action with a single click on a "confirm button", or by double clicking the annotation value (cf. UAM16). Either way may be cumbersome if a great number of anchors has to be annotated. That is why some tools provide a "fast mode" which allows the user to assign values without explicit confirmation; other tools provide shortcuts that allow the user to select and assign a value by pressing a predefined key (P5.4). Usually, selecting the anchor happens before selecting the annotation value (cf. Glozz for a different approach). There is, however, potential to speed up the process by providing a mode to apply a specific annotation value to multiple anchors (P5.5).

Display annotations: The display of annotations is closely connected to the annotation process, as it provides visual feedback about the annotation action that has been performed by the user. Displaying annotations is of course also crucial for allowing the user to edit or select them. There are some more issues concerning the display of annotation that will be described in some more detail in section 6.8.

Edit anchor scope: After the initial creation of an anchor, it may be necessary to adjust its scope. While some tools require the user to delete the existing anchor and create it anew with the adjusted scope, other tools provide assistance for the modification of existing anchors (P5.7).

Delete anchor: Only two annotation tools (*Analec* and *Knowtator*) allow for the explicit deletion of an existing anchor (and all its annotations). For all other tools, the anchor is automatically deleted when the last annotation that is associated with this anchor is deleted.

Edit annotation: Typically, annotation values can be edited by simply overwriting an existing value with a new value from the scheme.

Delete annotation: In order to delete existing annotations it is necessary to display the annotations. Usually, the displayed value can then be selected and deleted by means of a "Delete" function (P5.8, P5.9).

The annotation of relations between two or more anchors is a special case of annotation that requires additional interaction behavior. While the existence of the anchors that are about to be connected by means of a relation is mandatory for any tool, there is a variety of mechanisms for the creation of the relation. The annotation of relations is different from the previously described annotation process, as the annotation is not applied to only one anchor, but rather to two or more anchors. The function of such relational annotations is to establish a (directed) connection between two or more anchors, e.g. a co-reference relation (P5.10).

6.7.1. Usability patterns

Ten usability pattern were identified for this category:

P5.1 Semi-automatic creation of anchors

P5.2 Conventionalized actions for creation of anchors[189]

P5.3 Feedback for creation of anchor scope

P5.4 Accelerators for selecting and applying annotations

P5.5 Accelerators for annotating multiple anchors

P5.6 Access to existing annotations

P5.7 Assistance for modification of anchors

P5.8 Integrated delete annotation function

P5.9 Mass deletion of annotations

P5.10 Support annotation of relations

Overview of the usability problems and tool-specific strengths that were used as input for the patterns:

[189] Note: The conventionalized selection gestures (Tidwell, 2005, p. 256) are quoted from Tidwell's 1st edition of "Designing Interfaces", as they are no longer part of the second edition of the book.

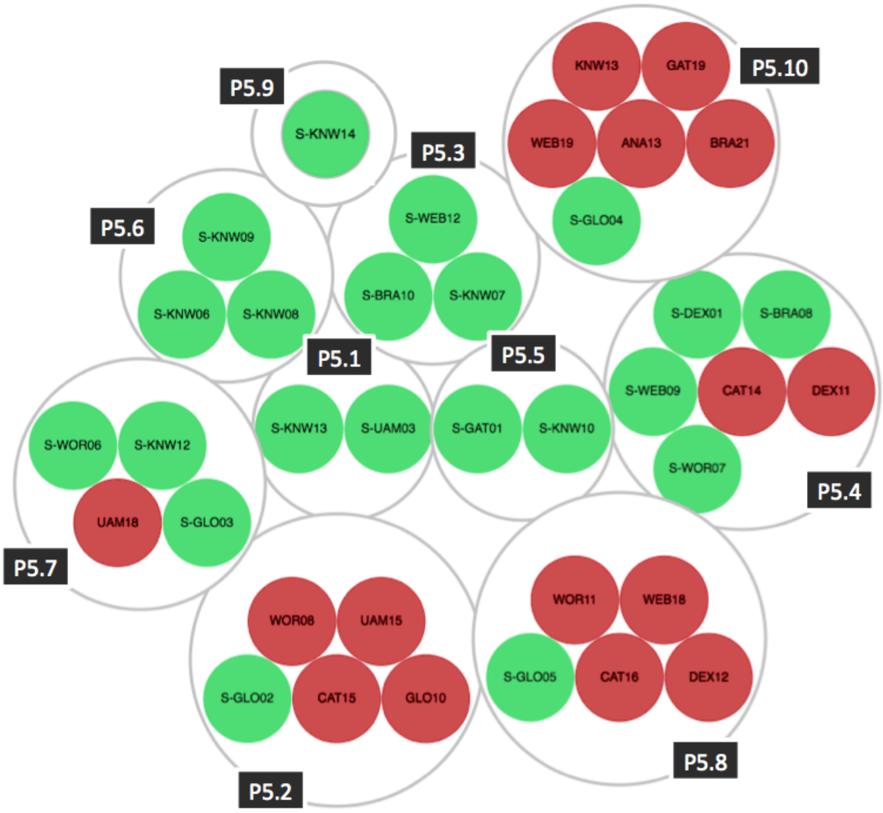


Figure 6.7.: Usability problems and strengths used as input for patterns in category "Annotation process".

P5.1 - Semi-automatic creation of anchors

Problem description

The manual definition of a large number of anchors is a time-consuming and tedious task for the user.

Related problems: -

Usability principle

Error prevention, flexibility and efficiency of use

Context

This pattern can be used to provide a component for the automatic creation of specific anchor types.

Forces

- Manual selection of anchors provides a high degree of flexibility.
- Manual selection of anchors can be a repetitive and thus tedious task.
- Automatic selection of anchors requires basic rules for the definition of an anchor scope.

Solution

The tool provides a semi-automatic segmentation function that allows the user to create anchors in the primary data according to a set of predefined rules:

- On the most basic level, the tool can make use of whitespaces and linebreaks to define anchors for word and paragraph scopes. If multiple options and rules are available, a step-by-step dialog facilitates the configuration of the automatic anchor creation mechanism.
- A more advanced approach allows the user to specify patterns for more complex anchor types by means of regular expressions or a similar formalism (cf. S-KNW13).

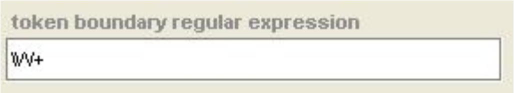
It is recommended to implement a "shield" dialog that requires the user to explicitly confirm that he wants to create anchors according to the selected parameters automatically. This dialog also provides a hint on the consequences that the automatic creation of anchors might have on the overall annotation process.

Rationale

This pattern increases the *efficiency* of the annotation process by providing a semi-automatic way to specify certain types of anchors in the primary data.

Example

Example 1

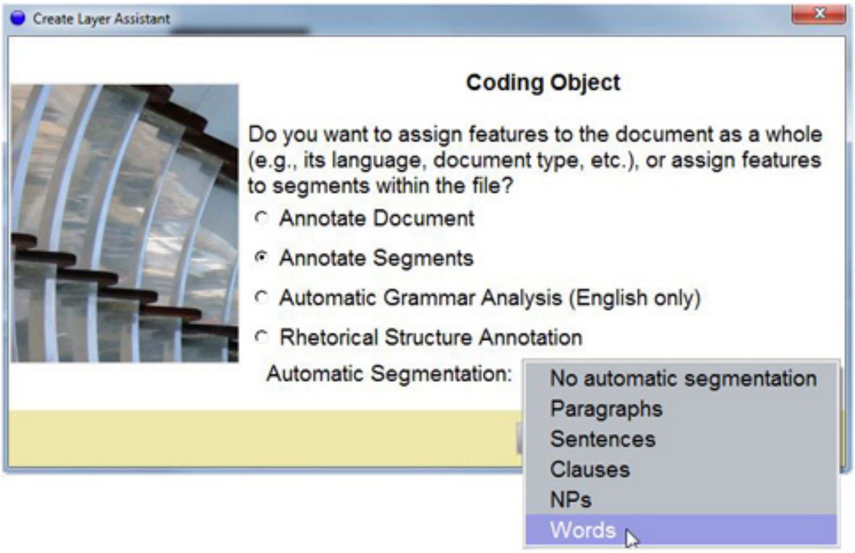


token boundary regular expression

\W+

Knowtator: The double click gesture for selecting a single word of text can be configured by means of regular expressions, thus allowing the user to select other spans of text than just single words (S-KNW13).

Example 2



Create Layer Assistant

Coding Object

Do you want to assign features to the document as a whole (e.g., its language, document type, etc.), or assign features to segments within the file?

○ Annotate Document
● Annotate Segments
○ Automatic Grammar Analysis (English only)
○ Rhetorical Structure Annotation

Automatic Segmentation: Words

UAM Corpus Tool: Automatic segmentation assistant for paragraphs, clauses, NPs and words (S-UAM03).

Known uses

MMAX2

Related patterns

- Internal relations: -
- External relations: This pattern uses the "Wizard" pattern (Van Welie & Trætteberg 2000; Tidwell 2011), the "Shield" pattern (Van Welie & Trætteberg 2000), and the "Streamlined repetition" pattern (Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns/pmwiki.php?n=PmWiki.Semi-automaticCreationOfAnchors
Page last modified on June 20, 2014, at 08:21 PM EST

P5.2 - Conventionalized actions for selection of anchors

Problem description

In order to define an anchor in the primary data, the user has to select a span of text. There are conventionalized actions for the selection of text, known from other types of software, such as word processors or text editors. If the expectations of the user about certain actions and system behaviors are not met by the annotation tool, the system is harder to learn and less efficient to use.

Related problems: CAT15, GLO10, UAM15, WOR08

Usability principle

Consistency and standards, error prevention

Context

This pattern can be used to implement an interaction behavior that allows the user to specify anchors in the primary data that can then be used as the target for annotation values from the annotation scheme. As the task of creating an anchor is very similar to a *text selection* task, the tool should adhere to established interaction conventions.

Forces

- The selection of anchors is essentially a task where a span of text is selected.
- There are conventionalized interaction-behaviors for the selection of text.
- The selection of anchors is oftentimes intertwined with other subtasks of the annotation process.
- Anchors are not only specified once, but may also be modified after their initial creation.

Solution

The tool supports conventionalized actions for common "selection-related gestures" (cf. Tidwell 2005, p. 256):

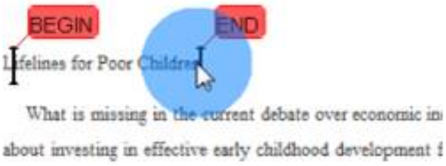
- *Double click* on a word = select this word
- *Click* (= start selection here), *drag* (= select all text in between), *release* (= end selection here)
- *Shift-click* = "start selection at the text intersection cursor, end it at the click point, and select all text between them"

In addition to these conventionalized gestures, a single click on an existing anchor typically selects the text span that is within that anchor's scope.

Rationale

By supporting conventionalized actions for the selection of text, that are known from other types of software, the annotation tool is easier to learn (*learnability*). Also, the overall annotation speed (*efficiency of use*) is increased, as users can build on interaction behaviors that are already known from similar tools.

Example



Lifelines for Poor Children

What is missing in the current debate over economic in

about investing in effective early childhood development f

Glozz: The tool provides visual aids for the click-drag-release gesture, by displaying a small "begin" flag for the click action, and an "end" flag for the release action (S-GLO02).

Known uses

GATE, Knowtator

Related patterns

- Internal relations: -
- External relations: This pattern uses "Habituation" pattern (Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.ConventionalizedActionsForCreationOfAnchors
Page last modified on September 12, 2014, at 07:48 PM EST

P5.3 - Feedback for creation of anchor scope

Problem description

The creation and modification of anchors as addressable spans of text in the primary data is an integral part of the annotation process. Yet, oftentimes it is unclear if the scope of the specified anchor is correct, as there is no appropriate feedback from the tool.

Related problems: -

Usability principle

Visibility of system status, recognition rather than recall

Context

This pattern can be used to implement a visual feedback mechanism that reassures the user about the scope of the anchor he has just created.

Forces

- Once an anchor has been specified, it may be unclear what the actual scope is, as the visualization of anchors is oftentimes insufficient.
- The user wants to make sure that the selected anchor scope is correct before he applies an annotation value in the next step.

Solution

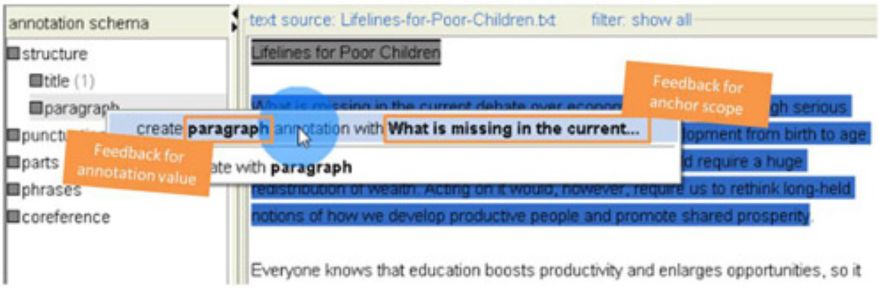
The tool provides some kind of visual feedback for the selected anchor span in the primary data. This is most commonly realized by highlighting the selected portion of text in a neutral color. This behavior is also known from other editors that allow the user to select spans of text. Another way to provide additional feedback is by displaying the selected anchor scope as a text string in an additional window or field, e.g. the annotation scheme window, from which an annotation value for the selected anchor has to be selected.

Rationale

By providing instant feedback on the creation of an anchor scope, it is less likely for the user to produce erroneous anchors and annotations. Thus, the overall *error* potential is reduced and users are more likely to achieve their desired annotation goals.

Example

Example 1


annotation schema
structure
title (1)
paragraph
punctu
parts
phrases
coreference
text source: Lifelines-for-Poor-Children.txt
filter: show all
create paragraph annotation with What is missing in the current...
ate with paragraph
Feedback for annotation value
Feedback for anchor scope

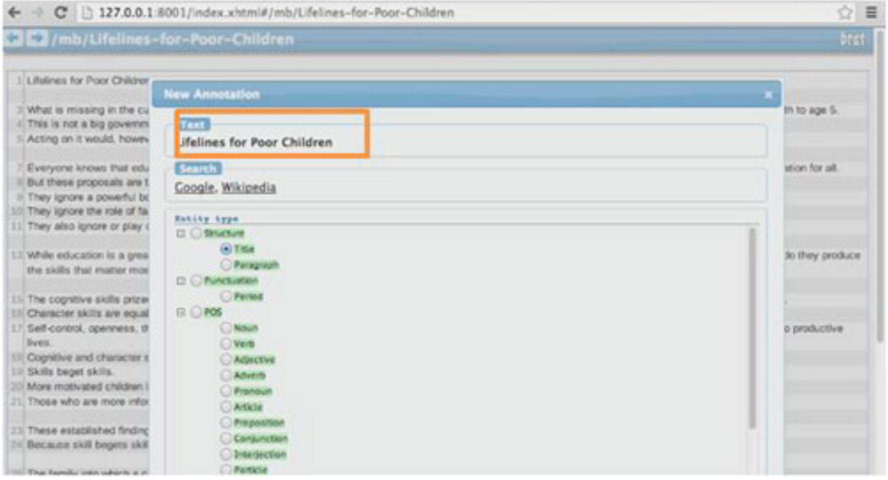
Lifelines for Poor Children

What is missing in the current debate over econom... gh serious ... lopment from birth to age ... d require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it

Knowtator: Feedback for annotation value and anchor scope (S-KNW07).

Example 2

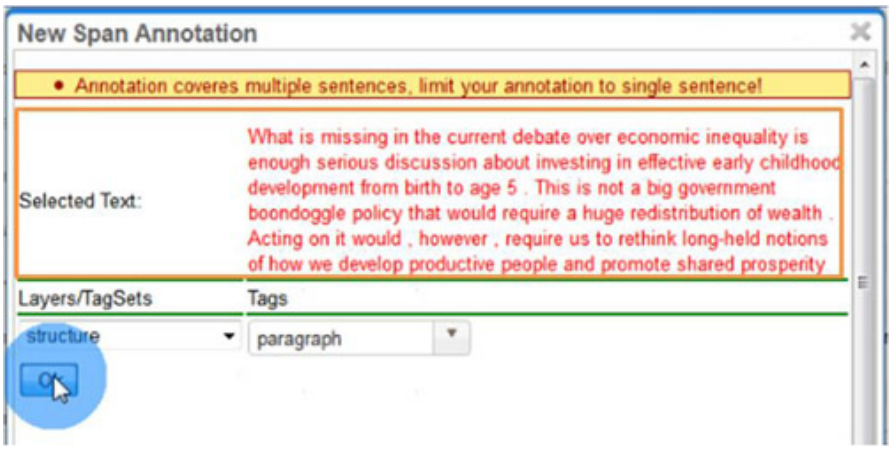


Text: Lifelines for Poor Children

○ Structure
● Title
○ Paragraph

Brat: Feedback for anchor scope in the annotation scheme window (S-BRA10).

Example 3



• Annotation coveres multiple sentences, limit your annotation to single sentence!

Selected Text: What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5 . This is not a big government boondoggle policy that would require a huge redistribution of wealth . Acting on it would , however , require us to rethink long-held notions of how we develop productive people and promote shared prosperity

Layers/TagSets: structure

Tags: paragraph

WebAnno: Feedback for anchor scope in the annotation scheme window (S-WEB12).

Related patterns

- Internal relations: -
- External relations: -

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.FeedbackForCreationOfAnchorScope
Page last modified on June 20, 2014, at 08:29 PM EST

P5.4 - Accelerators for selecting and applying annotations

Problem description

The process of selecting an annotation value from the annotation scheme by means of a mouse action and its application to an anchor (which requires another mouse action) can be very cumbersome and inefficient.

Related problems: CAT14, DEX11

Usability principle

Flexibility and efficiency of use

Context

This pattern can be used to implement accelerators for the process of selecting and applying annotation values to anchors in the primary data.

Forces

- Novice users may be unsure which annotation value to apply to which anchor.
- In order to avoid accidental annotations (cf. "shield" pattern), most tools require the manual selection of an annotation value and also a confirmation of its application to an anchor.
- Both these actions can become very annoying if large numbers of anchors have to be annotated, as they slow down the annotation process significantly.

Solution

This pattern describes three solutions that are not meant to be mutually exclusive:

Solution A: In case the user wants to annotate multiple anchors with the same annotation value, it can be very annoying to explicitly select that value from the annotation scheme. The annotation tool remembers the last selected value and uses it as default value for the consecutive annotation process, unless the user explicitly selects a different value from the scheme (cf. S-WEB09).

Solution B: The tool provides an *accelerated* annotation mode for experienced annotators that allows the user to apply a selected annotation value without explicit confirmation, i.e. whenever a value is selected from the annotation scheme, it is automatically applied to the previously selected anchor. This specific annotation mode can be entered and ended with a simple action such as a click on a button. Such a mode implies that the annotator is confident about the application of the annotation values to specific anchors, as accidental annotations are more prone to occur (due to the lack of an explicit confirmative action).

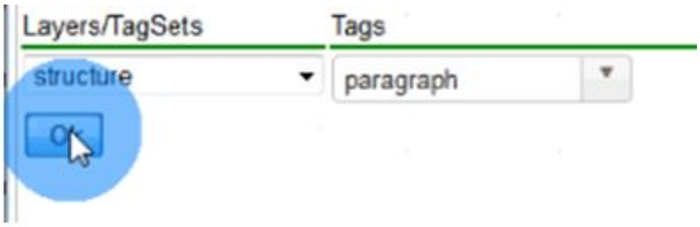
Solution C: The tool provides custom keyboard shortcuts that allow the user to select specific annotation values from the scheme, e.g. "n" for "noun". As, typically, there are many different annotation values in a scheme, it is unrealistic to define a shortcut for each value, which would be hard to remember during the annotation process. Therefore, it is important to allow the user to define custom shortcuts for selected annotation values. Combined with solution A, the user can speed up the annotation process significantly by enabling the non-confirmative annotation mode and by skipping the cumbersome interaction step of selecting an annotation value from the annotation scheme by means of a mouse gesture: the user may now select an anchor with the mouse (first hand) and apply a specific annotation value by means of a keyboard shortcut (second hand).

Rationale

The suggested solutions can speed up the annotation process significantly (*efficiency of use*), but at the same time increase the chance of accidental annotations that will have to be corrected manually. The solutions are therefore meant as accelerators for experienced annotators.

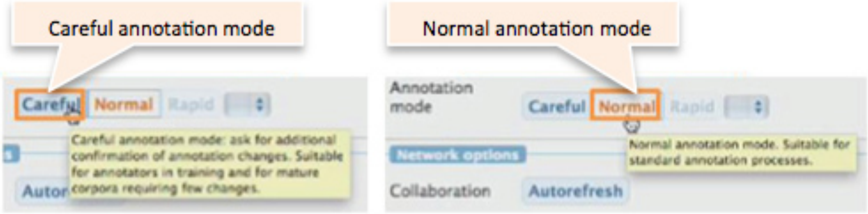
Example

Example for solution A



WebAnno: The last selected annotation value automatically becomes the new default value, and may be applied to consecutive anchors (S-WEB09).

Example for solution B

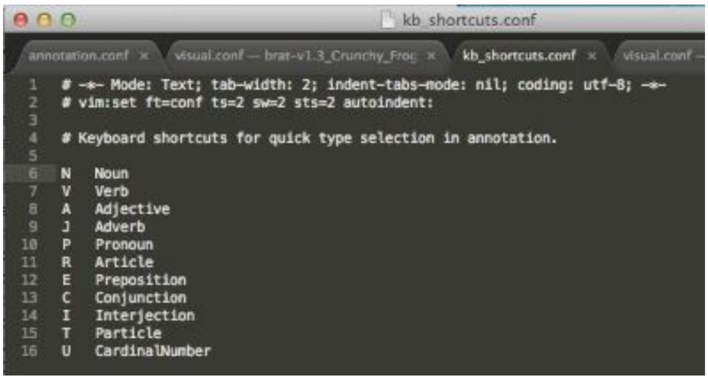


Careful annotation mode: ask for additional confirmation of annotation changes. Suitable for annotators in training and for mature corpora requiring few changes.

Normal annotation mode. Suitable for standard annotation processes.

Brat: There are two different modes of annotations: (1) in "Careful" mode, each annotation must be confirmed with an "ok" button, (2) in "Normal" mode, each annotation is applied as soon as a radio button has been selected (S-BRA08).

Example for solution C



```
# -*- Mode: Text; tab-width: 2; indent-tabs-mode: nil; coding: utf-8; -*-
# vim:set ft=conf ts=2 sw=2 sts=2 autoindent:

# Keyboard shortcuts for quick type selection in annotation.

N   Noun
V   Verb
A   Adjective
J   Adverb
P   Pronoun
R   Article
E   Preposition
C   Conjunction
I   Interjection
T   Particle
U   CardinalNumber
```

Brat: The tool allows the user to define custom shortcuts in an external config file (S-BRA08).

Known uses

WordFreak (S-WOR07) and Dexter (S-DEX01) provide pre-defined shortcuts, but do not allow the user to define custom shortcuts (cf. DEX11).

Related patterns

- Internal relations: -
- External relations: This pattern uses the "Streamlined repetition" pattern (Tidwell 2011), the "Keyboard only" pattern (Tidwell 2011), and the "Good defaults" pattern (Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns /pmwiki.php?n=PmWiki.AcceleratorsForSelectingAndApplyingAnnotations
Page last modified on September 15, 2014, at 09:20 PM EST

P5.5 - Accelerators for annotating multiple anchors

Problem description

Most annotation tools require some kind of confirmative action before an annotation value is actually applied to an anchor. In annotation scenarios where several anchors are to be annotated with the same annotation value, the manual confirmation to apply every single value is very tiresome and annoying.

Related problems: -

Usability principle

Flexibility and efficiency of use

Context

This pattern can be used to implement a function that speeds up the process of applying the same annotation value to multiple anchors.

Forces

- Users want to apply a specific annotation value to a number of anchors in the primary data (exemplary scenario: "annotate all adverbs in the primary data at once").
- To apply an annotation value to an anchor, the user has to confirm the application of each annotation. This "shield" mechanism is reasonable in order to avoid accidental annotations.
- At the same time it is very annoying to confirm the application of an annotation value every time, especially if it stays the same for multiple anchors.

Solution

This pattern describes two solutions that are not meant to be mutually exclusive:

Solution A: The tool provides an annotation mode that allows the user to apply one selected annotation value to multiple anchors. This specific annotation mode can be entered and ended with a simple action such as the click on a button.

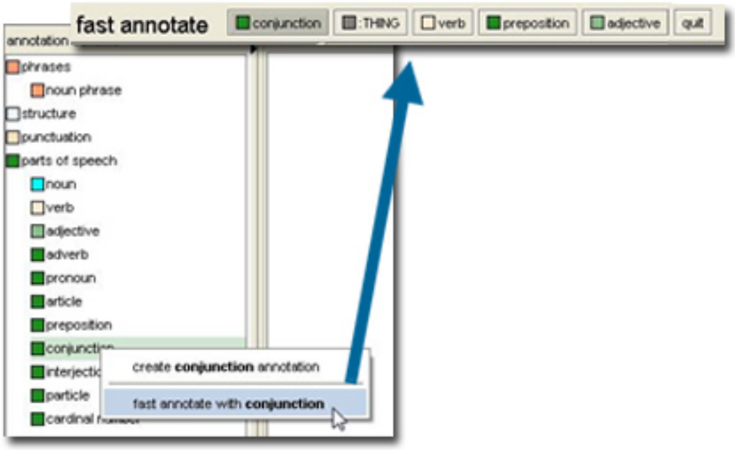
Solution B: The tool allows the user to apply a selected annotation value to similar anchors in the primary data automatically, in the fashion of a "find-replace" dialogue that is known from various text editors. The user can choose whether the selected value is to be applied to the "previous", "next" or "all" anchors that are equal (on basis of the text string) to the selected anchor scope.

Rationale

The suggested solutions can speed up the annotation process significantly (*efficiency of use*), but at the same time increase the chance of accidental annotations that will have to be corrected manually. The solutions are therefore meant as accelerators for experienced annotators.

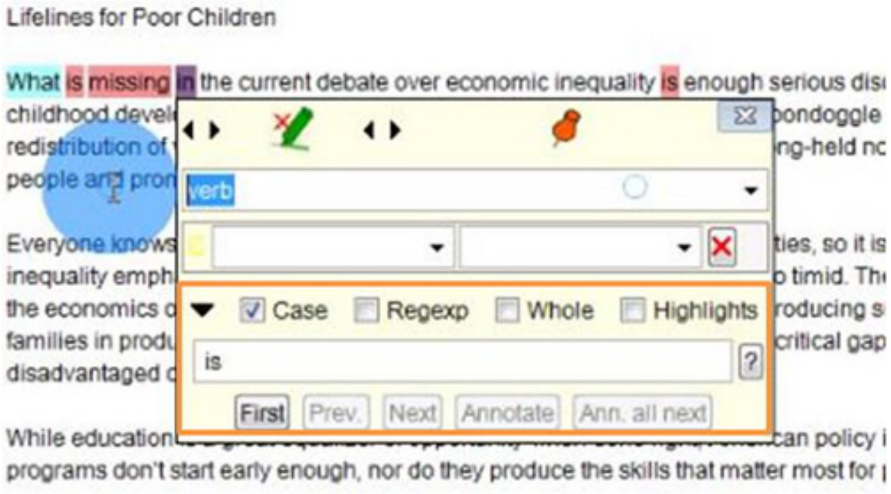
Example

Example for solution A



Knowtator: When applying an annotation value, there is always the option to enable "fast annotation mode". When this mode is enabled for one specific item of the annotation scheme, the value is automatically applied to each anchor selection. The fast mode can be ended via the "Quit" button on top of the annotation window. All tags that have been selected for "fast annotation" previously are also displayed on top of the annotation window, thus allowing the user to quickly switch between different annotation values in "fast mode" (S-KNW10).

Example for solution B



Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious dis
childhood devel oondoggle
redistribution of ng-held nc
people and pron

Everyone knows ties, so it is
inequality emph o timid. Th
the economics c roducing s
families in produ critical gap
disadvantaged c

While education can policy i
programs don't start early enough, nor do they produce the skills that matter most for

GATE : There is a function for the annotation of similar anchors with the same annotation value. This can speed up the manual annotation process significantly. The anchor can be formulated by means of the sophisticated GATE search expression syntax (S-GAT01).

Related patterns

- Internal relations: This pattern refines P5.1
- External relations: This pattern uses the "Streamlined repetition" pattern (Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.AcceleratorsForAnnotatingMultipleAnchors
Page last modified on September 12, 2014, at 02:26 PM EST

P5.6 - Access to existing annotations

Problem description

As the annotation of an anchor is never an isolated task, but rather happens in the context of a larger annotation scenario, the user needs to have access to existing annotations, in order to facilitate a consistent annotation.

Related problems: -

Usability principle

Visibility of system status, error prevention

Context

This pattern can be used to provide contextual information during the annotation process, which is derived from existing annotations.

Forces

- Typically (for an exception cf. P5.5), the user annotates one anchor after another.
- With a growing number of annotations, the user typically wants to know about previous annotation values, in order to annotate subsequent annotations consistently.

Solution

This pattern describes three solutions that are not meant to be mutually exclusive:

Solution A: The tool allows the user to quickly navigate between annotated anchors of one selected type by using arrow controls or arrow keys (cf. S-KNW06). By clicking on the "left" arrow, the user jumps to the previous anchor with that annotation value; clicking on the "right" arrow, the user jumps to the next anchor in the primary data.

Solution B: The tool provides an overview of the total counts for each annotation value in the annotation scheme (cf. S-KNW08). Such counts have two functions: First, they motivate the user as they give explicit feedback on the number of existing annotations. Second, the user can see if certain annotation values are prone to be used more often than others, which facilitates some annotation decisions and also indicates that a combined annotation of multiple anchors (cf. P5.4) might be viable.

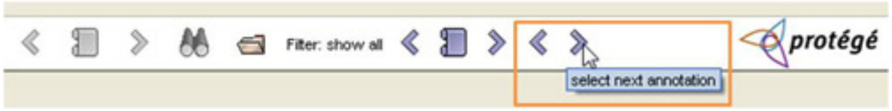
Solution C: Before a selected annotation value is applied to an anchor, the tool shows a list of anchors that have been annotated with that value before in a contextual menu (cf. S-KNW09). This facilitates the decision whether the intended annotation is a viable choice, as previous usages of the same annotation value are presented. At the same time, the user may recognize that the annotation value was applied erroneously in a previous case, and correct that error to improve the consistency of the overall annotated document.

Rationale

The suggested solutions help to access existing annotations more quickly, which has a positive effect on the tool's overall *efficiency of use*.

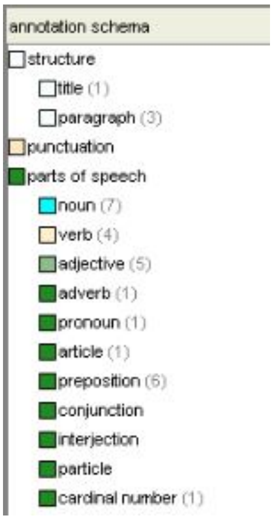
Example

Example for solution A



Knowtator: It is possible to jump from one annotation (of a selected type) to the next / previous annotation by using two small arrow icons, which are positioned in the top left corner of the main window (S-KNW06).

Example for solution B

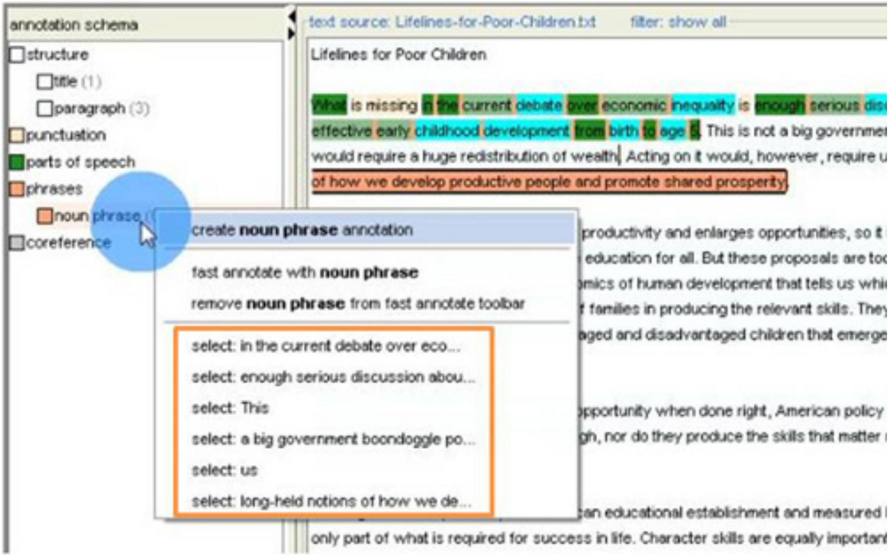


annotation schema

- □ structure
  - □ title (1)
  - □ paragraph (3)
- □ punctuation
- 🟩 parts of speech
  - 🟦 noun (7)
  - □ verb (4)
  - 🟩 adjective (5)
  - 🟩 adverb (1)
  - 🟩 pronoun (1)
  - 🟩 article (1)
  - 🟩 preposition (6)
  - 🟩 conjunction
  - 🟩 interjection
  - 🟩 particle
  - 🟩 cardinal number (1)

Knowtator: There are counts behind each item in the annotation scheme that indicate the number of existing annotations of this type (S-KNW08).

Example for solution C


annotation schema
structure
title (1)
paragraph (3)
punctuation
parts of speech
phrases
noun phrase
coreference
text source: Lifelines-for-Poor-Children.txt
filter: show all
create **noun phrase** annotation
fast annotate with **noun phrase**
remove **noun phrase** from fast annotate toolbar
select: in the current debate over eco...
select: enough serious discussion abou...
select: This
select: a big government boondoggle po...
select: us
select: long-held notions of how we de...

Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious dis
effective early childhood development from birth to age 5. This is not a big governme
would require a huge redistribution of wealth. Acting on it would, however, require u
of how we develop productive people and promote shared prosperity.

productivity and enlarges opportunities, so it
education for all. But these proposals are to
mics of human development that tells us whi
families in producing the relevant skills. They
aged and disadvantaged children that emerge

pportunity when done right, American policy
gh, nor do they produce the skills that matter

an educational establishment and measured
only part of what is required for success in life. Character skills are equally importan

Knowtator: When a new annotation is applied, the context menu shows a list of anchors that were already annotated with the same annotation value (S-KNW09).

Related patterns

- Internal relations: This pattern refines P6.1
- External relations: This pattern uses the "Context menu" pattern (Van Welie & Trætteberg 2000) and the "List browser" pattern (Van Welie & Trætteberg 2000)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.AccessToExistingAnnotations
Page last modified on September 12, 2014, at 02:27 PM EST

P5.7 - Assistance for modification of anchors

Problem description

While the creation of a new anchor can be accomplished by conventionalized gestures (P5.2), the modification of the scope of an existing anchor requires an advanced interaction concept and oftentimes is not available at all. In these cases, the anchor (and potentially existing annotations) first needs to be deleted, and then needs to be created anew. This is very annoying and time-consuming, as all annotations have to be applied again to this anchor.

Related problems: UAM18

Usability principle

User control and freedom, flexibility and efficiency of use

Context

This pattern can be used to implement a function that assists the user with modifying the scope of an existing anchor, whithout having to delete it.

Forces

- After an anchor has been created, and after an annotation value has been applied to this anchor, it may become necessary to adjust (increase / decrease) the scope of the anchor.
- Modifying an anchor scope requires an advanced interaction concept that many tools fail to provide.

Solution

This pattern describes two solutions that are not meant to be mutually exclusive:

Solution A: The tool provides small arrow controls that can be used to increase or decrease the scope of a selected anchor for both, its left and its right side. The anchor is typically selected via mouse click.

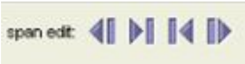
Solution B: The tool provides small handles that appear on the left and on the right side of a selected anchor. The scope can be adjusted by dragging the handles in the respective direction to increase or decrease the anchor scope. It is important to provide handles that are large enough to be a drag-target for the user (cf. UAM18 for a negative example where the handles are too small).

Rationale

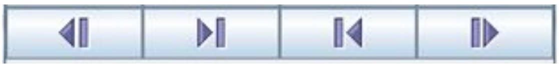
By providing an interaction concept that allows the user to modify existing anchor scopes, the annotation process is speeded up (*efficiency of use*).

Example

Examples for solution A

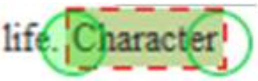


WordFreak: Arrow controls can be used to increase or decrease the anchor to its left or right side (S-WOR06).



Knowtator: Arrow controls can be used to increase or decrease the anchor to its left or right side (S-KNW12).

Example for solution B



Glozz: There are visual aids in the form of small circles. These circles can be dragged like handles and allow the user to adjust the scope of the anchor to either the left or the right side (S-GLO03).

Known uses

GATE

Related patterns

- Internal relations: -
- External relations: -

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.AssistanceForModificationOfAnchors
Page last modified on September 12, 2014, at 02:28 PM EST

P5.8 - Integrated delete annotation function

Problem description

During the annotation process, the user may apply erroneous annotation values to an anchor. The deletion of existing annotations, however, requires complex interactions, as it is part of the overall annotation process.

Related problems: CAT16, DEX12, WEB18, WOR11

Usability principle

Flexibility and efficiency of use, aesthetic and minimalist design

Context

This pattern can be used to avoid cumbersome tool interactions for the deletion of existing annotation values.

Forces

- The annotation process typically involves the creation of anchors and the application of annotation values. Occasionally it may, however, occur that annotation values have been applied erroneously, which makes it necessary to delete an existing annotation value.
- Many standard interaction gestures are already used for the creation of anchors and the application of annotations (cf. WEB18).

Solution

As the deletion of annotations is part of the overall annotation process, it needs to be integrated seamlessly into the process of creating annotations: Whenever the user clicks on an anchor, he is able to see all existing annotations in a context menu or a permanently visible inspector window (note: this is supported by most tools). The window with the existing annotation values also provides either a conventionalized icon (e.g. a trashcan) or a "delete" button that allows the user to delete the selected value. Additionally, it is possible to delete the selected value by pressing the DEL key (conventionalized gesture for the deletion of items, known from many other tools; cf. S-GLO05).

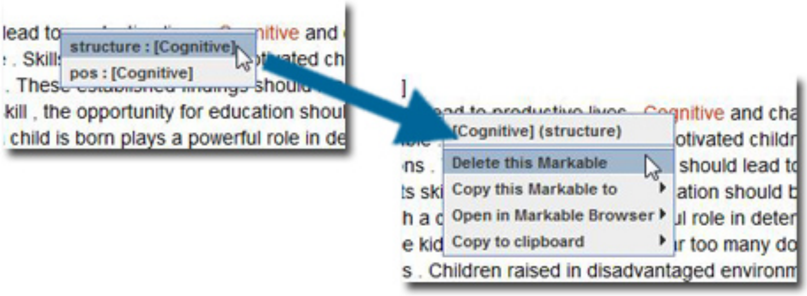
A dialog that asks for confirmation ("Shield") before the value is deleted is not necessary, as it slows down the interaction process. In case a value is deleted accidentally, it can be applied anew with little effort.

Rationale

Integrating the deletion of annotations into the overall annotation process increases the *efficiency of use* as well as the *memorability* of the delete function.

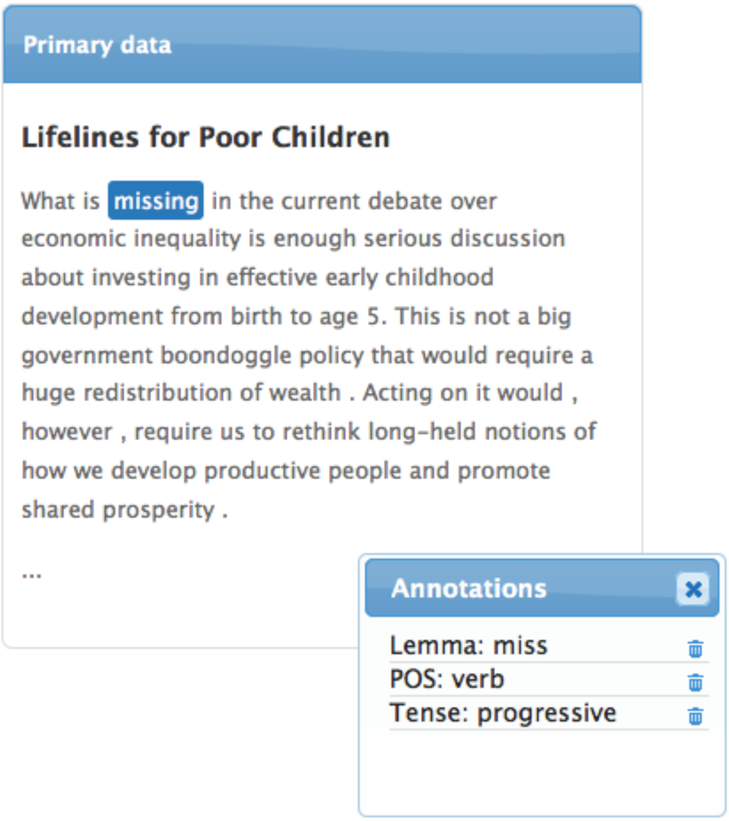
Example

Example



MMAX2: Right-clicking an anchor shows all existing annotations, which may be deleted in a second step.

Interactive prototype



Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth . Acting on it would , however , require us to rethink long-held notions of how we develop productive people and promote shared prosperity .

...

Lemma: miss

POS: verb

Tense: progressive

Prototype P5.8 - The prototype allows the user to open a context menu for an annotated anchor. The context menu displays exemplary annotation values, which may be deleted by the user.

Related patterns

- Internal relations: -
- External relations: This pattern uses the "Context menu" pattern (Van Welie & Trætteberg 2000)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.IntegratedDeleteAnnotationFunction
Page last modified on September 12, 2014, at 08:06 PM EST

P5.9 - Mass deletion of annotations

Problem description

The deletion of multiple annotations is a tedious task when every value has to be deleted separately (cf. P5.8).

Related problems: -

Usability principle

- Flexibility and efficiency of use

Context

This pattern can be used to facilitate the deletion of multiple annotation values at once.

Forces

- The user wants to delete multiple annotation values of one type or even for a whole annotation level at once, as he realizes that a phenomenon has been annotated erroneously in a consistent way.
- Most of the time, the user wants to delete only single annotation values.

Solution

Make the deletion of single annotation values as easy as possible and integrate this action into the overall annotation process (cf. P5.8). For the less frequent task of deleting multiple annotation values at once, provide an advanced interaction mode that is hidden by default. If the user chooses to enter this "mass deletion mode", he may specify which type of annotations are to be deleted. Ideally, the user can choose to delete annotations according to the following criteria:

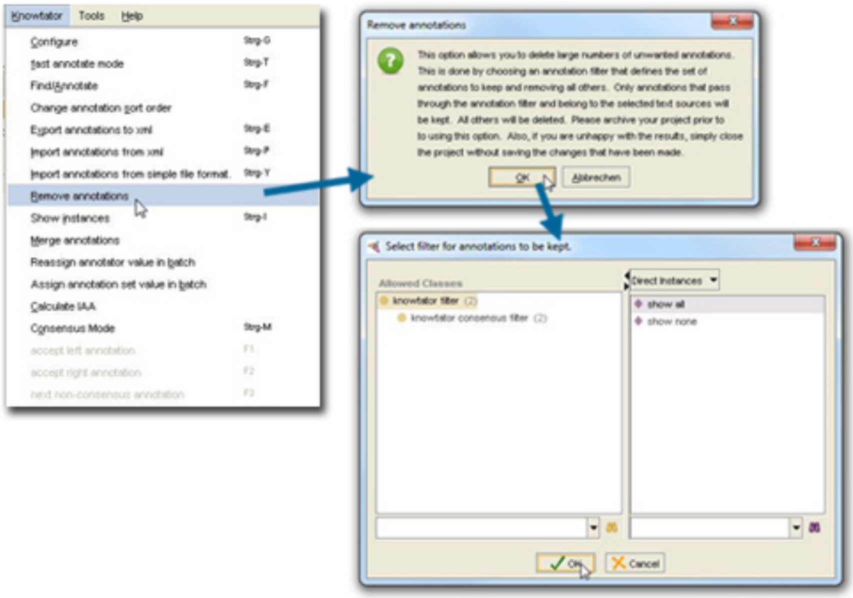
- Delete all annotations according to a specific value.
- Delete all annotations according to a specific anchor.
- Delete all annotations according to a hierarchical (most likely *parent*) unit, e.g. a whole level of annotation (cf. P4.2).

As opposed to the deletion of single annotation values, it is recommended to implement a "shield" function that requires the user to explicitly confirm the deletion of multiple annotations at once. This shield helps to avoid accidental deletions of large numbers of annotation values. If the deletion takes longer than a few seconds, the tool provides user feedback on the progress of the deletion during that process.

Rationale

By providing a function that allows the user to delete multiple annotation values at once, the task of "annotation deletion" can be speeded up significantly (*efficiency of use*). At the same time, accidental deletions might occur (especially with novice user) during such a mass deletion of annotations (increased *error rate*).

Example


Knowtator
Tools
Help
Configure Strg-G
fast annotate mode Strg-T
Find/Annotate Strg-F
Change annotation sort order
Export annotations to xml Strg-E
Import annotations from xml Strg-P
Import annotations from simple file format. Strg-Y
Remove annotations
Show instances Strg-I
Merge annotations
Reassign annotator value in batch
Assign annotation set value in batch
Calculate IAA
Consensus Mode Strg-M
accept left annotation F1
accept right annotation F2
next non-consensus annotation F3
Remove annotations
OK
Abbrechen
Select filter for annotations to be kept.
Allowed Classes
knowtator filter (2)
knowtator consensus filter (2)
Direct Instances
show all
show none
OK
Cancel

This option allows you to delete large numbers of unwanted annotations. This is done by choosing an annotation filter that defines the set of annotations to keep and removing all others. Only annotations that pass through the annotation filter and belong to the selected text sources will be kept. All others will be deleted. Please archive your project prior to to using this option. Also, if you are unhappy with the results, simply close the project without saving the changes that have been made.

Knowtator: Wizard for the mass deletion of selected annotation values (S-KNW14).

Related patterns

- Internal relations: This pattern refines P5.8
- External relations: This pattern uses the "Progress" pattern (Van Welie & Trætteberg 2000), the "Shield" pattern (Van Welie & Trætteberg 2000), and the "Streamlined repetition" pattern (Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.MassDeletionOfAnnotations
Page last modified on September 12, 2014, at 02:32 PM EST

P5.10 - Support annotation of relations

Problem description

Creating a relational annotation between two or more anchors is a cumbersome task, as it requires the selection of multiple anchors and the application of a *relation* annotation value. Such relational annotations become even more challenging when the relation is directed, or when more than two anchors are involved.

Related problems: ANA13, BRA21, GAT19, KNW13, WEB19

Usability principle

Error prevention, recognition rather than recall, flexibility and efficiency of use

Context

This pattern can be used to provide an intuitive and transparent way for the creation of relational annotations between two or more anchors.

Forces

- The user wants to create a relational annotation between two or more anchors.
- The user wants to indicate the direction of the relational annotation, i.e. he wants to (implicitly) annotate different anchor roles.

Solution

The tool provides an intuitive drag-and-drop solution for the creation of relational annotations. As it is necessary to use standard mouse gestures, such as a "left click", for this solution, it is also necessary to provide a special "annotate relations" mode that can be enabled by means of a button or a menu item. When in this mode, hovering over an anchor displays a small handle (e.g. a circle). This handle can be dragged and dropped to another anchor. There is immediate visual feedback, as dragging the handle also draws an arrow that indicates the relation and its direction (cf. P6.3). If the arrow is dropped on another anchor, a relation is created between these anchors. The direction of the arrow goes toward the second anchor, where the relation was dragged to.

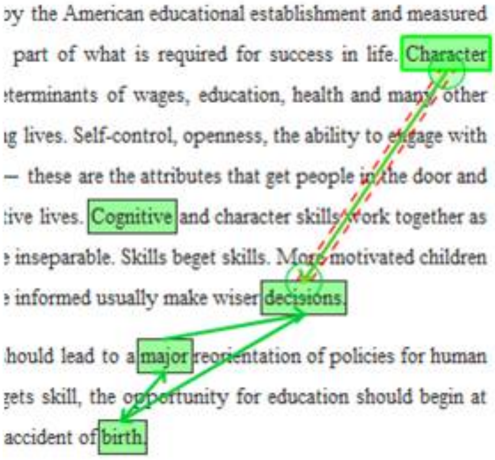
When hovering over an anchor that already has a relational annotation, an additional handle is displayed for the existing relation. This handle can be dragged to another anchor, to redirect the existing relation annotation to that anchor.

Note: For the deletion of relations cf. suggestions in P5.8 and P5.9

Rationale

The creation of relational annotations is a cumbersome task, as it involves multiple anchors. The solution described in this pattern facilitates the creation of such relational annotations and thus increases the *learnability* of the tool as well as its *efficiency of use*.

Example



y the American educational establishment and measured
part of what is required for success in life. Character
terminants of wages, education, health and many other
g lives. Self-control, openness, the ability to engage with
– these are the attributes that get people in the door and
ive lives. Cognitive and character skills work together as
inseparable. Skills beget skills. More motivated children
informed usually make wiser decisions.

hould lead to a major reorientation of policies for human
gets skill, the opportunity for education should begin at
accident of birth.

Glozz: To create a relation, the user only has to select the two anchors he wants to relate. There is a visual aid in the form of an arrow that helps to "draw" the relation. Relations can also be edited by dragging one end of the relation to another anchor. Note: In Glozz, there is a special "create relations" mode that has to be enabled to perform the above actions (S-GLO04).

Known uses

Brat, WebAnno (with minor UI problems: BRA21, WEB19)

Related patterns

- Internal relations: This pattern uses P5.8 and P5.9
- External relations: -

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns/pmwiki.php?n=PmWiki.SupportAnnotationOfRelations
Page last modified on September 12, 2014, at 08:17 PM EST

6.7.2. List of remaining strengths

Note taking *Brat* allows the user to take notes in a free text field (S-BRA09). Such personal notes facilitate the process for applying a specific annotation value and increase the overall consistency of the annotations, as the annotator can use previously created notes as additional guidance during his decision making process.

Monitoring of users and progress Only few tools enable the definition of different user roles, such as *annotator*, *admin*, etc. *WebAnno* not only implements an intuitive user management (S-WEB10), but also provides a feature for monitoring the progress of different users on different annotation documents (S-WEB11). It is possible to monitor the level of agreement for different annotators on different levels of annotation. In *WebAnno*, a "Curation mode" can be used to compare annotations by different annotators, and merge them into a curated version (S-WEB13). *Brat* also provides a "comparison mode", which can be used to view two versions of an annotated document side by side in order to compare differences between annotators (S-BRA11).

Facilitated navigation between anchors While most tools require the user to click on an anchor in order to select it, or to display existing annotations on this anchor, *WordFreak* facilitates the navigation between anchors by allowing the user to jump back and forward between anchors by means of arrow controls or arrow keys (cf. S-WOR05). This feature is particularly helpful when a tool implements the use of keyboard shortcuts during the annotation process (e.g. shortcut for applying an annotation value, cf. P5.4).

6.7.3. List of remaining problem clusters

No undo / redo of actions Although some of the tools utilize conventionalized interaction gestures that are known from other tools (cf. P5.2), none of the tools implemented *undo* and *redo* of actions, which also has been established as a conventionalized function of any type of editing software (ANA10, BRA15, CAT17, DEX13, GAT16, GLO13, KNW09, MAX17, UAM19, WEB21, WOR13).

Switching to different modes As was described at the beginning of this section, the annotation process is characterized by numerous sub-steps that are necessary to achieve the overall annotation task. A good tool needs to support an efficient and integrated annotation process. There are, however, some negative examples that do not quite integrate the different sub-steps into a unified annotation process. For instance, *Analec* (ANA12) and *Glozz* (GLO11) require the user to explicitly switch to different modes when he wants to create new annotations (*annotation mode*) or when he wants to edit existing annotations (*edit mode*). In *WordFreak* (WOR10) it is not possible to annotate the primary data with annotation values from different levels of annotation, without explicitly switching the schemes (for every level of annotation), which is very annoying.

No feedback after the application of an annotation Most tools provide immediate feedback after an annotation value has been applied by displaying the annotation (cf. section 6.8). If such visual feedback is missing (GAT14, ANA11), the user is not sure about whether the correct annotation value has been applied to an anchor.

6.7.4. List of tool-specific problems

Brat There are a number of tool-specific problems and small bugs that occur *Brat* during the annotation process: (1) Applying an annotation value to an anchor as well as deleting an existing annotation value causes a significant delay (BRA16) that slows down the overall annotation process. (2) In order to edit an existing annotation the user has to double click on the label to open the annotation window. The double click does not always work, and sometimes requires several tries (BRA18). (3) To edit the scope of an existing anchor, the user can click the "Move" button in the annotation window that opens when an existing anchor is clicked (BRA19). "Move" is a somewhat unintuitive choice of wording to describe a function that can be used to edit the scope of an anchor. (4) Also, the function named "Add Frag", which is positioned next to the "Move" button, is not worded intuitively (BRA20), i.e. it is not clear what it can be used for. There are no tooltips that explain the irritating functions. (5) A final problem can be observed whenever annotations are nested, as they produce an error hint in the context menu that is displayed when the user hovers over an annotation (BRA17). It is not clear how the error affects the overall annotation process, or how it can be resolved.

CATMA Applying annotations in *CATMA* is not intuitive (CAT13): Although described in the manual, it is not clear to the user that he needs to select an anchor in the left "Document pane", and then apply an annotation value from the tag set by clicking on its color symbol. It would be much more plausible to click on the annotation value itself rather than on the nearby color icon.

GATE If the user wants to delete an anchor in *GATE*, he has to hover over the anchor to open the annotation window. The window contains a small icon that shows a green pen with a small red cross (GAT18). It is not clear that the user has to click on this icon to delete the selected anchor. Whenever possible, tool designers should rely on established icon designs (e.g. a small trashcan or a red cross to indicate a "delete" action).

Knowtator *Knowtator* provides some advanced modes, such as "Required mode" and "Consensus mode"; it is, however, not clear how these modes work (KNW10). During the annotation process, a bug could be observed that prevented the selection of anchors from time to time (KNW11). The *Protégé* framework had to be started anew to get rid of this unwanted tool behavior. The final problem is concerned with the wording of the functions that are involved in deleting annotations: There are two buttons in the annotation window, one which says "Clear annotation" and one which says "Delete annotation". While the "Delete" function obviously deletes an existing annotation, it is not clear (despite a short explanation in a tooltip) what the "Clear" function can be used for (KNW12).

MMAX2 There is a bug concerning the selection of annotation values from the scheme: The first element from the drop-down annotation scheme cannot be selected unless some other value has been clicked before (MAX14). Some advanced functions in *MMAX2* (although roughly described in the manual) remain unclear after the evaluation of the tool (MAX15). These functions are "One-click annotation" and "Anno hint". One major problem of *MMAX2* is the complicated and unintuitive creation of anchors and the application of annotations, as it requires multiple, tedious steps (MAX16): First, the user has to create an anchor (*MMAX2* terminology = *markable*) by means of *click-drag-release*. The user also has to select the annotation level on which the markable is to be created (before being able to apply an actual annotation value). Next, the created markable has to be clicked, to select it. It would be more efficient, if the markable was automatically selected after its creation in the first step. Finally, the user can select a tag from the respective annotation level and assign it to the markable by clicking the "Apply" button. *MMAX2* also makes use of numerous context menus that appear on left and right clicks on either an annotated anchor or a span of text without any annotations. These different applications of context menus (that look very similar) are not quite intuitive and generate serious cognitive overhead, which makes it difficult to understand the context menu concept (MAX18).

UAM Corpus Tool In *UAM Corpus Tool*, a value from the annotation scheme is applied to a previously selected anchor by means of a double click on the value (UAM16). This interaction behavior is not conventionalized, and also not clear to the user. Furthermore, the nearby "Save" button suggests that it could be used to "save an annotation" to a selected anchor, while it really saves the whole project.

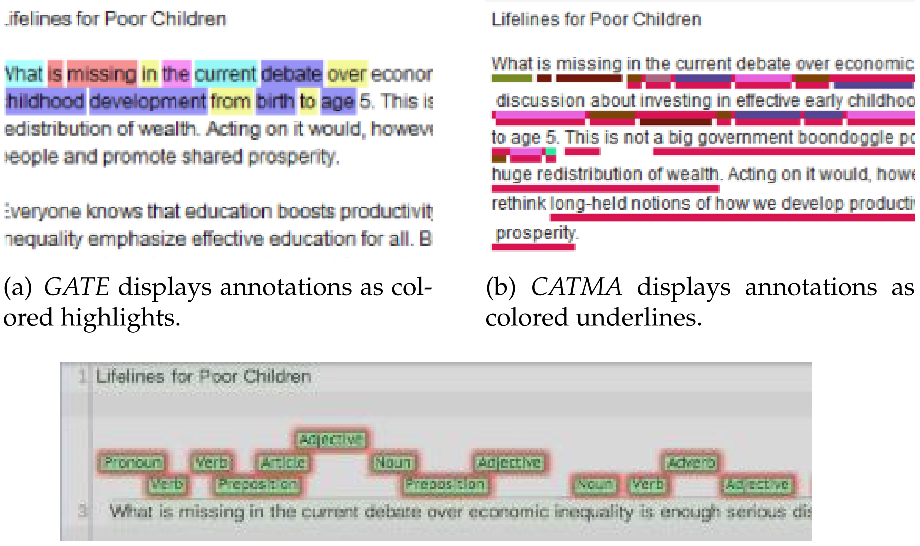
WebAnno Similar to *Brat* (*WebAnno* and *Brat* use a similar annotation engine, also cf. BRA16), there is a significant delay when applying or deleting an annotation (WEB17). Another problem that occurs during the annotation process is that *Brat* jumps to the top of the page (displayed in a web browser) after every successfully applied annotation (WEB20). This is very annoying, especially if the user annotates an anchor that is located further down the page, which means he has to scroll down to the recent primary data position after every single annotation.

WordFreak As was described in P5.7, some tools provide small arrow controls that allow the user to adjust an existing anchor scope (cf. S-WOR06). In addition, *WordFreak* provides another set of arrow controls that can be used to navigate the existing anchors. The buttons can easily be confused, because they are grouped together (WOR09). *WordFreak* may also confuse the user, as it requires previously defined annotations (e.g. parts of speech) as a basis for some other annotations (e.g. noun phrases). It is not clear that there is a technical dependency between these different types of annotations (WOR12).

6.8. Annotation visualization

Viewing annotations

After an annotation value has been applied to an anchor, the user should be able to view the annotation. Most tools display the annotations directly in the primary data (cf. Figure 6.8), either by means of colored highlights and underlines (*CATMA*, *Dexter*, *GATE*, *Glozz*, *Knowtator*), or by displaying explicit labels with the annotation value in the primary data (*Brat*, *WebAnno*, *WordFreak*).



ifelines for Poor Children

Vhat is missing in the current debate over econor
hildhood development from birth to age 5. This is
edistribution of wealth. Acting on it would, howev
eople and promote shared prosperity.

Everyone knows that education boosts productivit
nequality emphasize effective education for all. B

(a) *GATE* displays annotations as colored highlights.

Lifelines for Poor Children

What is missing in the current debate over economic
discussion about investing in effective early childhoo
to age 5. This is not a big government boondoggle pc
huge redistribution of wealth. Acting on it would, howe
rethink long-held notions of how we develop producti
prosperity.

(b) *CATMA* displays annotations as colored underlines.

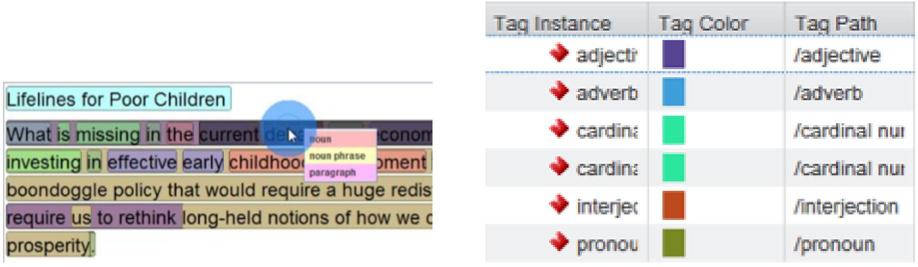
1 Lifelines for Poor Children

3 What is missing in the current debate over economic inequality is enough serious dis

(c) *Brat* displays labels with annotation values above the anchors.

Figure 6.8.: Examples for annotation visualization in the primary data.

There are, however, more ways to visualize annotation values that can be combined with previously described approaches (cf. Figure 6.9). While some tools display existing annotations in a context menu (*Brat*, *Dexter*, *Knowtator*, *MMAX2*, *WebAnno*), other tools display the values in a separate window or pane (*CATMA*, *GATE*, *Glozz*, *Knowtator*), or as highlighted value in the annotation scheme window (*Analec*, *GATE*, *MMAX2*, *UAM Corpus Tool*). The first pattern describes how these different visualization strategies can be used to display parallel annotations (i.e. multiple annotations for the same anchor) in a way the user can easily identify all existing annotation values (P6.1). As such a parallel display is inclined to look crammed, users should also have the option to customize which annotations are displayed by being able to *show* and *hide* certain annotation values or even whole annotation levels (P6.2).



Lifelines for Poor Children

What is missing in the current de… …conom

investing in effective early childhoo… …pment

boondoggle policy that would require a huge redis

require us to rethink long-held notions of how we c

prosperity.

Tag Instance	Tag Color	Tag Path
adjecti	🟪	/adjective
adverb	🟦	/adverb
cardin:	🟩	/cardinal nui
cardin:	🟩	/cardinal nui
interjec	🟥	/interjection
pronou	🟩	/pronoun

(a) *Dexter* displays all annotation values for one anchor in a context menu.

(b) *CATMA* displays annotation values in a separate annotation window.

Figure 6.9.: Examples for annotation visualization by means of context menus and separate annotation windows.

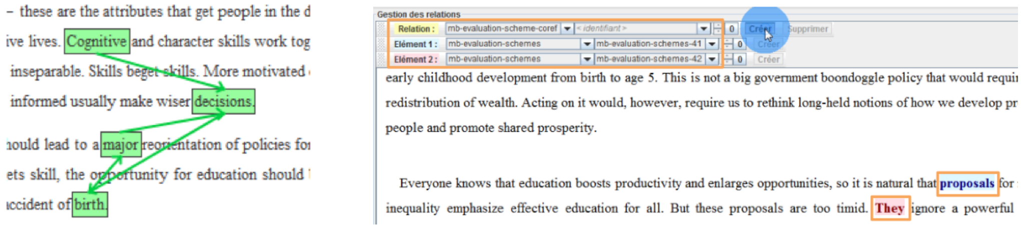
Relational annotations

The visualization requirements for relational annotations are somewhat different from the previously described annotations, as they involve two or more anchors, which are connected by at least one directed relation, such as "coreference". If at all[190], most tools visualize relational annotations by means of (colored) edges and arrows that connect the anchors that are part of the relation (cf. Figure 6.10). The practice to use *indices*, that could be observed for some participants in the handwritten annotation experiment (cf. chapter 2.8.4), was not implemented by any of the evaluated tools[191]. In the previous experiment, indices were primarily used to realize relations across two pages, which could hardly be achieved by means of arrows. It seems that none of the tools provides a function to establish relations across several documents, but rather support inter-document relational annotation[192].

[190] Cf. the overview of tool features in chapter 4.6.1; only eight of the eleven tested tools supported the annotation of relations between anchors at all.

[191] An example for a tool that visualizes annotations by means of indices is reported in *CorefDraw* (Harabagiu et al., 2001, p. 4)

[192] The terminology used here is based upon *inter-document chains* and *cross-document chains*,



– these are the attributes that get people in the d
ive lives. Cognitive and character skills work tog
inseparable. Skills beget skills. More motivated
informed usually make wiser decisions.
ould lead to a major reorientation of policies fo
ets skill, the opportunity for education should
ccident of birth.

(a) *Glozz* uses directed arrows to visualize relations between two or more anchors.

Gestion des relations

Relation : mb-evaluation-scheme-coref < identifiant > 0 Créer Supprimer
Elément 1 : mb-evaluation-schemes mb-evaluation-schemes-41 0 Créer
Elément 2 : mb-evaluation-schemes mb-evaluation-schemes-42 0 Créer

early childhood development from birth to age 5. This is not a big government boondoggle policy that would requi
redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop pr
people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for
inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful

(b) *Analec* uses colored highlights to visualize the anchors in the primary data. Relational annotations are displayed on top of the screen.

Figure 6.10.: Examples for the visualization of relational annotations.

Visualizing relation annotations directly in the primary data seems to be well suited as immediate visual feedback for the application of a relational annotation value to two or more anchors (cf. P5.10). It has, however, been noted that the predominantly "text-based visualization" (Burkovski & Heidemann, 2011, p. 693) has several drawbacks when it comes to tasks such as document navigation or error detection (cf. Witte & Tang, 2007), which can be better addressed with alternative visualization techniques (P6.3).

as mentioned in Witte & Tang (2007, p. 3), where a chain is a set of coreferences within a document.

6.8.1. Usability patterns

Three usability patterns were identified for this category:

P6.1 Visualization of parallel annotations

P6.2 Tailored display of annotations

P6.3 Alternative visualizations of relational annotations[193]

Overview of the usability problems and tool-specific strengths that were used as input for the patterns:

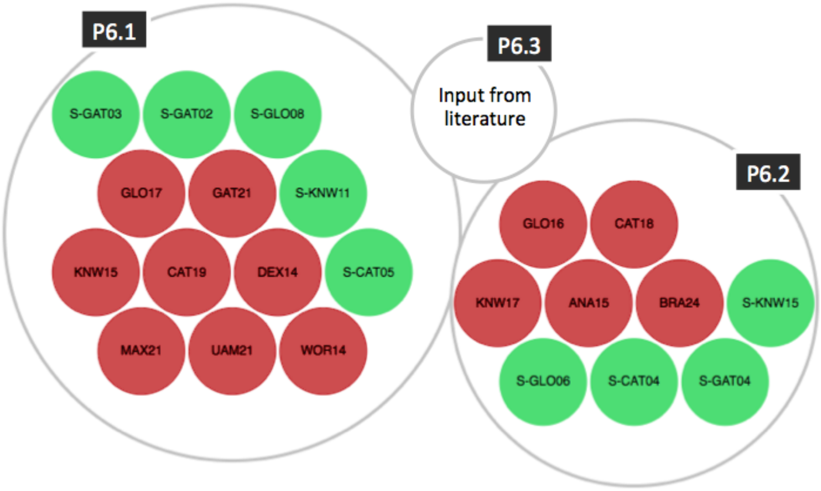


Figure 6.11.: Usability problems and strengths used as input for patterns in category "Annotation visualization".

[193] Note: This is the only pattern that is inferred merely on the basis of related literature.

P6.1 - Visualization of parallel annotations

Problem description

Typically, anchors are annotated on multiple levels of annotation, i.e. one anchor can have several parallel annotations. These parallel annotations are hard to visualize inside the primary data document. Most tools provide either insuffcent visualization for parallel annotations or allow the user to display only one annotation at once.

Related problems: CAT19, DEX14, GAT21, GLO17, KNW15, MAX21, UAM21, WOR14

Usability principle

Visibility of system status, aesthetic and minimalist design

Context

While it is possible to display one annotation value per anchor by means of different colors or by providing labels that contain the annotation value (cf. S-BRA13, S-WEB15, S-WOR08), parallel annotations generally require a more advanced visualization concept. This pattern can be used to implement an annotation visualization that allows the user to view multiple, parallel annotations on one anchor.

Forces

- An anchor can be annotated on multiple levels of annotation.
- Displaying all existing annotations of an anchor at once can cram the interface and irritate the user.

Solution

This pattern describes four solutions that are not meant to be mutually exclusive:

Solution A: The tool uses colored underlines instead of colored highlights to indicate that an anchor has a specific annotation value. Underlines can be stacked to display parallel annotations. The order of underlines must be consistent throughout the whole primary data document (cf. CAT19).

Solution B: Show parallel annotations in a context menu that is displayed next to the respective anchor. The context menu appears when the anchor is clicked or hovered over. The annotation values are displayed as text strings in the context menu, and may also be highlighted in the respective color of the annotation value. Alternatively, the annotation values may be displayed in a separate window or pane rather than in a context menu (also cf. S-GAT02).

Solution C: The tool provides a tabular or list view for existing annotations in a separate window. This view also contains information such as the related anchor scope or the annotation's ID.

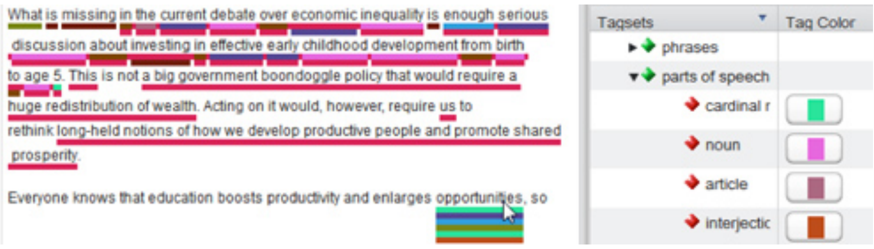
Solution D: The tool provides an annotation stack view that displays an anchor and (optionally) some of its left and right textual context in the horizontal dimension. In the vertical dimension, parallel annotation values are displayed as a stack of different annotation levels.

Rationale

By visualizing multiple, parallel annotations for one anchor, the user has more control about the annotation process and is therefore less likely to produce annotation errors (*error rate*).

Example

Example for solution A

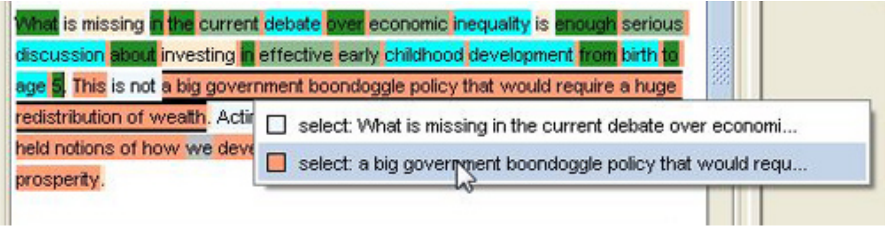


What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so

CATMA: Parallel, colored underlining of annotations (S-CAT05).

Examples for solution B



What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acti

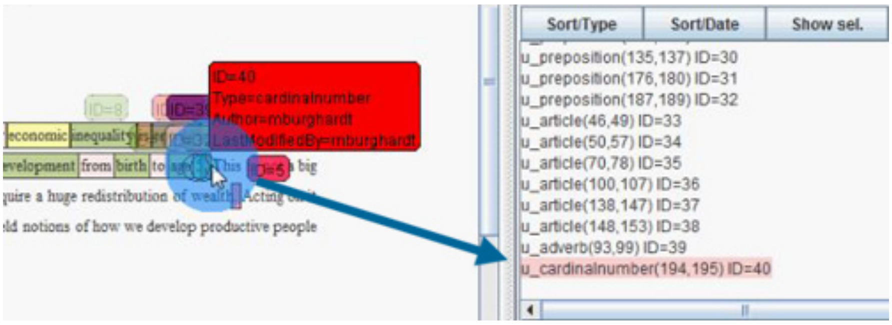
held notions of how we deve

prosperity.

☐ select: What is missing in the current debate over economi...

☐ select: a big government boondoggle policy that would requ...

Knowtator: Context menu shows all existing, parallel annotations for a selected anchor (S-KNW11).

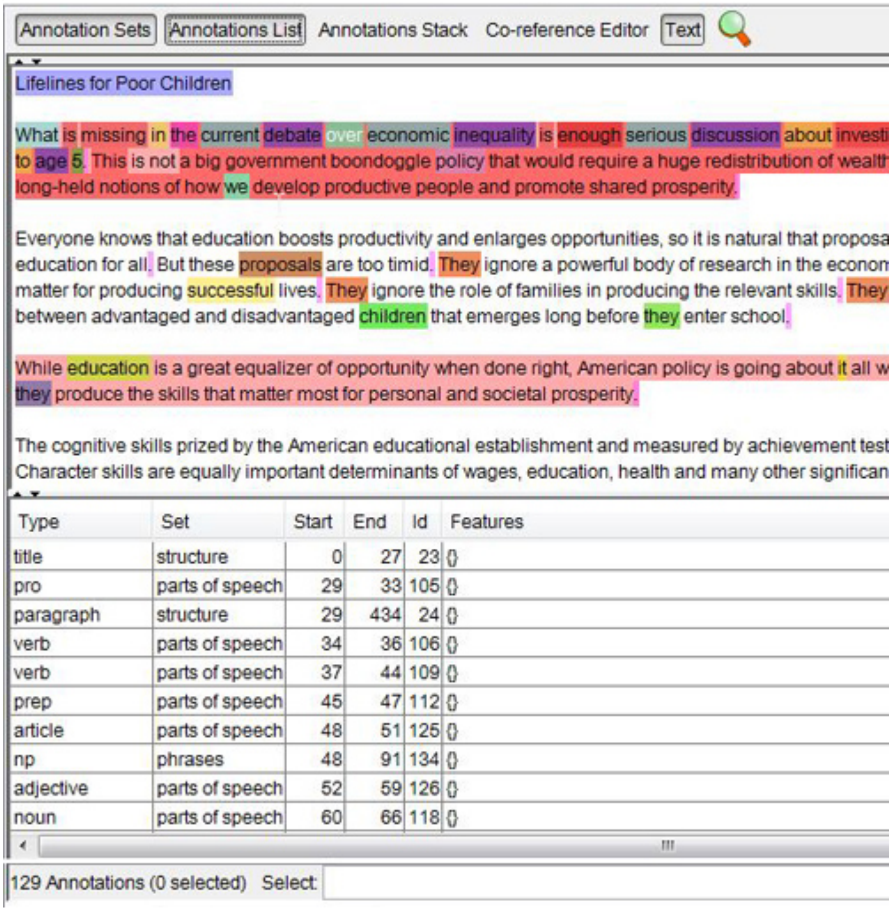

Sort/Type
Sort/Date
Show sel.
economic inequality
velopment from birth to age 5 This is not a big
quire a huge redistribution of wealth. Acting on
ld notions of how we develop productive people
ID=8
ID=39
ID=5

ID=40
Type=cardinalnumber
Author=mburghardt
LastModifiedBy=mburghardt

```
u_preposition(135,137) ID=30
u_preposition(176,180) ID=31
u_preposition(187,189) ID=32
u_article(46,49) ID=33
u_article(50,57) ID=34
u_article(70,78) ID=35
u_article(100,107) ID=36
u_article(138,147) ID=37
u_article(148,153) ID=38
u_adverb(93,99) ID=39
u_cardinalnumber(194,195) ID=40
```

Glozz: Structured, tabular view of annotations and relevant meta data (S-GLO08).

Example for solution C



Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investi
to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth
long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposa
education for all. But these proposals are too timid. They ignore a powerful body of research in the econom
matter for producing successful lives. They ignore the role of families in producing the relevant skills. They
between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all w
they produce the skills that matter most for personal and societal prosperity.

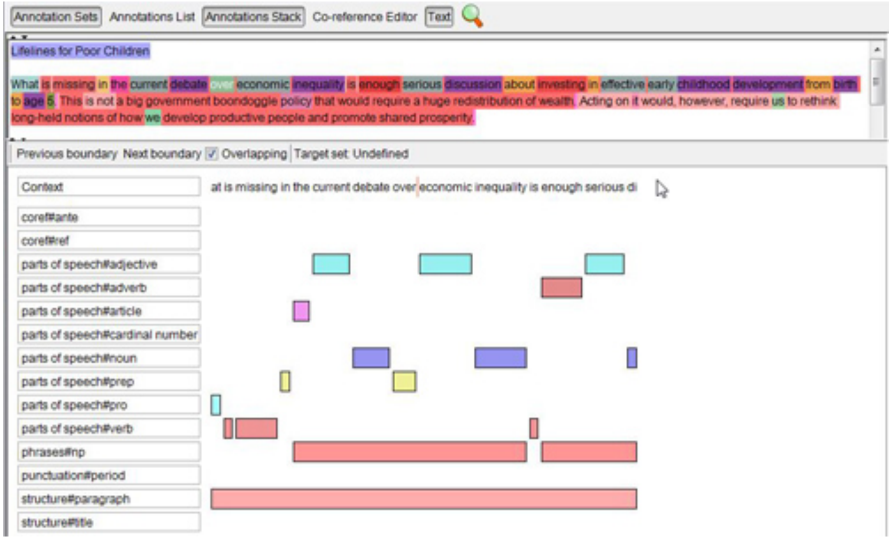
The cognitive skills prized by the American educational establishment and measured by achievement test
Character skills are equally important determinants of wages, education, health and many other significan

Type	Set	Start	End	Id	Features
title	structure	0	27	23	{}
pro	parts of speech	29	33	105	{}
paragraph	structure	29	434	24	{}
verb	parts of speech	34	36	106	{}
verb	parts of speech	37	44	109	{}
prep	parts of speech	45	47	112	{}
article	parts of speech	48	51	125	{}
np	phrases	48	91	134	{}
adjective	parts of speech	52	59	126	{}
noun	parts of speech	60	66	118	{}

129 Annotations (0 selected) Select:

GATE: Structured, tabular view of annotations and relevant meta data (S-GAT02).

Example for solution D



Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Context: at is missing in the current debate over economic inequality is enough serious di

GATE: Parallel annotations are displayed in a stack view with different layers (S-GAT03).

Related patterns

- Internal relations: -
- External relations: This pattern uses the "Context menu" pattern (Van Welie & Trætteberg 2000) and the "Datatips" pattern (Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns /pmwiki.php?n=PmWiki.VisualizationOfParallelAnnotations
Page last modified on September 12, 2014, at 08:26 PM EST

P6.2 - Tailored display of annotations

Problem description

Annotation values are typically visualized in the primary data document. However, displaying too many annotations at once results in a crammed interface that irritates the user and slows down the annotation process.

Related problems: ANA15, BRA24, CAT18, GLO16, KNW17

Usability principle

User control and freedom, aesthetic and minimalist design

Context

This pattern can be used to implement a tailored display of specific annotation values from specific annotation levels.

Forces

- Most annotation projects comprise many different annotation values on several different levels of annotation.
- While it is helpful to be able to view some specific annotation values during the annotation process, which can be used as contextual information for further annotations, the display of all existing annotation values at the same time is rather irritating.

Solution

The tool provides a function that allows the user to show or hide certain types of annotation values, or even a whole annotation level. Although many tools implement a function that can be used to tailor the display of annotations, these are oftentimes difficult to access (cf. CAT18, GLO16, KNW17).

This pattern describes two solutions that are not meant to be mutually exclusive:

Solution A: An easy-to-use function that can show or hide annotations is integrated in the annotation scheme, from which concrete annotation values are selected during the annotation process. The tool provides an unobstrusive checkbox near every annotation item and near every annotation level, to allow for easy showing and hiding of annotations. The checkbox is labeled with a meaningful titel, such as "visibility", at the top of the annotation scheme. Alternatively, the checkboxes can be replaced by meaningful icons that indicate two different states: (1) annotation is visible, (2) annotation is not visible.

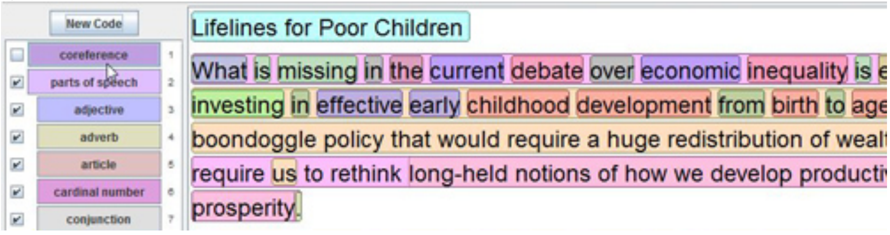
Solution B: Another way to implement a function that allows the user to manage the visibility of annotations is by providing a dialog that can be used to create different show / hide filters. The main difference to solution A is that the dialog is not directly integrated in the annotation scheme, but rather has to be called explicitly by the user.

Rationale

By providing a way to tailor the display of annotations in the primary data, the user can reduce cognitive overhead that is created by too much annotation information that may not even be relevant for the current annotation task. This has a positive effect on the tool's *efficiency of use*.

Example

Examples for solution A

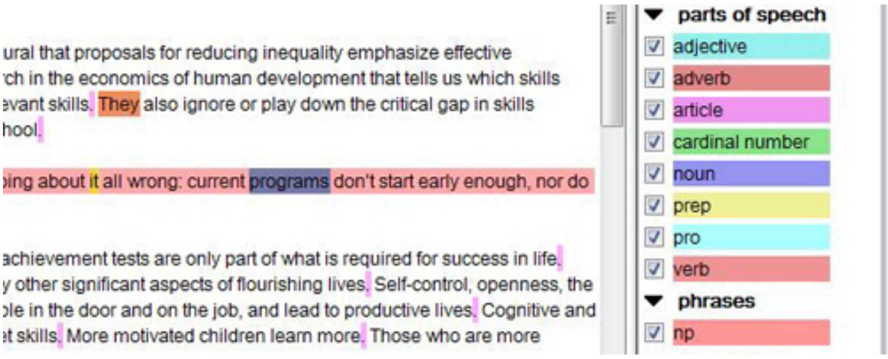


□ coreference 1
☑ parts of speech 2
☑ adjective 3
☑ adverb 4
☑ article 5
☑ cardinal number 6
☑ conjunction 7

Lifelines for Poor Children

What is missing in the current debate over economic inequality is e
investing in effective early childhood development from birth to age
boondoggle policy that would require a huge redistribution of wealt
require us to rethink long-held notions of how we develop producti
prosperity.

Dexter: All annotation items can be shown or hidden by means of checkboxes that are positioned in the annotation scheme.


ural that proposals for reducing inequality emphasize effective
ch in the economics of human development that tells us which skills
evant skills. They also ignore or play down the critical gap in skills
hool.

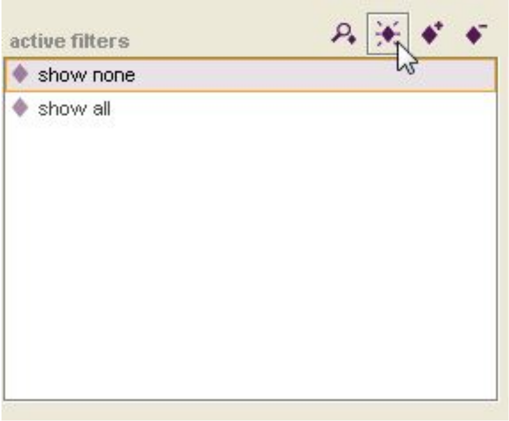
oing about it all wrong: current programs don't start early enough, nor do

achievement tests are only part of what is required for success in life.
y other significant aspects of flourishing lives. Self-control, openness, the
ole in the door and on the job, and lead to productive lives. Cognitive and
it skills. More motivated children learn more. Those who are more

▼ parts of speech
☑ adjective
☑ adverb
☑ article
☑ cardinal number
☑ noun
☑ prep
☑ pro
☑ verb
▼ phrases
☑ np

GATE: All annotation items can be shown or hidden by means of checkboxes that are positioned in the annotation scheme (S-GAT04).

Example for solution B



active filters

- show none
- show all

Knowtator: Dialog for creating filters that affect the visibility of annotation levels and specific annotation items (S-KNW15).

Known uses

CATMA (S-CAT04), Glozz (S-GLO06)

Related patterns

- Internal relations: -
- External relations: This pattern uses the "Preferences" pattern (Van Welie & Trætteberg 2000)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.TailoredDisplayOfAnnotations
Page last modified on September 12, 2014, at 08:37 PM EST

P6.3 - Alternative visualization of relational annotations

Problem description

The predominantly "text-based visualizations" for coreference annotation (cf. Burkovski & Heidemann, 2011) suffer from a major drawback that is described by Witte & Tang (2007, p. 2):

> "Only a part of the coreference chain - for the document text visible within the screen estate - can be viewed. Analyzing larger documents, or cross-document chains, requires permanent scrolling to cover the complete chain, which significantly slows down a developer attempting to gain an overview of all instances within a chain. Moreover, although several chains can potentially be visualized parallel using e.g. different colors, this quickly becomes too visually complex to be useful."

Related problems: -

Usability principle

Aesthetic and minimalist design

Context

This pattern can be used to implement an alternative visualization for relation annotations, that allows the user to display all existing annotations at once, even if they are scattered across multiple documents. Such an alternative view can be helpful for *document navigation* or *correction* tasks.

Forces

Text-based visualizations give immediate visual feedback after a relational annotation has been created. They are, however, inappropriate when it comes to displaying all existing relational annotations, especially if these are *cross-document* annotations.

Solution

Witte & Tang (2007) describe a solution that makes use of *Topic Maps* and *OWL ontologies*. While these two concepts are essentially a theoretical framework that can be used to implement alternative visualizations that address the previously described problems, the core of the solution can be summarized as follows: Display relational annotations in a separate view that is detached from the primary data view. This view displays all existing relational annotation chains as an integrated graph. Such a graph view even allows the user to visualize relation annotations from different documents, and thus greatly facilitates the navigation in coreference chains and documents.

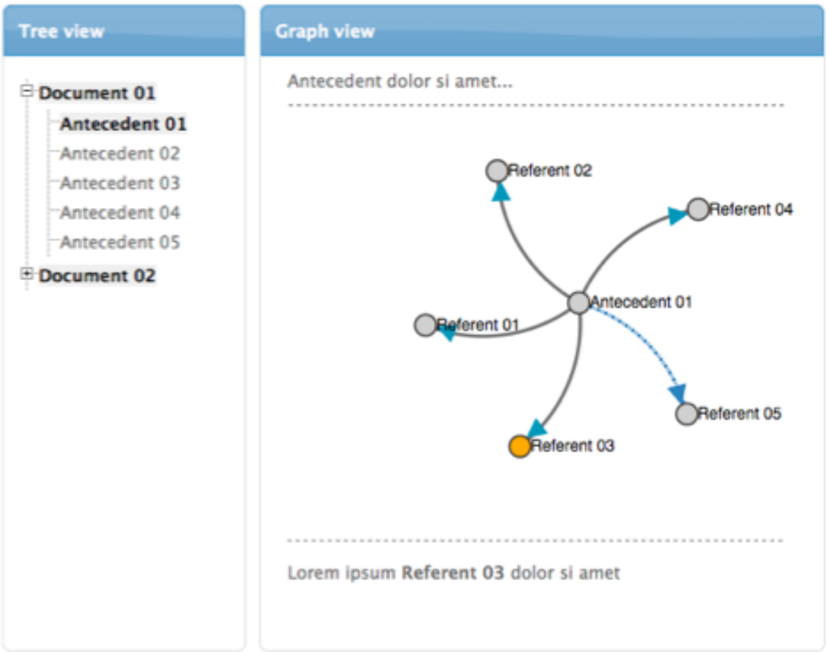
Note: For a semi-automatic coreference annotation and visualization approach also cf. Burkovski & Heidemann (2011), who suggest a combination of *self-organizing maps* (neural-network algorithm for unsupervised machine learning) and *corefenene matrices* (interface for annotation / visualization).

Rationale

Alternative, specialized visualizations for relational annotations help the user to detect and correct errors for this type of annotation, and therefore reduce the overall *error rate*.

Example

Interactive prototype



Prototype P6.3 - The prototype implements an alternative view for the visualization of coreference annotations.

This solution is inspired by the visualizations described by Witte & Tang (2007).

Known uses

Cf. Witte & Tang (2007)

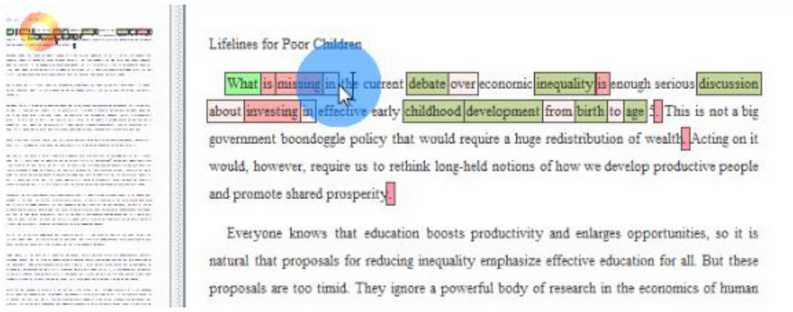
Related patterns

- Internal relations: This pattern refines P3.3 and requires P5.10
- External relations: This pattern uses the "Alternative views" pattern (Tidwell 2011)

Retrieved from http://132.199.139.24/~bum05778/pmwiki-patterns
/pmwiki.php?n=PmWiki.AlternativeVisualizationsOfRelationalAnnotations
Page last modified on September 12, 2014, at 02:48 PM EST

6.8.2. List of remaining strengths

Macro view and positional syncing *Glozz* provides a macro view (cf. Figure 6.12) of the primary data by means of a miniature document that is positioned on the margin of the main annotation interface (S-GLO07). If the user moves the mouse cursor in the primary data, the current position is also indicated in the miniature document. If the user clicks into the miniature, the cursor jumps to the respective position in the original document. Overall, such a macro view of the primary data facilitates the orientation during the annotation process, the positional syncing feature allows for quick navigation in large documents.



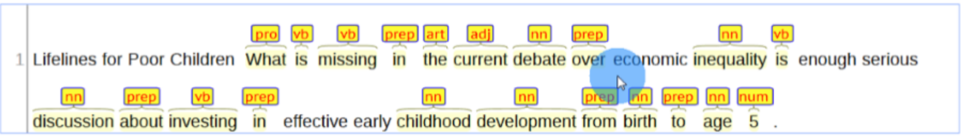
Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human

Figure 6.12.: *Glozz's* macro view and positional syncing feature (on the left side).

Display of annotation values as textual labels While the predominant way of visualizing annotations in the primary data is by means of colored highlights or underlines, some tools provide labels which contain the actual annotation values as text strings (S-BRA13, S-WEB15, S-WOR08, cf. Figure 6.13). Such textual labels can be combined with colors and are much easier to interpret than mere color coding. However, this kind of annotation visualization only works for small amounts of annotations, as too many labels are inclined to cram the interface. To avoid labels with particularly long textual values, *Brat* and *WebAnno* allow the user to define short forms for annotation values, which are then displayed in the labels (S-BRA13, S-WEB15).

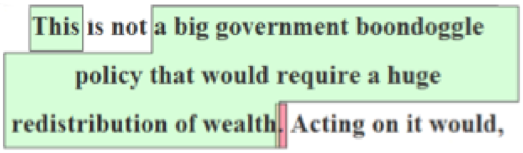


1 Lifelines for Poor Children What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5 .

Figure 6.13.: *WebAnno* as an example for an annotation tool that displays annotation values as small labels, directly above the anchors in the primary data.

Advanced color scheme *Knowtator* allows the user to set colors for single annotation items, or even for a whole annotation level (S-KNW16). The colors can be specified individually by means of RGB values, or by selecting values from a predefined palette.

Visualization for multiline annotations If an anchor spans two or more lines in the primary data, the associated annotation needs to be visualized in an adequate way. *Glozz* provides a very good solution by highlighting different annotations with different colors, but at the same time providing a visual demarcation (cf. Figure 6.14).



This ıs not a big government boondoggle
policy that would require a huge
redistribution of wealth. Acting on it would,

Figure 6.14.: *Glozz's* visualization of multiline annotations.

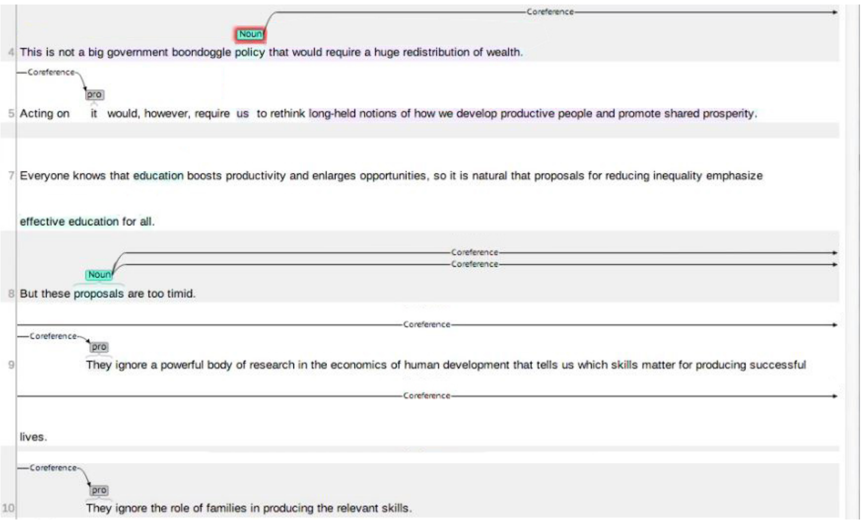
Show annotations on hover Most tools require the user to click an anchor in order to show its associated annotations. By utilizing the *hover* gesture to display annotations, for instance in a context menu, the click gesture can be used to trigger other actions, such as entering a function that allows the user to edit or delete a specific annotation value (cf. S-BRA12, S-WEB14).

Optional XML-view of annotated data *UAM Corpus Tool* provides an option that allows the user to display the existing annotations in an XML code view (S-UAM02). This can be an interesting option for users who are familiar with markup languages, and who would like to know what is happening in the background while annotating via the GUI.

6.8.3. List of tool-specific problems

Analec In *Analec*, annotations are displayed in the annotation scheme window when an annotated anchor is clicked, i.e. the annotations are not visible in the primary data. There is, however, the option to define different colors for different annotation values. The problem is, that these colors are only displayed if no annotation scheme is actively selected, i.e. it is not possible to display annotation values in the primary data during the annotation process (ANA14). This is rather awkward, as existing annotations cannot be used to facilitate the decision making process for further annotations.

Brat, WebAnno As *Brat* and *WebAnno*, both use the same visualization framework, the following problem could be observed for both tools (BRA23, WEB22): Relational annotations that span multiple lines are displayed in a confusing way, as an arrow points until the end of the line, and starts anew in the next line. Relations are hard to recognize, as the user has to follow the arrow across several lines. The visualization becomes even harder to read if multiple relations are visualized in the same line (cf. Figure 6.15).



4 This is not a big government boondoggle policy that would require a huge redistribution of wealth.

5 Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

7 Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all.

8 But these proposals are too timid.

9 They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives.

10 They ignore the role of families in producing the relevant skills.

Figure 6.15.: Problematic visualization of relations that span multiple lines in *Brat*.

GATE Although *GATE* provides a number of different ways to view annotations (cf. P6.3), these are not self-explanatory and may obfuscate novice users (GAT20).

Glozz Color as a means to distinguish different annotations is very important. In *Glozz*, a color value has to be specified for each annotation value; there is no default color for the annotations. The overall process for specifying colors via the "Style editor", and the necessity to explicitly activate the "Individual stylesheet" in the "Color mode" menu are rather cumbersome (GLO15).

MMAX2 In *MMAX2*, basic styling parameters such as color or anchor type, can only be specified once, at the beginning of the project ("Project Wizard"), and only for a whole annotation level. At this point, it is not yet clear how the selected parameters will influence the visualization of the actual annotations, as there is no "Preview" function (MAX19). If the user wants to style the annotations later, during the annotation project, he has to do this outside the tool, by means of an XSL file. Another *MMAX2* problem is concerned with the display of annotations in the primary data, which are very hard to differentiate, as the color is applied to the font rather than to highlights or underlines in the text. If colors with bad contrast on the white background are specified, the text is hardly readable. The display of parallel annotations in the context menu alongside commands that are also listed in a similar context menu is irritating (MAX20). One issue with *MMAX2* (MAX22) concerns the display of relational annotations: While the tool can display relational annotations by means of an arc between two or more anchors, these can only be viewed if the user clicks on an anchor that is part of the relational annotation. It is not possible to display more than one relational annotation at once (cf. P6.3).

Knowtator *Knowtator* revealed some bugs that are related to the display of annotations: The filters for showing or hiding specified annotations do not work properly. On a few occasions, not all annotations were displayed in the text, although they were specified by the filter (KNW14, KNW16). Most of the times, restarting the *Protégé* framework resulted in solving these issues.

UAM Corpus Tool As all annotations are displayed as thin, green underlines in *UAM Corpus Tool*, they are very hard to distinguish. The only way to reveal information about existing annotations, is by clicking on the annotated anchor (UAM20).

WordFreak Apart from the predefined annotation levels "Parts of speech" and "Phrases", annotation values are not displayed in the primary data, but only in the footer area of the main window. This makes it very hard to recognize existing annotations (WOR15).

6.9. Summary

This chapter has presented the main results of this work: a collection of twenty-six usability patterns for the design of annotation tool interfaces. The patterns are also available as a hypertext version in the publicly accessible *annotation usability wiki*. The data items from the heuristics walkthrough evaluation that could not be integrated into a pattern were also documented and discussed in their respective category, alongside with the specific patterns in the same category. The status and validity of the patterns as well as the process of pattern identification will be discussed in the final chapter, which also provides an outlook to future work beyond the scope of this dissertation.

7. Summary and outlook

7.1. Summary of main contributions and discussion

This dissertation has shown that existing linguistic annotation tools suffer from a number of recurring usability problems, but at the same time have their individual strengths. It has been demonstrated that the usability of such tools can be *engineered* systematically by means of an analytic evaluation method and a structured pattern identification approach. This chapter provides a summary and discussion of the main contributions, which are organized according to the research agenda that has been introduced in chapter 1. The dissertation concludes with a discussion of opportunities for future work.

7.1.1. Review of annotation theory and implications for linguistic annotation

Summary: There is a large body of existing research that discusses the functions and characteristics of annotations. The review of related literature (cf. chapter 2) showed that most of the research has been dedicated to scenarios of *human handwritten annotation*, which are fundamentally different from linguistic annotation with regard to its degree of formality. Nevertheless, the existing work on handwritten annotations can be analyzed to derive implications for the domain of linguistic annotation. These implications are vital for understanding the anatomy of an annotation as well as the overall annotation process, and are therefore important prerequisites for the design of an evaluation study of linguistic annotation tools (cf. chapters 4.4 and 4.5).

Chapter 3 presented a review of related work on linguistic annotation, including typical standards and associated technologies. A basic workflow for the creation and usage of linguistic corpora was introduced to paint a bigger picture of linguistic annotation and its typical context of application. Finally, the chances and limitations of automatic and manual annotation approaches were discussed: Although automatic approaches can be used successfully for certain types of annotation tasks, a large number of manual annotation tasks remains. At the same time it has shown that the interfaces for such tools typically suffer from severe usability problems and require a high level of technical expertise, making them hard to use for a *traditional* linguist (cf. the related work described in chapter 3.6.3). Essentially, chapter 3 motivated the main theme of the dissertation, which is to enhance the usability of linguistic tools for manual annotation.

Contributions

- Review of related work on annotation theory and discussion of its implications for the domain of linguistic annotation
- Case study on handwritten annotation behavior and implications for the domain of linguistic annotation
- Workflow model for the creation and usage of annotated corpora

7.1.2. Evaluation study for linguistic annotation tools

Summary: Chapter 4 introduced a number of domain-specific parameters that are important for the characterization of linguistic annotation scenarios with regard to usability aspects. Among these parameters are typical user groups for linguistic annotation tools that were extracted from existing literature. A new user group, the *Traditionalist*, was added to this classification. Furthermore, requirements with regard to functionality and usability were extracted from a number of related studies on linguistic annotation tools. Building on these requirements as well as on the domain knowledge about handwritten annotation theory (cf. chapter 2) and linguistic annotation (cf. chapter 3), a basic interaction model, with typical micro-tasks that occur during the annotation process was proposed. In order to identify appropriate test objects for the usability evaluation, a basic classification of linguistic annotation tools and related tools and resources was presented. In addition, a number of criteria were identified that can be used for a more detailed distinction of existing annotation tools. Eventually, a total of eleven linguistic annotation tools was selected from the pool of freely available annotation tools.

After a reflection on appropriate usability testing methods (cf. chapter 4.5.1), I decided to employ an analytic approach using a single evaluator. The main arguments against evaluation approaches that rely on more than one evaluator – empirical approaches as well as analytic approaches – can be summarized as follows: The overall goal of this work is to identify typical problems and strengths of existing annotation tools, in order to derive generic patterns for the design of user-friendly tools. This goal implies a deep and systematic evaluation of multiple tools, which requires evaluators who are fluent in both, the domain of linguistic annotation as well as in the domain of usability testing (also cf. Nielsen, 1992, for his notion of "double specialists"). Testing these multiple tools by means of an empirical user study would be laborious and likely produce rather diverse evaluation results (different users find different problems for different tools) that are more difficult to refine into generic design patterns. As I have knowledge about linguistic annotation as well as about usability evaluation methods, I decided to conduct a series of single evaluator heuristic walkthroughs for the selected annotation tools in order to guarantee a systematic approach and consistency of the results.

Discussion: The main critique with this approach obviously is the issue of subjectivity and bias, as only one evaluator is involved. At the same time it can be argued that multiple evaluators are likely to produce rather heterogeneous results (cf. the notion of the *evaluator effect*, as described by Hertzum & Jacobsen, 2003), which is unfavorable for the identification of typical, recurring usability problems for a specific domain of application. Furthermore, the heuristic walkthrough method has some built-in features to foster objectivity: These features are the usability heuristics and the control questions, which were both used to identify and categorize any of the problems. An additional feature to encourage objectivity is provided by the systematic evaluation procedure, which utilizes the same tasks for all of the evaluated tools.

A related point of criticism concerns the number of identified usability problems: Multiple evaluators will find more usability problems than a single evaluator. There is, however, evidence that a single evaluator will also find a decent number of usability problems when relying on objective usability heuristics (cf. Nielsen, 1994b; Sears, 1997). The trade-off for fewer, but overall more consistent usability problems, seems to be favorable for the identification of typical problems of annotation tools. Along the same lines, the following observation could be made: As many usability problems recur for different annotation tools, the evaluator's awareness for rather similar problems increases during the series of heuristic walkthroughs. At the same time, the number of total usability problems identified per tool does not seem to increase along the timeline of the evaluation study, which indicates that there is no systematic bias in the identification of usability problems that have been discovered before in subsequent tools (cf. Figure 4.8). As only 82 of the 207 problems (and 62 of the 84 strengths) could be used as input for the identified usability patterns (cf. Figure 6.1), it does not seem likely that more usability problems would have increased the total amount of patterns substantially. Nevertheless, the pattern format is flexible enough to include additional usability problems and strengths that might be discovered in future tool evaluations.

While a single evaluator approach was well suited for this dissertation, this does not mean that the pattern identification process described in chapter 5.6 can only be applied by a single person. Keeping in mind the trade-offs discussed above, it also seems reasonable to use multiple evaluators for the heuristic walkthrough, and to refine the evaluation data into usability patterns collaboratively. In the long run, it would be worthwhile to compare the quality and applicability of patterns derived from heuristic walkthrough data with regard to the number of evaluators that were involved in the creation of that data.

Contributions

- Review and synthesis of related work on user groups, requirements, tasks and tool categories for the domain of linguistic annotation
- Classification and identification of distinguishing features for linguistic annotation tools
- Design and execution of a usability evaluation study for linguistic annotation tools

7.1.3. Systematic pattern identification process

Summary: At the end of chapter 4, the quantitative results of the evaluation study were described, followed by a discussion about possible ways to document the qualitative aspects, i.e. the actual problems and strengths of the tested annotation tools. It has been shown that typical *usability problem reports* are likely to be misinterpreted if they do not provide adequate contextual information (Jeffries, 1994). Accordingly, I have argued that using *patterns* as a means to document core results and insights from the evaluation study can avoid such common misinterpretations, as they provide detailed information about the actual problem and its solution. Patterns also document contextual information that allows the reader to understand the reasons behind the documented problem as well as the reasons that justify the proposed solution (cf. Figure 4.14).

Chapter 5 introduced the concept of design patterns in more detail, illustrating that the original idea by Christopher Alexander, who proposed a pattern language for the domain of urban architecture, has been widely adopted in different fields, most notably the software engineering community. It also showed that more recently, the HCI community has discovered patterns as a means to document design knowledge, which brings several advantages over other formats such as guidelines or heuristics (cf. chapter 5.4). Although patterns are used in different fields, the process of pattern identification is largely unclear (cf. chapter 5.6.2). One of the contributions of this work is the definition of a systematic and transparent approach for the identification of patterns, based on usability evaluation data generated through a series of heuristic walkthroughs. While the pattern format provides essential information that is missing in many traditional usability reports, the heuristic walkthrough data integrates very well with this format (cf. Figure 5.2). I proposed a 13-step guide for the systematic identification of usability patterns on basis of heuristic walkthrough data, which lead to a collection of usability patterns for the design of linguistic annotation tools.

Discussion: Although patterns provide a detailed description of a problem and its solution in a generic form that can be understood by tool developers as well as tool users, it must be noted that the systematic identification of such patterns is far more time-consuming than the creation of a traditional usability report: While the preliminary categorization of problems and strengths (cf. chapter 4.5.3) facilitates the identification of patterns (*Clustering phase 1*), the clustering of approx. 300 items according to their content (*Clustering phase 2*) is still a very complex and time consuming task. It has also shown that not all problems and strengths are appropriate as input for a usability pattern. De facto, many of the observed problems are either too specific or too general to be suitable for the derivation of a pattern.

Another point of discussion is the validity and applicability of the patterns. Van Welie & Van der Veer (2003, p. 6) point out that pattern languages are always subjective to a certain degree, as they reflect the mental model of the designer who created it. While the pattern identification approach described in this work is more systematic than other pattern mining approaches (cf. chapter 5.6.1), up to this point, the identified patterns are based solely on the subjective interpretation of the results of a large-scale usability inspection. For a discussion of the validity and applicability of the presented pattern collection it is important to comprehend the creation of design patterns as an iterative process. This process starts with a first (subjective) suggestion of a *pattern candidate*[194], and is successively refined afterwards. In the software engineering and HCI communities, the successive refinement has been formalized as the so called *shepherding process*, which is essentially a reviewing process where the *shepherds* (experienced pattern authors) help to improve the pattern candidates of novice patterns authors (Harrison, 1999). The dynamic nature of patterns as an evolving construct that may change during the course of time has also been described by (Alexander et al., 1977, p. xv):

> Patterns are very much alive and evolving. In fact, if you like, each pattern may be looked upon as a hypothesis.

As for this work, the status of the suggested patterns may also be described as pattern candidates[195] that will have to be refined and enhanced in future shepherding processes by the HCI pattern community as well as the linguistic annotation community. An important contribution, however, is the definition of a systematic and transparent process that makes clear how these *pattern hypotheses* have come about. Beyond the scope of this dissertation, the suggested pattern identification process may also be used in other domains of application where the usability of systems can be evaluated by means of the heuristic walkthrough method described in chapter 4.5.2.

[194] The term *pattern candidates* was adopted from Ratzka (2008), who uses it to denote patterns that were gathered in a top-down fashion from domain-specific theory, and which need to be validated by the observation of several concrete examples.

[195] The candidate status of the patterns is also indicated by the subtitle of the dissertation: "*Toward* Usability Patterns for Linguistic Annotation Tools".

Contribution

- Definition of a systematic process for the identification of usability pattern candidates

7.1.4. Usability patterns for linguistic annotation tools

Summary: The review of annotation theory, the design of an evaluation study, and the definition of a systematic pattern identification approach can all be seen as subordinate objectives for the overall goal of this dissertation, which is to provide advice for the design of user-friendly, linguistic annotation tools. This generic design knowledge has been documented in the form of usability patterns. Chapter 6 presents twenty-six usability patterns as the main results of this work. Besides the documentation of the pattern candidates in this dissertation, the patterns are also publicly available via the *annotation usability wiki*: `http://www.annotation-usability.net`

As an additional contribution, that extends the scope of typical HCI patterns, a number of HTML / JavaScript prototypes were created. Whenever appropriate and technically feasible, these prototypes illustrate the proposed solutions in an interactive way. The prototypes can be accessed via the wiki as well (cf. the "Example" section of the respective patterns).

Discussion: The issue of the validity of patterns has already been addressed in the above section, and will be picked up again in section 7.2. Another point of discussion may be seen in the *completeness* of the pattern collection: While the ultimate goal of any pattern engineer is to create a comprehensive pattern language, most pattern collections[196] do not succeed in covering the design issues of a certain domain of application exhaustively. This limitation also applies for this work. Although the collection of patterns seems quite comprehensive, it still must be noted that most patterns were derived by means of a single-evaluator heuristic walkthrough of eleven annotation tools. It cannot be ruled out that the evaluation of more tools, and an increase of the number of evaluators, might result in more usability patterns for the domain of linguistic annotation. Nevertheless, the pattern collection at hand may serve as a starting point for the community of annotation tool designers, and may be extended via the public wiki platform in the long run.

Contribution

- Creation of a collection of generic usability patterns for the design of user-friendly, linguistic annotation tools

[196] Cf. Kruschitz & Hitz (2009) for a more detailed discussed of *pattern languages* vs. *pattern collections / catalogs*.

7.2. Outlook and future work

The previous section has presented a summary and discussion of the main contributions of this dissertation. Moreover, this work opens a number of opportunities for consecutive studies and future work.

7.2.1. Evaluation and revision of patterns

It was pointed out that the patterns derived from the heuristic walkthrough data have the status of pattern candidates, which will need further revision and improvement through feedback from the community. The technical infrastructure for such a systematic revision has been established by means of the wiki platform, which is well suited for managing and archiving different pattern versions by different authors. A systematic revision of the patterns can be realized in three phases:

1. First, the patterns are revised by the pattern community (cf. *shepherding process*), i.e. they are primarily checked for their form, structure and overall comprehensibility. The pattern community has established a number of regular conferences (e.g. *PLoP*, *EuroPLoP*, *Viking PLoP*, etc.)[197] that are well suited for the submission of the patterns presented in this dissertation.
2. Next, the patterns are presented to the target audience of annotation tool developers. The most obvious way to gather immediate feedback from this group of users is to organize a workshop at one of the popular conferences on computer and corpus linguistics, e.g. the annual *International Conference of the Association of Computational Linguistics* (ACL) or the biannual *International Conference on Language Resources and Evaluation* (LREC).
3. Once the patterns have been revised and adopted by the community, and once annotation tools have been designed with regard to the existing usability patterns, it will also be possible to evaluate the newly created tools, and to derive feedback on the validity of the patterns. This last step of evaluation, however, implies the availability of such new tools, and must therefore be seen as long-term desideratum. Meanwhile, it is viable to to develop more interactive prototypes, similar to those presented in appendix E, and to evaluate single patterns by means of the prototypical realization. Such an approach would also be in accordance with an iterative usability engineering life cycle as proposed by Nielsen (e.g. 1993, p. 93ff.).

[197] Cf. the *Hillside Group* website, available at `http://hillside.net/conferences`, for an overview of all relevant conferences on patterns.

7.2.2. Toward a pattern language for Humanist-Computer Interaction

The scope of this work has been narrowed down to the concrete scenario of linguistic tools for manual annotation tasks. However, the workflow model in chapter 3.5 has illustrated that the creation and usage of corpora typically requires more than just manual annotation of the primary data documents. It would be worthwhile to extend the scope of *annotation usability* to the wider context of *corpus usability*. Particularly interesting – with regard to user-friendly interfaces – are the task contexts of *corpus querying* and *visualization / analysis of results*. While these topics are already the subject of research (cf. e.g. Luhn, 1960; Soehn et al., 2008; Wattenberg & Viégas, 2008; Culy & Lyding, 2010), the main results and insights are documented in a rather unstructured way (case studies, prototypes, etc.). By reformulating this existing design knowledge consistently as usability patterns, it would be possible to create a more comprehensive usability pattern language for the application context of corpus creation and usage. At the same time, the typical user groups identified for linguistic annotation tools are likely to be the same for other branches of humanities disciplines, i.e. *traditional* scholars who refrain from tools and interfaces that require technical expertise and are difficult to use. A number of recent tools and projects already address this issue, by providing easy-to-use interfaces for a number of digital humanities applications (cf. e.g. Ruecker et al., 2011; Bazo et al., 2013; Wilhelm et al., 2013). The ultimate goal would be to document all existing knowledge about good interface design for digital humanities applications in a common format or language. The collection of patterns for linguistic annotation tools presented in this dissertation may be seen as a first step toward a pattern language for *Humanist-Computer Interaction*.

References

Agosti, M., Albrechtsen, H., Ferro, N., Frommholz, I., Hansen, P., Orio, N., et al. (2005). DiLAS: a Digital Library Annotation Service. In J.-F. Boujut (Ed.), *Proceedings of the International Workshop on Annotation for Collaboration – Methods, Tools, and Practices, IWAC '05* (pp. 91–101). CNRS - Programme société de l'information. Available from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.90.5310

Agosti, M., Bonfiglio-Dosio, G., & Ferro, N. (2007). A historical and contemporary study on annotations to derive key features for systems design. *International Journal on Digital Libraries*, *8*(1), 1–19. Available from http://dl.acm.org/citation.cfm?id=1298592

Agosti, M., Coppotelli, T., Ferro, N., & Pretto, L. (2007). Annotations and Digital Libraries: Designing Adequate Test-Beds. In D. H.-L. Goh, T. H. Cao, I. Torvik Sølvberg, & E. Rasmussen (Eds.), *Proceedings of the 10th international conference on Asian digital libraries, ICADL'07: Looking back 10 years and forging new frontiers* (pp. 150–159). Berlin, Heidelberg: Springer. Available from http://link.springer.com/chapter/10.1007%2F978-3-540-77094-7_23

Agosti, M., & Ferro, N. (2007). A formal model of annotations of digital content. *ACM Transactions on Information Systems*, *26*(1), 3–57. Available from http://portal.acm.org/citation.cfm?doid=1292591.1292594

Agosti, M., Ferro, N., Frommholz, I., & Thiel, U. (2004). Annotations in digital libraries and collaboratories–facets, models and usage. In R. Heery & L. Lyon (Eds.), *Proceedings of the 8th ECDL '04* (pp. 244–255). Berlin, Heidelberg: Springer. Available from http://www.springerlink.com/index/AH229DA5JE3KA8Y7.pdf

Aitchison, J. (1994). *Words in the Mind: An Introduction to the Mental Lexicon.* Oxford, Malden: Blackwell.

Alexander, C. (1979). *The Timeless Way of Building*. New York: Oxford University Press.

Alexander, C. (1996). *The Origins of Pattern Theory the Future of the Theory, and the Generation of a Living World.* Transciption of a presentation held at the ACM Conference on Object-Oriented Programs, Systems, Languages and Applications (OOPSLA). Retrieved January 4, 2014, from http://www.patternlanguage.com/archive/ieee/ieeetext.htm

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A Pattern Language.* New York: Oxford University Press.

Bargeron, D., Gupta, A., & Brush Bernheim, A. (2001). *A common annotation framework.* Microsoft Research, TechReport MSR-TR-2001-108. Retrieved June 6, 2014, from `http://research.microsoft.com/pubs/69900/tr-2001-108.pdf`

Barnum, C. M. (2011). *Usability Testing Essentials: Ready, Set... Test!* Amsterdam et al.: Morgan Kaufmann.

Bayle, E., Bellamy, R., & Casaday, G. (1998). Putting it all together: towards a pattern language for interaction design. *ACM SIGCHI Bulletin*, *30*(1), 17–23. Available from `http://dl.acm.org/citation.cfm?id=280580`

Bazo, A., Burghardt, M., & Wolff, C. (2013). TWORPUS – An Easy-to-Use Tool for the Creation of Tailored Twitter Corpora. In I. Gurevych, C. Biemann, & T. Zesch (Eds.), *Proceedings of the 25th International Conference of the German Society for Computational Linguistics and Language Technology, GSCL '13* (pp. 23–34). Heidelberg: Springer. Available from `http://link.springer.com/chapter/10.1007/978-3-642-40722-2_3#page-1`

Bélanger, M.-E. (2010a). *Annotations and the Digital Humanities Research Cycle: Implications for Personal Information Management.* iConference 2010 Posters. Retrieved June 16, 2014, from `http://hdl.handle.net/2142/15035`

Bélanger, M.-E. (2010b). *The Annotative Practices of Graduate Students: Tensions & Negotiations Fostering an Epistemic Practice.* Master's thesis, University of Toronto. Available from `https://tspace.library.utoronto.ca/bitstream/1807/25427/1/Belanger_Marie-Eve_201011_MISt_thesis.pdf`

Benello, J., Mackie, A. W., & Anderson, J. A. (1989). Syntactic Category Disambiguation with Neural Networks. *Computer Speech & Language*, *3*(3), 203–217. Available from `http://www.sciencedirect.com/science/article/pii/0885230889900181`

Bevan, N. (1999). Quality in use: Meeting user needs for quality. *Journal of Systems and Software*, *49*(1), 89–96. Available from `http://www.sciencedirect.com/science/article/pii/S0164121299000709`

Bevan, N. (2009). What is the difference between the purpose of usability and user experience evaluation methods? In T. Gross et al. (Eds.), *In Proceedings of the UXEM Workshop at INTERACT 2009.* Springer. Available from `http://nigelbevan.com/papers/What_is_the_difference_between_usability_and_user_experience_evaluation_methods.pdf`

Bias, R. (1991). Interface-Walkthroughs: Efficient collaborative testing. *IEEE Software*, *8*(5), 94–95. Available from `http://ieeexplore.ieee.org/xpl/`

login.jsp?tp=&arnumber=84220&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel1%2F52%2F2752%2F00084220

Bias, R. (1994). The pluralistic usability walkthrough: Coordinated empathies. In J. Nielsen & R. L. Mack (Eds.), *Usability Inspection Methods* (pp. 63–76). New York: John Wiley & Sons.

Bopp, S. (2010). *Einführung in die Korpuslinguistik mit DeReKo und COSMAS II.* Tutorial am Lehrstuhl für Deutsche Sprachwissenschaft, Universität Augsburg. Retrieved May 28, 2014, from http://www.philhist.uni-augsburg.de/lehrstuehle/germanistik/sprachwissenschaft/mitarbeiter/stelspass/materialien_lehrveranstaltungen/korpuslinguistik_dereko_cosmas2_bopp.pdf

Borchers, J. O. (2000). CHI meets PLoP: An interaction patterns workshop. *ACM SIGCHI Bulletin*, *32*(1), 9–12. Available from http://dl.acm.org/citation.cfm?id=333330

Borchers, J. O. (2001). *A Pattern Approach to Interaction Design*. Chichester: Wiley.

Bottoni, P., Levialdi, S., & Rizzo, P. (2003). An analysis and case study of digital annotation. In N. Bianchi-Berthouze (Ed.), *Proceedings of the Third International Workshop on Databases in Networked Information Systems, DNIS '03* (pp. 216–231). Berlin, Heidelberg: Springer. Available from http://www.springerlink.com/index/u1cu2mx7cexjf01h.pdf

Bradley, J., & Vetch, P. (2007). Supporting Annotation as a Scholarly Tool - Experiences From the Online Chopin Variorum Edition. *Literary and Linguistic Computing*, *22*(2), 225–241. Available from http://llc.oxfordjournals.org/cgi/doi/10.1093/llc/fqm001

Brants, S., Dipper, S., Hansen, S., Lezius, W., & Smith, G. (2002). The TIGER treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories* (pp. 24–41). Available from http://www.coli.uni-saarland.de/publikationen/softcopies/Brants:2002:TT.pdf

Brants, T. (2000a). Inter-annotator agreement for a German newspaper corpus. In *Proceedings of Second International Conference on Language Resources and Evaluation, LREC '00.* Available from http://hnk.ffzg.hr/bibl/lrec2000/pdf/333.pdf

Brants, T. (2000b). TnT - A Statistical Part-of-Speech Tagger. In *Proceedings of the sixth conference on Applied natural language processing, ANLC '00* (pp. 224–231). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from http://dl.acm.org/citation.cfm?id=974178

Brants, T., & Plaehn, O. (2000). Interactive corpus annotation. In *Proceedings of Second International Conference on Language Resources and Evaluation, LREC '00* (pp. 453–459). Available from http://lrec.elra.info/proceedings/lrec2000/pdf/334.pdf

Brants, T., Skut, W., & Uszkoreit, H. (1999). Syntactic annotation of a German newspaper corpus. In *Proceedings of the ATALA Treebank Workshop* (pp. 73–87). Springer Netherlands. Available from `http://link.springer.com/chapter/10.1007/978-94-010-0201-1_5`

Brill, E. (1992). A simple rule-based part of speech tagger. In *Proceedings of the workshop on Speech and Natural Language, HLT '91* (pp. 112–116). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://portal.acm.org/citation.cfm?doid=1075527.1075553`

Brown, W. B., Malveau, R. C., McCormick, H. W., & Mowbray, T. J. (1998). *AntiPatterns - Refactoring Software, Architectures and Projects in Crisis*. New York: Wiley.

Burghardt, M. (2012). Usability Recommendations for Annotation Tools. In *Proceedings of the ACL 2012, 6th Linguistic Annotation Workshop, LAW '12* (pp. 104–112). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dl.acm.org/citation.cfm?id=2392762`

Burghardt, M., Schneidermeier, T., & Wolff, C. (2013). Usability Guidelines for Desktop Search Engines. In M. Kurosu (Ed.), *Proceedings of the 15th International Conference on Human-Computer Interaction, HCII '13* (pp. 176–183). Berlin, Heidelberg: Springer. Available from `http://link.springer.com/chapter/10.1007/978-3-642-39232-0_20`

Burghardt, M., & Wolff, C. (2009a). Stand off-Annotation für Textdokumente: Vom Konzept zur Implementierung (zur Standardisierung?). In C. Chiarcos, E. de Castilho, & M. Stede (Eds.), *Proceedings of the Biennial GSCL-Conference 2009: From Form to Meaning - Processing Texts Automatically* (pp. 53–59). Gunther Narr Verlag. Available from `http://epub.uni-regensburg.de/14223/`

Burghardt, M., & Wolff, C. (2009b). Werkzeuge zur Annotation diachroner Korpora. In W. Hoeppner (Ed.), *Proceedings of the GSCL-Symposium "Sprachtechnologie und eHumanities"* (pp. 21–31). Duisburg: Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen. Available from `http://epub.uni-regensburg.de/6756/`

Burkovski, A., & Heidemann, G. (2011). Visualization for Coreference Annotation. In G. Angelova, K. Bontcheva, R. Mitkov, & N. Nicolov (Eds.), *Proceedings of the Recent Advances in Natural Language Processing, RANLP '11* (pp. 692–697). Association for Computational Linguistics. Available from `http://aclweb.org/anthology//R/R11/R11-1100.pdf`

Burnage, G., & Dunlop, D. (1992). Encoding the British National Corpus. In G. Sampson & D. McCarthy (Eds.), *Corpus Linguistics: Readings in a Widening Discipline* (pp. 149–159). London, New York: Continuum.

Busa, R. A. (1980). The Annals of Humanities Computing: The Index Thomisticus. *Computers and the Humanities*, *14*(2), 83–90. Available from http://dx.doi.org/10.1007/BF02403798

Busa, R. A. (2004). Perspectives on the Digital Humanities. In S. Schreibman, R. Siemens, & J. Unsworth (Eds.), *A Companion to Digital Humanities* (pp. xvi–xxi). Oxford: Blackwell. Available from http://www.digitalhumanities.org/companion/view?docId=blackwell/9781405103213/9781405103213.xml&chunk.id=ss1-1-2&toc.depth=1&toc.id=ss1-1-2&brand=default

Byrne, J. G. (1989). Competitive Evaluation in Industry: Some Comments. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, *33*(5), 423–425. Available from http://pro.sagepub.com/lookup/doi/10.1177/154193128903300541

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human computer interaction*. Hillsdale, NJ et al.: Erlbaum.

Carletta, J., & Isard, A. (1999). The MATE Annotation Workbench: User Requirements. In M. Walker (Ed.), *Proceedings of the ACL Workshop: Towards Standards and Tools for Discourse Tagging* (pp. 11–17). New Brunswick, NJ: Association for Computational Linguistics. Available from https://aclweb.org/anthology/W/W99/W99-0300.pdf

Casaday, G. (1997). Notes on a pattern language for interactive usability. In *Extended Abstracts on Human Factors in Computing Systems, CHI'97* (pp. 289–290). New York: ACM. Available from http://dl.acm.org/citation.cfm?id=1120398

Castilho, R. de, Holtz, M., & Teich, E. (2007). Computational support for corpus analysis work flows: The case of integrating automatic and manual annotations. In *Book of Abstracts GSCL Workshop: Linguistic Processing Pipelines* (pp. 1–4). Available from http://www.ukp.tu-darmstadt.de/fileadmin/user_upload/Group_UKP/publikationen/2009/gscl09-re-etal.pdf

Chen, P. P.-S. (1976). The Entity-Relationship Model - Toward a Unified View of Data Model. *ACM Transactions on Database Systems*, *1*(1), 9–36. Available from http://dl.acm.org/citation.cfm?id=320440

Clark, J., & DeRose, S. (1999). *XML Path Language (XPath), Version 1.0, W3C Recommendation.* World Wide Web Consortium (W3C). Retrieved May 31, 2014, from http://www.w3.org/TR/xpath/

Conte, A., Fredj, M., Hassine, I., Giraudin, J.-P., & Rieu, D. (2002). A tool and a formalism to design and apply patterns. In Z. Bellahsene, D. Patel, & C. Rolland (Eds.), *Proceedings of the 8th International Conference on Object-Oriented Information Systems, OOIS '02* (pp. 135–146). London: Springer. Available from http://dl.acm.org/citation.cfm?id=645789.667803&coll=DL&dl=GUIDE&CFID=467603704&CFTOKEN=88816314

Coplien, J. O. (n.d.). *Software Patterns.* The Hillside Group. Retrieved March 1, 2014, from http://hillside.net/component/content/article/50-patterns-library/patterns/222-design-pattern-definition

Culy, C., & Lyding, V. (2010). Double Tree: An Advanced KWIC Visualization for Expert Users. In E. Banissi et al. (Eds.), *14th International Conference Information Visualisation* (pp. 98–103). Los Alamitos, CA: IEEE. Available from http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5571339&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5571339

Dandapat, S., Biswas, P., Choudhury, M., & Bali, K. (2009). Complex Linguistic Annotation – No Easy Way Out! A Case from Bangla and Hindi POS Labeling Tasks. In *Proceedings of the 3rd Linguistic Annotation Workshop, LAW '09* (pp. 10–18). Stroudsburg, PA: Association for Computational Linguistics. Available from http://wmmks.csie.ncku.edu.tw/ACL-IJCNLP-2009/LAW-III/LAW-III-2009.pdf

DeLano, D. E. (1998). Patterns Mining. In L. Rising (Ed.), *The Patterns Handbook: Techniques, Strategies, and Applications* (pp. 87–95). Cambridge, UK: Press Syndicate of the University of Cambridge.

DeRose, S., Daniel, R. J., Grosso, P., Maler, E., Marsh, J., & Walsh, N. (2002). *XML Pointer Language (XPointer), W3C Working Draft.* World Wide Web Consortium (W3C). Retrieved May 31, 2014, from http://www.w3.org/TR/xptr/

DeRose, S., Maler, E., & Orchard, D. (2001). *XML Linking Language (XLink), Version 1.0, W3C Recommendation.* World Wide Web Consortium (W3C). Retrieved May 31, 2014, from http://www.w3.org/TR/xlink/

Desurvire, H., Kondziela, J., & Atwood, M. E. (1992). What is Gained and Lost when Using Methods Other Than Empirical Testing. In *Posters and short talks of the 1992 sigchi conference on human factors in computing systems* (pp. 125–126). New York, USA: ACM. Available from http://doi.acm.org/10.1145/1125021.1125115

Desurvire, H., Lawrence, D., & Atwood, M. (1991). Empiricism versus judgement: Comparing user interface evaluation methods on a new telephone-based interface. *ACM SIGCHI Bulletin*, *23*(4), 58–59. Available from http://dl.acm.org/citation.cfm?id=1056062

Dipper, S., Götze, M., & Stede, M. (2004). Simple annotation tools for complex annotation tasks: an evaluation. In M. T. Lino, M. F. Xavier, F. Ferreira, R. Costa, & R. Silva (Eds.), *Proceedings of 4th International Conference on Language Resources and Evaluation, LREC '04* (pp. 54–62). Lisbon, Portugal: Universidade Nova de Lisboa. Available from http://www.linguistics.ruhr-uni-bochum.de/~dipper/papers/xbrac04-sfb.pdf

Dobreva, M., O'Dwyer, A., & Feliciati, P. (2012). *User Studies for Digital Library Development*. London: Facet Publishing.

Dumas, J. S., Molich, R., & Jeffries, R. (2004). Describing usability problems: Are we sending the right message? *Interactions*, *11*(4), 24–29. Available from http://dl.acm.org/citation.cfm?id=1005274

Dybkjaer, L., Berman, S., Bernsen, N., Carletta, J., Heid, U., & Llisterri, J. (2001). *Requirements Specification for a Tool in Support of Annotation of Natural Interaction and Multimodal Data (Deliverable D11.2).* ISLE Natural Interactivity and Multimodality Working Group. Retrieved June 16, 2014, from http://spokendialogue.dk/Publications/2001e/D11.2-ISLE-29.7.2001-F.pdf

Dybkjaer, L., Berman, S., Kipp, M., Olsen, M., Pirrelli, V., Reithinger, N., et al. (2001). *Survey of existing tools, standards and user needs for annotation of natural interaction and multimodal data (Deliverable D11.1).* ISLE Natural Interactivity and Multimodality Working Group. Retrieved June 16, 2014, from http://spokendialogue.dk/Publications/2001f/D11.1-14.2.2001-F.pdf

Dybkjaer, L., & Bernsen, N. (2002). Data, annotation schemes and coding tools for natural interactivity. In J. H. L. Hansen & L. Pellom, Bryan (Eds.), *Proceedings of the INTERSPEECH, 7th International Conference for Spoken Language Processing, ICSLP '02* (pp. 217–220). Denver, CO: ISCA. Available from http://www.spokendialogue.dk/Publications/2002g/1247-9.6.2002-F.pdf

E-Teaching.org Redaktion. (2011). *Muster finden (Pattern Mining).* E-Teaching.org - Institut für Wissensmedien. Retrieved May 31, 2014, from http://www.e-teaching.org/didaktik/konzeption/entwurfsmuster/mining/

EAGLES Evaluation Working Group. (1999). *Evaluation of Natural Language Processing Systems. Pre-final draft version.* Expert Advisory Group on Language Engineering Standards (EAGLES), EAGLES Document EAG-II-EWG-PR.1. Retrieved May 31, 2014, from http://www.issco.unige.ch/en/research/projects/eagles/index.html

Elo, S., & Kyngäs, H. (2008). The qualitative content analysis process. *Journal of Advanced Nursing*, *62*(1), 107–115. Available from http://www.ncbi.nlm.nih.gov/pubmed/18352969

Engerer, V. (2012). Informationswissenschaft und Linguistik: Kurze Geschichte eines fruchtbaren interdisziplinären Verhältnisses in drei Akten. *International Journal for Language Data Processing*, *36*(2), 71–91.

Eryigit, G. (2007). ITU Treebank Annotation Tool. In *Proceedings of the 1st Lin-*

guistic Annotation Workshop, LAW '07 (pp. 117–120). Stroudsburg, PA: Association for Computational Linguistics. Available from http://dl.acm.org/citation.cfm?id=1642078

Ferreira, S. M., & Pithan, D. N. (2005). Usability of digital libraries: A study based on the areas of information science and human-computer-interaction. *OCLC Systems & Services*, *21*(4), 311–323. Available from http://www.emeraldinsight.com/journals.htm?articleid=1529765

Fincher, S. (2004, July). *PLML: Pattern Language Markup Language.* Retrieved August 23, 2013, from http://www.cs.kent.ac.uk/people/staff/saf/patterns/plml.html

Flynn, P. (2006). *If XML is so easy , how come it's so hard? The usability of editing software for structured documents.* Article presented at Extreme Markup Conference 2006. Retrieved May 31, 2014, from http://research.ucc.ie/articles/extreme06

Flynn, P. (2009). *Why writers don't use XML: The usability of editing software for structured documents.* Online Proceedings of Balisage: The Markup Conference 2009. Retrieved May 31, 2014, from http://www.balisage.net/Proceedings/vol3/html/Flynn01/BalisageVol3-Flynn01.html

Fogli, D., Fresta, G., & Mussio, P. (2004). On Electronic Annotation and its Implementation. In *Proceedings of the working conference on Advanced visual interfaces, AVI '04* (pp. 98–102). New York: ACM. Available from http://dl.acm.org/citation.cfm?id=989877&dl=ACM&coll=DL&CFID=356786406&CFTOKEN=97781857

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code.* Amsterdam: Addison Wesley.

Frischer, B., Unsworth, J., Dwyer, A., Jones, A., Lancaster, L., Rockwell, G., et al. (2005). *Summit on Digital Tools for the Humanities - Final Report.* September 28-30, 2005, at the University of Virginia. Retrieved June 16, 2014, from http://www.iath.virginia.edu/dtsummit/SummitText.pdf

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns. Elements of Reusable Object-Oriented Software.* Boston et al.: Addison-Wesley.

Garg, S., Martinovski, B., Robinson, S., Stephan, J., Tetreault, J., & Traum, D. R. (2004). Evaluation of Transcription and Annotation tools for a Multi-modal, Multi-party dialogue corpus. In M. T. Lino, M. F. Xavier, F. Ferreira, R. Costa, & R. Silva (Eds.), *Proceedings of 4th International Conference on Language Resources and Evaluation, LREC '04* (pp. 2163–2166). Lisbon, Portugal: Universidade Nova de Lisboa. Available from http://www.lrec-conf.org/proceedings/lrec2004/pdf/758.pdf

Garrido, A., Rossi, G., & Distante, D. (2011). Refactoring for Usability in Web Applications. *IEEE Software*, *28*(3), 60–67. Available from http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5518753&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5518753

Garside, R., & Smith, N. (1997). A Hybrid Grammatical Tagger: CLAWS4. In R. Garside, G. Leech, & A. McEnery (Eds.), *Corpus Annotation. Linguistic Information from Computer Text Corpora* (pp. 102–122). Harlow, Essex: Addison Wesley Longman.

Gediga, G., Hamborg, K. C., & Düntsch, I. (2002). *Evaluation of Software Systems.* Article published in the "Encyclopedia of Computer Science and Technology", preprint available at CiteSeer. Retrieved May 31, 2014, from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.87.8362

Gibbs, F., & Owens, T. (2012). Building Better Digital Humanities Tools: Toward broader audiences and user-centered designs. *Digital Humanities Quarterly (Online Journal)*, *6*(2). Available from http://www.digitalhumanities.org/dhq/vol/6/2/000136/000136.html

Goodwin, N. C. (1987). Functionality and usability. *Communications of the ACM*, *30*(3), 229–233. Available from http://portal.acm.org/citation.cfm?id=214758

Gries, S. T. (2009). *Quantitative corpus linguistics with R: A practical introduction*. New York: Routledge.

Gugerli, D. (2009). *Suchmaschinen: Die Welt als Datenbank*. Frankfurt am Main: Suhrkamp.

Hammwöhner, R. (2007). Interlingual Aspects Of Wikipedia's Quality. In M. A. Robbert, R. O'Hare, M. L. Markus, & B. D. Klein (Eds.), *Proceedings of the 12th International Conference on Information Quality, ICIQ '07* (pp. 477–488). Cambridge, MA: MIT. Available from http://epub.uni-regensburg.de/15572/

Harabagiu, S. M., Bunescu, R. C., & Trausan-Matu, S. (2001). COREFDRAW: A tool for Annotation and Visualization of Coreference Data. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence, ICTAI '01* (pp. 273 – 279). Los Alamitos, CA: IEEE. Available from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=974474

Harris, R. A. (1993). *The Linguistics Wars*. Oxford et al.: Oxford University Press.

Harrison, N. B. (1999). *The Language of Shepherding - A Pattern Language for Shepherds and Sheep.* The Hillside Group. Retrieved June 18, 2014, from http://hillside.net/index.php/the-language-of-shepherding

Hassenzahl, M., Beu, A., & Burmester, M. (2001). Engineering joy. *IEEE Software*, *2001*(January/February), 2–8. Available from http://www

.researchgate.net/publication/3247685_Engineering_joy/file/5046351bf48af7a9fe.pdf

Hastreiter, I., Burghardt, M., Elsweiler, D., & Wolff, C. (2013). Digitale Annotation im akademischen Kontext. Empirische Studie zur toolbasierten Annotationspraxis im Wissenschaftsbetrieb. In H.-C. Hobohm (Ed.), *Proceedings of the 13th International Symposium of Information Science, ISI '13* (pp. 118 – 129). Glückstadt: Verlag Werner Hülsbusch (VWH). Available from http://opus4.kobv.de/opus4-fhpotsdam/frontdoor/index/index/docId/402

Hertzum, M., & Jacobsen, N. E. (2003). The Evaluator Effect: A Chilling Fact About Usability Evaluation Methods. *International Journal of Human-Computer Interaction*, *15*(1), 183–204. Available from http://www.idemployee.id.tue.nl/g.w.m.rauterberg/lecturenotes/JFS-lecture/hertzum-jacobsen-2003.pdf

Hinze, A., Heese, R., Luczak-Rösch, M., & Paschke, A. (2012). Semantic Enrichment by Non-Experts: Usability of Manual Annotation Tools. In P. Cudré-Mauroux et al. (Eds.), *Proceedings of the 11th International Semantic Web Conference, ISWC '12* (pp. 165–181). Springer: Berlin, Heidelberg. Available from http://link.springer.com/chapter/10.1007%2F978-3-642-35176-1_11

Ide, N. (1998). Encoding linguistic corpora. In E. Charniak (Ed.), *Proceedings of the Sixth Workshop on Very Large Corpora, COLING-ACL '98* (pp. 9–17). Stroudsburg, PA: Association for Computational Linguistics. Available from http://www.cs.vassar.edu/~ide/papers/ces.wvlc.pdf

Ide, N. (2000). The XML framework and its implications for corpus access and use. In *Proceedings of Data Architectures and Software Support for Large Corpora* (pp. 28–32). Available from http://www.personal.psu.edu/xxl13/teaching/sp07/apling597e/resources/Ide_2000.pdf

Ide, N. (2007). *Annotation Science: From Theory to Practice and Use.* Invited talk at the Biennial GLDV-Conference 2007: Data Structures for Linguistic Resources and Applications. Retrieved June 2, 2014, from http://www.cs.vassar.edu/~ide/papers/GLDV.pdf

Ide, N., Baker, C., Fellbaum, C., & Fillmore, C. (2008). MASC: The Manually Annotated Sub-Corpus of American English Nancy Ide. In N. Calzolari et al. (Eds.), *Proceedings of the 6th Language Resources and Evaluation Conference, LREC '08.* Paris: ELRA. Available from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.180.8443

Ide, N., & Brew, C. (2000). Requirements, tools, and architectures for annotated corpora. In *Proceedings of Data Architectures and Software Support for Large Corpora* (pp. 1–5). Paris: ELRA. Available from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.7134&amp;rep=rep1&amp;type=pdf

Ide, N., & Fellbaum, C. (2010). The Manually Annotated Sub-Corpus: A Community Resource For and By the People. In *Proceedings of the ACL 2010 Conference Short Papers* (pp. 68–73). Stroudsburg, PA: Association for Computational Linguistics. Available from http://dl.acm.org/citation.cfm?id=1858855

Ide, N., & Romary, L. (2004). Representing linguistic corpora and their annotations. In *Proceedings of the 5th Language Resources and Evaluation Conference, LREC '06* (pp. 4–7). Paris: ELRA. Available from http://www.cs.vassar.edu/~ide/papers/LAF-LREC06.pdf

Ide, N., Romary, L., & Clergerie, E. de la. (2003). International Standard for a Linguistic Annotation Framework. In *Proceedings of HLT-NAACL'03 Workshop on The Software Engineering and Architecture of Language Technology, SEALTS '03* (pp. 25–30). Stroudsburg, PA: Association for Computational Linguistics. Available from http://dl.acm.org/citation.cfm?id=1119230

Ide, N., & Suderman, K. (2007). GrAF: A Graph-based Format for Linguistic Annotations. In *Proceedings of the 1st Linguistic Annotation Workshop, LAW '09* (pp. 1–8). Stroudsburg, PA: Association for Computational Linguistics. Available from http://www.cs.vassar.edu/~ide/papers/LAW.pdf

Ide, N., & Suderman, K. (2009). Bridging the Gaps: Interoperability for GrAF, GATE, and UIMA. In *Proceedings of the 3rd Linguistic Annotation Workshop, LAW '09* (pp. 27–34). Stroudsburg, PA: Association for Computational Linguistics. Available from http://www.aclweb.org/anthology/W/W09/W09-3004

Isard, A., McKelvie, D., Mengel, A., & Mø ller, M. (2000). The MATE workbench annotation tool, a technical description. In *Proceedings of the 2nd Language Resources and Evaluation Conference, LREC '00* (pp. 11–17). Available from http://www.anras.de/pubs/mate-lrec2000.pdf

ISO 24612. (2012). *Language resource management – Linguistic annotation framework (LAF).*

ISO 9241-11. (1999). *Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability.*

ISO 9241-210. (2010). *Ergonomics of human-system interaction – Part 210: Human-centred design process for interactive systems.*

ISO TC 37/SC 4. (2001). *Overview of standards from TC 37/SC 4 Language resource management.* ISO, Standards Development. Retrieved June 16, 2014, from http://www.iso.org/iso/home/standards_development/list_of_iso_technical_committees/iso_technical_committee.htm?commid=297592

Jeffries, R. (1994). Usability Problem Reports: Helping Evaluators Communicate Effectively with Developers. In J. Nielsen & R. Mack (Eds.), *Usability Inspection Methods* (pp. 273–294). New York et al.: Wiley & Sons.

Jeffries, R., Miller, J. R., Wharton, C., & Uyeda, K. M. (1991). User Interface Evaluation in the Real World: A Comparison of Four Techniques. In S. P. Robertson, G. M. Olson, & J. S. Olson (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '91* (pp. 119–124). New York: ACM. Available from `http://dl.acm.org/citation.cfm?id=108862`

Johnson, J. (2007). *GUI Bloopers 2.0: Common User Interface Design Don'ts and Dos* (2nd ed. ed.). Amsterdam et al.: Morgan Kaufmann.

Johnson, J. (2010). *Designing with the Mind in Mind. Simple Guide to Understanding User Interface Design Rules*. Amsterdam et al.: Morgan Kaufman.

Kaeding, F. W. (1898). *Häufigkeitswörterbuch der deutschen Sprache*. Steglitz bei Berlin: Selbstverlag.

Kaplan, D., Iida, R., Nishina, K., & Tokunaga, T. (2011). Slate – A Tool for Creating and Maintaining Annotated Corpora. *Journal for Language Technology and Computational Linguistics (JLCL)*, *26*(2), 91–103. Available from `http://www.jlcl.org/2011_Heft2/11.pdf`

Kaplan, D., Iida, R., & Tokunaga, T. (2010). Slat 2.0: Corpus construction and annotation process management. In *Proceedings of the 16th Annual Meeting of The Association for Natural Language Processing* (pp. 510–513). Available from `https://www.cl.cs.titech.ac.jp/publication/654.pdf`

Karat, C.-M. (1994). A Comparison of User Interface Evaluation Methods. In J. Nielsen & R. L. Mack (Eds.), *Usability Inspection Methods* (pp. 203–233). New York: Wiley & Sons.

Karat, C.-M., Campbell, R., & Fiegel, T. (1992). Comparison of empirical testing and walkthrough methods in user interface evaluation. In P. Bauersfeld, J. Bennett, & G. Lynch (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '92* (pp. 397–404). New York: ACM. Available from `http://dl.acm.org/citation.cfm?id=142873`

Koenig, A. (1998). Patterns and Antipatterns. In L. Rising (Ed.), *The Patterns Handbook: Techniques, Strategies, and Applications* (pp. 383–389). Cambridge, UK: Cambridge University Press.

Krippendorff, K. (2013). *Content Analysis: An Introduction to Its Methodology* (3rd ed. ed.). Los Angeles: Sage Publications.

Krischkowsky, A., Wurhofer, D., Perterer, N., & Tscheligi, M. (2013). Developing Patterns Step-by-Step A Pattern Generation Guidance for HCI Researchers. In A. Zimmermann (Ed.), *Proceedings of the 5th International Conferences on Pervasive Patterns and Applications, PATTERNS '13* (pp. 66–72). ThinkMind.

Kruschitz, C., & Hitz, M. (2009). The Anatomy of HCI Design Patterns. In P. Dini, W. Gentzsch, P. Geraci, P. Lorenz, & K. Singh (Eds.), *Proceedings of Computation World 2009: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns* (pp. 202–207). Los Alamitos, CA: IEEE. Available from `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5359582`

Kruschitz, C., & Hitz, M. (2010). Are human-computer interaction design patterns really used? In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction Extending Boundaries, NordiCHI '10* (pp. 711–714). New York: ACM. Available from `http://portal.acm.org/citation.cfm?doid=1868914.1869011`

Lazar, J., Feng, J. H., & Hochheiser, H. (2010). *Research Methods In Human-Computer Interaction*. Chichester: Wiley Publishing.

Leech, G. (1993). Corpus Annotation Schemes. *Literary and Linguistic Computing*, *8*(4), 275–281. Available from `http://llc.oupjournals.org/cgi/doi/10.1093/llc/8.4.275`

Leech, G. (1997). Introducing Corpus Annotation. In R. Garside, G. Leech, & A. McEnery (Eds.), *Corpus Annotation. Linguistic Information from Computer Text Corpora* (pp. 1–18). Harlow, Essex: Addison Wesley Longman.

Leech, G., Garside, R., & Bryant, M. (1994). CLAWS4: The Tagging of the British National Corpus. In *Proceedings of the 15th conference on computational linguistics, COLING '94* (pp. 622–628). Stroudsburg, PA: ACL. Available from `http://dl.acm.org/citation.cfm?id=991996`

Lenk, A., Klems, M., Nimis, J., Tai, S., & Sandholm, T. (2009). What' s Inside the Cloud? An Architectural Map of the Cloud Landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, CLOUD '09* (pp. 23–31). Washington, DC,: IEEE Computer Society. Available from `http://dl.acm.org/citation.cfm?id=1564625`

Lewis, C., Polson, P. G., Wharton, C., & Riemann, J. (1990). Testing a walk-through methodology for theory-based design of walk-up-and-use interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '90* (pp. 235–242). New York: ACM. Available from `http://dl.acm.org/citation.cfm?id=97279`

Lobin, H. (2004). Dokumentgrammatiken als Grundlagen von XML-Tools. In A. Mehler & H. Lobin (Eds.), *Automatische Textanalyse: Systeme und Methoden zur Annotation und Analyse natürlihsprachlicher Texte* (pp. 23–38). Wiesbaden: VS Verlag für Sozialwissenschaften.

Lohmann, S., Ziegler, J., & Tetzlaff, L. (2009). Comparison of tag cloud layouts: Task-related performance and visual exploration. In T. Gross et al. (Eds.), *Proceedings of the 12th IFIP TC 13 International Conference, Human-*

Computer Interaction, INTERACT '09 (pp. 392–404). Heidelberg, Berlin: Springer. Available from http://link.springer.com/chapter/10.1007/978-3-642-03655-2_43

Luhn, H. P. (1960). Keyword-in-Context Index for Technical Literature (KWIC). *American Documentation*, *11*(4), 288–295. Available from http://onlinelibrary.wiley.com/doi/10.1002/asi.5090110403/abstract

Mack, R., & Nielsen, J. (1993). Usability inspection methods: report on a workshop held at CHI'92. *SIGCHI Bulletin*, *25*(1), 28–33. Available from http://dl.acm.org/citation.cfm?id=157207

Mack, R., & Nielsen, J. (1994). Executive Summary. In J. Nielsen & R. Mack (Eds.), *Usability Inspection Methods* (pp. 1–23). New York: Wiley & Sons.

Mahatody, T., Sagar, M., & Kolski, C. (2010). State of the Art on the Cognitive Walkthrough Method, Its Variants and Evolutions. *International Journal of Human-Computer Interaction*, *26*(8), 741–785. Available from http://www.tandfonline.com/doi/abs/10.1080/10447311003781409?journalCode=hihc20#.U59kaS9fshc

Mahemoff, M. J., & Johnston, L. (1998). Pattern languages for usability: an investigation of alternative approaches. In J. Tanaka (Ed.), *Proceedings of the Asia-Pacific Conference on Human Computer Interaction, APCHI '98* (pp. 25–31). Los Alamitos, CA: IEEE. Available from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=704138

Marcus, M., Marcinkiewicz, M., & Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, *19*(2), 313–330. Available from http://dl.acm.org/citation.cfm?id=972475

Marshall, C. C. (1997). Annotation: from paper books to the digital library. In *Proceedings of the 2nd ACM international conference on Digital libraries* (pp. 131–140). New York: ACM. Available from http://portal.acm.org/citation.cfm?id=263806

Marshall, C. C. (1998). Toward an ecology of hypertext annotation. In *Proceedings of the 9th ACM conference on Hypertext and hypermedia : links, objects, time and space-structure in hypermedia systems links, objects, time and space-structure in hypermedia systems, HYPERTEXT '98* (pp. 40–49). New York: ACM. Available from http://dl.acm.org/citation.cfm?id=276632

Marshall, C. C. (2010). *Reading and Writing the Electronic Book* (Synthesis ed.; G. Marchionini, Ed.). San Rafael, CA: Morgan & Claypool Publishers.

Marshall, C. C., & Brush, B. A. (2004). Exploring the relationship between personal and public annotations. In *Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries, JCDL '04* (pp. 349–357). New York:

ACM. Available from http://portal.acm.org/citation.cfm?doid=996350.996432

Martin, D., Rodden, T., Rouncefield, M., Sommerville, I., & Viller, S. (2001). Finding patterns in the fieldwork. In W. Prinz, M. Jarke, Y. Rogers, K. Schmidt, & V. Wulf (Eds.), *Proceedings of the seventh conference on European Conference on Computer Supported Cooperative Work, ECSCW'01* (pp. 39–58). Norwell, MA: Kluwer Academic Publishing. Available from http://dl.acm.org/citation.cfm?id=1241870

McEnery, T., & Hardie, A. (2012). *Corpus Linguistics: Method, Theory and Practice.* Cambridge: Cambridge University Press.

McEnery, T., & Wilson, A. (1996). *Corpus Linguistics*. Edinburgh: Edinburgh University Press.

McLoughlin, T. (2008). Bridging the Gap. In M. Rehbein & S. Ryder (Eds.), *Jahrbuch für Computerphilologie* (pp. 37–54). Münster: mentis Verlag. Available from http://computerphilologie.tu-darmstadt.de/jg08/mclough.pdf

Mehler, A., & Lobin, H. (2004). Aspekte der texttechnologischen Modellierung. In A. Mehler & H. Lobin (Eds.), *Automatische Textanalyse: Systeme und Methoden zur Annotation und Analyse natürlihsprachlicher Texte* (pp. 1–21). Wiesbaden: VS Verlag für Sozialwissenschaften.

Merialdo, B. (1994). Tagging English text with a probabilistic model. *Computational Linguistics*, *20*(2), 155–171. Available from http://dl.acm.org/citation.cfm?id=972526

Meyer, C. F. (2002). *English Corpus Linguistics: An Introduction.* Cambridge: Cambridge University Press.

Michel, J.-B., Shen, Y. K., Aiden, A. P., Veres, A., Gray, M. K., Pickett, J. P., et al. (2011). Quantitative analysis of culture using millions of digitized books. *Science*, *331*(6014), 176–182. Available from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3279742&tool=pmcentrez&rendertype=abstract

Mintert, S. (1998). *Auszeichnungssprachen im World Wide Web.* Published at mintert.com. Retrieved June 16, 2014, from http://www.mintert.com/xml/mlweb/

Molich, R., Jeffries, R., & Dumas, J. (2007). Making usability recommendations useful and usable. *Journal of Usability Studies*, *2*(4), 162–179. Available from http://www.usabilityprofessionals.org/upa_publications/jus/2007august/useful-usable.pdf

Molich, R., & Nielsen, J. (1990). Improving a Human-Computer Dialogue.

Communications of the ACM, *33*(3), 338–348. Available from http://dl.acm.org/citation.cfm?id=77486

Montgomery, C. A. (1972). Linguistics and Information Science. *Journal of the American Society for Information Science*, *23*(3), 195–219. Available from http://onlinelibrary.wiley.com/doi/10.1002/asi.4630230309/abstract

Moretti, F. (2007). *Graphs, Maps, Trees: Abstract Models for Literary History*. London: Verso.

Moretti, F. (2013). *Distant Reading*. London: Verso. Available from http://www.amazon.com/Distant-Reading-Franco-Moretti/dp/1781680841

Nagao, K. (2003). *Digital content annotation and transcoding.* Norwood: Artech House.

Nakamura, M., & Shikano, K. (1989). A study of English word category prediction based on neutral networks. In *Prodeedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP '89* (pp. 731–734). Los Alamitos, CA: IEEE. Available from http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=266531&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D266531

Nielsen, J. (1992). Finding usability problems through heuristic evaluation. In P. Bauersfeld, J. Bennett, & G. Lynch (Eds.), *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '92* (pp. 373–380). New York: ACM. Available from http://portal.acm.org/citation.cfm?doid=142750.142834

Nielsen, J. (1993). *Usability Engineering*. Amsterdam et al.: Morgan Kaufman.

Nielsen, J. (1994a). Enhancing the explanatory power of usability heuristics. In C. Plaisant (Ed.), *Conference Companion on Human Factors in Computing Systems, CHI '94* (pp. 152–158). New York: ACM. Available from http://portal.acm.org/citation.cfm?doid=259963.260333

Nielsen, J. (1994b, January). *Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier.* Alertbox. Retrieved August 26, 2013, from http://www.nngroup.com/articles/guerrilla-hci/

Nielsen, J. (1994c). Heuristic evaluation. In J. Nielsen & R. Mack (Eds.), *Usability Inspection Methods* (pp. 25–62). New York: Wiley & Sons.

Nielsen, J. (1994d). Usability Inspection Methods. In C. Plaisant (Ed.), *Conference Companion on Human Factors in Computing Systems, CHI '94* (pp. 413–414). New York: ACM. Available from http://dl.acm.org/citation.cfm?id=260531

Nielsen, J., & Mack, R. (Eds.). (1994). *Usability Inspection Methods*. New York: John Wiley & Sons.

Nielsen, J., & Molich, R. (1990). Heuristic Evaluation of User Interfaces. In J. Carrasco Chew & J. Whiteside (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '90* (pp. 249–256). New York: ACM. Available from `http://dl.acm.org/citation.cfm?id=97281`

NISO Press. (2004). *Understanding Metadata.* National Information Standards Organization (NISO). Retrieved June 16, 2014, from `http://www.niso.org/publications/press/UnderstandingMetadata.pdf`

Novick, D. G., & Hollingsed, T. (2007). *Usability inspection methods after 15 years of research and practice*. Unpublished doctoral dissertation, Departmental paper at the University of Texas, Department of Computer Science, El Paso. Available from `http://digitalcommons.utep.edu/cs_papers/16`

Obrist, M., Wurhofer, D., Beck, E., & Tscheligi, M. (2010). CUX patterns approach: Towards contextual user experience patterns. In *Proceedings of the 2nd International Conferences on Pervasive Patterns and Applications, PATTERNS '10* (pp. 60–65). ThinkMind. Available from `http://www.thinkmind.org/index.php?view=article&articleid=patterns_2010_3_20_70079`

Ovsiannikov, I. (2002). *Annotation Databases for Distributed Documents*. Doctoral dissertation, University of South California. Available from `http://www.ovsy.com/resume`

Ovsiannikov, I., Arbib, M., & McNeill, T. (1999). Annotation technology. *International Journal of Human-Computer Studies*, *50*(4), 329–362. Available from `http://www.sciencedirect.com/science/article/pii/S1071581999902471`

Palmer, M., & Xue, N. (2010). Linguistic Annotation. In A. Clark, C. Fox, & S. Lappin (Eds.), *Computational Linguistics and Natural Language Processing Handbook* (pp. 238–270). Oxford et al.: Wiley-Blackwell.

Paulsson, F., & Engman, J. (2007). Treating metadata as annotations: separating the content markup from the content. *International Journal of Emerging Technologies in Learning*, *2*(1), 1–12. Available from `http://en.scientificcommons.org/37191132`

Pemberton, L. (2000). *The promise of pattern languages for interaction design.* Presentation at the Human Factors Symposium 2000, Loughborough. Retrieved June 16, 2014, from `http://jabba.edb.utexas.edu/it/ArticleSummary/PaulSummary.pdf`

Pilz, T., Ernst-Gerlach, A., Kempken, S., Rayson, P., & Archer, D. (2008). The Identification of Spelling Variants in English and German Historical Texts: Manual or Automatic? *Literary and Linguistic Computing*, *23*(1), 65–72. Available from http://llc.oxfordjournals.org/cgi/doi/10.1093/llc/fqm044

Polson, P., Lewis, C., Rieman, J., & Wharton, C. (1992). Cognitive walkthroughs: A method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies, 36*(5), 741–773. Available from http://www.sciencedirect.com/science/article/pii/002073739290039N

Pontin, J. (2007). *Artificial Intelligence, With Help From the Humans.* The New York Times Online. Retrieved July 4, 2013, from http://www.nytimes.com/2007/03/25/business/yourmoney/25Stream.html?_r=0

Popescu-Belis, A. (2010). Managing Multimodal Data, Metadata and Annotations: Challenges and Solutions. In J.-P. Thiran, F. Marqués, & H. Bourlard (Eds.), *Multimodal Signal Processing for Human-Computer Interaction* (pp. 187–203). Oxford, UK: Academic Press.

Poritz, A. (1988). Hidden Markov models: a guided tour. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP '88* (pp. 7–13). Los Alamitos, CA: IEEE. Available from http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=196495&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D196495

Ratzka, A. (2008). Steps in Identifying Interaction Design Patterns for Multimodal Systems. In P. Fobrig & F. Paternò (Eds.), *Proceedings of the 2nd Conference on Human-Centered Software Engineering, HCSE '08, and 7th International Workshop on Task Models and Diagrams, TAMODIA '08* (pp. 58–71). Berlin, Heidelberg: Springer. Available from http://link.springer.com/chapter/10.1007/978-3-540-85992-5_5

Reidsma, D., Hofs, D., & Jovanovic, N. (2005). *Designing focused and efficient annotation tools.* University of Twente, EEMCS EPrints Service. Retrieved June 16, 2014, from http://doc.utwente.nl/65561/

Reidsma, D., Jovanovic, N., & Hofs, D. (2004). *Designing Annotation Tools Based in Properties of Annotation Problems.* University of Twente, UTpublications. Retrieved June 16, 2014, from http://doc.utwente.nl/49282/

Ringger, E., McClanahan, P., Haertel, R., Busby, G., Carmen, M., Caroll, J., et al. (2007). Active learning for part-of-speech tagging: Accelerating corpus annotation. In *Proceedings of the 1st Linguistic Annotation Workshop, LAW'07* (pp. 101–108). Stroudsburg, PA: ACL. Available from http://dl.acm.org/citation.cfm?id=1642075

Rosson, M. B., & Carroll, J. M. (2002). *Usability Engineering - Scenario-Based Development of Human-Computer Interaction.* San Franscisco et al.: Morgan Kaufmann.

Rowe, D. (2011). *How and why we annotate: An annotation tax-*

onomy. Master's thesis, Dalhousie University, Halifax, Nova Scotia. Available from https://www.cs.dal.ca/sites/default/files/technical_reports/CS-2011-03.pdf

Ruecker, S., Radzikowska, M., & Sinclair, S. (2011). *Visual Interface Design for Digital Cultural Heritage*. Farnham et al.: Ashgate Publishing.

Ruvane, M. B. (2006). Annotation as process: A vital information seeking activity in historical geographic research. *Proceedings of the American Society for Information Science and Technology*, *42*(1). Available from http://doi.wiley.com/10.1002/meet.14504201178

Ruvane, M. B. (2007). Defining annotations: a visual (re)interpretation. *Proceedings of the American Society for Information Science and Technology*, *43*(1), 1–5. Available from http://onlinelibrary.wiley.com/doi/10.1002/meet.14504301226/abstract

Santos, D., & Frankenberg-Garcia, A. (2007). The corpus, its users and their needs: a user-oriented evaluation of COMPARA. *International Journal of Corpus Linguistics*, *12*(3), 335–374. Available from http://www.ingentaconnect.com/content/jbp/ijcl/2007/00000012/00000003/art00002

Schilit, B. N., Golovchinsky, G., & Price, M. N. (1998). Beyond paper: supporting active reading with free form digital ink annotations. In M. E. Atwood, C.-M. Karat, A. Lund, J. Coutaz, & J. Karat (Eds.), *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '98* (pp. 249–256). New York: ACM. Available from http://portal.acm.org/citation.cfm?id=274680

Schmid, H. (1994a). Part-of-speech tagging with neural networks. In *Proceedings of the 15th conference on computational linguistics, COLING '94* (pp. 172–176). Stroudsburg, PA: ACL. Available from http://dl.acm.org/citation.cfm?doid=991886.991915

Schmid, H. (1994b). Probabilistic Part-of-Speech Tagging Using Decision Trees. In D. Jones (Ed.), *Proceedings of International Conference on New Methods in Language Processing* (pp. 44–49). CiteSeer. Available from http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger1.pdf

Schreibman, S., Siemens, R., & Unsworth, J. (2004). The Digital Humanities and Humanities Computing: An Introduction. In S. Schreibman, R. Siemens, & J. Unsworth (Eds.), *A companion to digital humanities.* Oxford: Blackwell. Available from http://nora.lis.uiuc.edu:3030/companion/view?docId=blackwell/9781405103213/9781405103213.xml&chunk.id=ss1-1-3&toc.depth=1&toc.id=ss1-1-3&brand=9781405103213_brand

Sears, A. (1997). Heuristic Walkthroughs: Finding the Problems Without the Noise. *International Journal of Human-Computer Interaction*, *9*(3), 213–234. Available from http://www.tandfonline.com/doi/abs/10.1207/s15327590ijhc0903_2#.U59qhi9fshc

Settles, B. (2010). *Active learning literature survey.* Computer Sciences Technical Report 1648, University of Wisconsin, Madison. Retrieved June 16, 2014, from http://apophenia.wdfiles.com/local--files/start/settles_active_learning.pdf

Shillingsburg, P. L. (2006). *From Gutenberg to Google: Electronic Representations of Literary Texts.* Cambridge, UK: Cambridge University Press.

Shneiderman, B., & Plaisant, C. (2009). *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (5th revise ed.). Upper Saddle River, NJ, et al.: Addison Wesley.

Soehn, J.-P., Zinsmeister, H., & Rehm, G. (2008). Requirements of a user-friendly, general-purpose corpus query interface. In *Proceedings of the LREC 2008, Workshop on Sustainability of Language Resources and Tools for Natural Language Processing* (pp. 27–32). ELRA. Available from http://ling.uni-konstanz.de/pages/home/zinsmeister/publ/SoehnRehmZinsmeister2008.pdf

Song, H., & Yao, T. (2010). Active Learning Based Corpus Annotation. In *IPS-SIGHAN Joint Conference on Chinese Language Processing* (pp. 28–29). Available from http://www.aclweb.org/anthology/W/W10/W10-4121.pdf

Stemler, S. (2001). An Overview of Content Analysis. *Practical Assessment, Research & Evaluation*, *17*(7). Available from http://pareonline.net/getvn.asp?v=7&n=17

Stone, S. (1982). Humanities Scholars: Information Needs and Uses. *Journal of Documentation*, *38*(4), 292–313. Available from http://www.emeraldinsight.com/journals.htm?articleid=1649976

TEI. (n.d.). *TEI: Goals and Missions.* Text Encoding Inititative. Retrieved March 4, 2013, from http://www.tei-c.org/About/mission.xml

TEI. (2014a). *About These Guidelines (TEI P5).* Text Encoding Inititative. Retrieved June 16, 2014, from http://www.tei-c.org/release/doc/tei-p5-doc/en/html/AB.html

TEI. (2014b). *P5: Guidelines for Electronic Text Encoding and Interchange.* Text Encoding Inititative. Retrieved June 10, 2014, from http://www.tei-c.org/release/doc/tei-p5-doc/en/html/index.html

Thompson, C., Califf, M., & Mooney, R. (1999). Active learning for natural language parsing and information extraction. In *Proceedings of the international conference on machine learning, ICML '99* (pp. 406–414). Available from http://dl.acm.org/citation.cfm?id=657614

Thompson, H. S., & McKelvie, D. (1997). *Hyperlink semantics for standoff markup of read-only documents.* Language Technology Group HCRC, University of Edinburgh. Barcelona. Available from `http://www.ltg.ed.ac.uk/~ht/sgmleu97.html`

Tidwell, J. (2005). *Designing Interfaces* (1st ed.). Sebastopol, CA: O'Reilly Media.

Tidwell, J. (2011). *Designing Interfaces* (2nd ed.). Sebastopol, CA: O'Reilly Media.

Tognini-Bonelli, E. (2001). *Corpus Linguistics at Work*. Amsterdam: Benjamins.

Tomanek, K., Hahn, U., & Lohmann, S. (2010). A cognitive cost model of annotations based on eye-tracking data. In *Proceedings of the 48th Annual Meeting of the ACL* (pp. 1158–1167). Stroudsburg, PA: ACL. Available from `http://dl.acm.org/citation.cfm?id=1858799`

Unicode Consortium. (2013). *What is Unicode?* Unicode - General Information. Retrieved June 10, 2014, from `http://www.unicode.org/standard/WhatIsUnicode.html`

Unsworth, J. (2000). *Scholarly Primitives: what methods do humanities researchers have in common, and how might our tools reflect this?* Delivered as part of a symposium on "Humanities Computing: formal methods, experimental practice". Retrieved June 10, 2014, from `http://people.brandeis.edu/~unsworth/Kings.5-00/primitives.html`

Unsworth, J. (2003). *Tool-Time, or 'Haven't We Been Here Already?' Ten Years in Humanities Computing.* Delivered as part of "Transforming Disciplines: The Humanities and Computer Science". Retrieved June 10, 2014, from `http://people.brandeis.edu/~unsworth/carnegie-ninch.03.html`

Unsworth, J. (2004). *What is humanities computing (and what is not)?* Delivered at Texas A & M, as part of the Humanities Informatics lecture series, College Station, TX, September 10, 2004. Retrieved June 10, 2014, from `http://people.brandeis.edu/~unsworth/texas-hc.html`

Van Biljon, J., Kotzé, P., Renaud, K., McGee, M., & Seffah, A. (2004). The Use of Anti-Patterns in Human Computer Interaction: Wise or Ill-Advised? In G. Marsden, P. Kotzé, & A. Adesina-Ojo (Eds.), *Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, SAICSIT '04* (pp. 176–185). South African Institute for Computer Scientists and Information Technologists. Available from `http://dl.acm.org/citation.cfm?id=1035053&picked=prox&CFID=472310716&CFTOKEN=33804360`

Van Duyne, D. K., Landay, J., & Hong, J. (2006). *The Design of Sites: Patterns for Creating Winning Websites* (2nd ed.). Upper Saddle River et al.: Prentice-Hall.

Van Welie, M. (2001). *Task-Based User Interface Design*. Doctoral dissertation, siks dissertation series no. 2001-6, Vrije Universiteit Amsterdam. Available from http://www.welie.com/papers/Welie-PhD-thesis.pdf

Van Welie, M., & Trætteberg, H. (2000). *Interaction patterns in user interfaces.* Proceedings of the 7th Pattern Languages of Programs Conference, PLoP '00. Retrieved June 12, 2013, from http://hillside.net/plop/plop/plop2k/proceedings/Welie/Welie.pdf

Van Welie, M., Van Der Veer, G., & Eliëns, A. (1999). *Breaking down usability.* Proceedings of the 7th International Conference on Human-Computer Interaction, INTERACT '99. Retrieved September 4, 2013, from http://www.cs.vu.nl/~gerrit/gta/docs/Interact99.pdf

Van Welie, M., & Van der Veer, G. C. (2000). *A structure for usability based patterns.* CHI 2000 Workshop on Pattern Languages for Interaction Design: Building Momentum. Retrieved October 18, 2013, from http://www.it.bton.ac.uk/staff/rng/CHI2K_Plworkshop/PositionPapers/Welie.PDF

Van Welie, M., & Van der Veer, G. C. (2003). *Pattern languages in interaction design: Structure and organization.* Proceedings of INTERACT '03. Retrieved June 12, 2013, from http://www.idemployee.id.tue.nl/g.w.m.rauterberg/conferences/interact2003/INTERACT2003-p527.pdf

Van Welie, M., Van der Veer, G. C., & Eliëns, A. (2001). Patterns as tools for user interface design. In J. Vanderdonckt & C. Farenc (Eds.), *Tools for Working with Guidelines* (pp. 313–324). London: Springer. Available from http://link.springer.com/chapter/10.1007/978-1-4471-0279-3_30

Vanhoutte, E. (2011). *So You Think You Can Edit? The Masterchef Edition.* The Mind Tool: Edward Vanhoutte's Blog. Retrieved July 9, 2013, from http://edwardvanhoutte.blogspot.de/2011/10/so-you-think-you-can-edit-masterchef.html

Warfel, T. Z. (2009). *Prototyping: A Practitioner's Guide.* New York: Rosenfeld Media.

Warwick, C. (2012). Studying users in digital humanities. In C. Warwick, M. Terras, & J. Nyhan (Eds.), *Digital Humanities in Practice* (pp. 1–21). London: Facet Publishing.

Warwick, C., Terras, M., Huntington, P., & Pappa, N. (2008). If You Build It Will They Come? The LAIRAH Study: Quantifying the Use of Online Resources in the Arts and Humanities through Statistical Analysis of User Log Data. *Literary and Linguistic Computing*, *23*(1), 85–102. Available from http://llc.oxfordjournals.org/cgi/doi/10.1093/llc/fqm045

Wattenberg, M., & Viégas, F. B. (2008). The Word Tree, an Interactive Visual Concordance. *IEEE Transactions on visualization and computer graphics*, *14*(6), 1221–1228. Available from http://ieeexplore.ieee.org/xpl/login

.jsp?tp=&arnumber=4658133&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D4658133

Wellhausen, T., & Fiesser, A. (2012). How to write a pattern? A rough guide for first-time pattern authors. In *Proceedings of the 16th European Conference on Pattern Languages of Programs, EuroPLoP '11 (Article No. 5).* New York: ACM. Available from http://dl.acm.org/citation.cfm?id=2396721

Wharton, C., Riemann, J., Lewis, C., & Polson, P. (1994). The cognitive walkthrough method: a practitioner's guide. In J. Nielsen & R. Mack (Eds.), *Usability Inspection Methods* (pp. 105–140). New York: Wiley & Sons. Available from http://dl.acm.org/citation.cfm?id=189200.189214

Wilcock, G. (2009). *Introduction to Linguistic Annotation and Text Analytics*. San Rafael, CA: Morgan & Claypool Publishers.

Wilhelm, T., Burghardt, M., & Wolff, C. (2013). "To See or Not to See" - An Interactive Tool for the Visualization and Analysis of Shakespeare Plays. In R. Franken-Wendelstorf, E. Lindinger, & J. Sieck (Eds.), *Kultur und Informatik: Visual Worlds & Interactive Spaces* (pp. 175–185). Glückstadt: Verlag Werner Hülsbusch. Available from http://epub.uni-regensburg.de/28417/

Witte, R., & Tang, T. (2007). *Task-Dependent Visualization of Coreference Resolution Results.* International Conference on Recent Advances in Natural Language Processing, RANLP '07. Retrieved February 25, 2014, from http://www.rene-witte.net/system/files/075_witte.pdf

Wolfe, J. L., & Neuwirth, C. M. (2001, July). From the Margins to the Center: The Future of Annotation. *Journal of Business and Technical Communication*, *15*(3), 333–371. Available from http://jbt.sagepub.com/cgi/doi/10.1177/105065190101500304

Womser-Hacker, C. (2010). Was ist Informationswissenschaft? Die Hildesheimer Antwort auf aktuelle Herausforderungen der globalisierten Informationsgesellschaft. *Information - Wissenschaft und Praxis*, *61*(6-7), 335–340. Available from http://www.b-i-t-online.de/pdf/iwp/IWP2010-6-7.pdf

Zloof, M. M. (1977). Query-by-Example: A data base language. *IBM Systems Journal*, *16*(4), 324–343. Available from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5388055

Appendices

A. Basic steps of the evaluation procedure

This passage contains the basic steps of the evaluation procedure (cf. chapter 4.5.6) that was applied for each of the tested tools. The steps were all printed out and available during the evaluation, to guarantee a consistent evaluation of all annotation tools.

1. Obtain tool: download the tool files

2. First contact / dry practice: scan through the manual / documentation and existing publications to get a first overview of what the tool is like

3. Task-oriented evaluation: achieve Tasks 1-6

    - Screen recording of each task (saved as AVI files)
    - Write down the basic steps necessary to achieve the task as well as a final roundup of the tool (cf. appendix C)
    - Always keep in mind Sear's four thought-focusing questions, and loosely write down usability problems, specific strengths, and other interesting characteristics of the tool
    - Make screenshots of interesting interface aspects

4. Free-form evaluation: document strengths and weaknesses in a spreadsheet

    - Fill in the meta information for the tool (*name*, *source of download*, *date of download*, *date of evaluation*)
    - Keep in mind Nielsen's usability heuristics to systematically identify usability problems and document them with the following information: *id*, *short title*, *description*, *related task*, *violated heuristic*, *severity ranking*, *category*
    - Identify positive characteristic of the tool and document them with the same information as the problems (note: leave out severity ranking for obvious reasons)

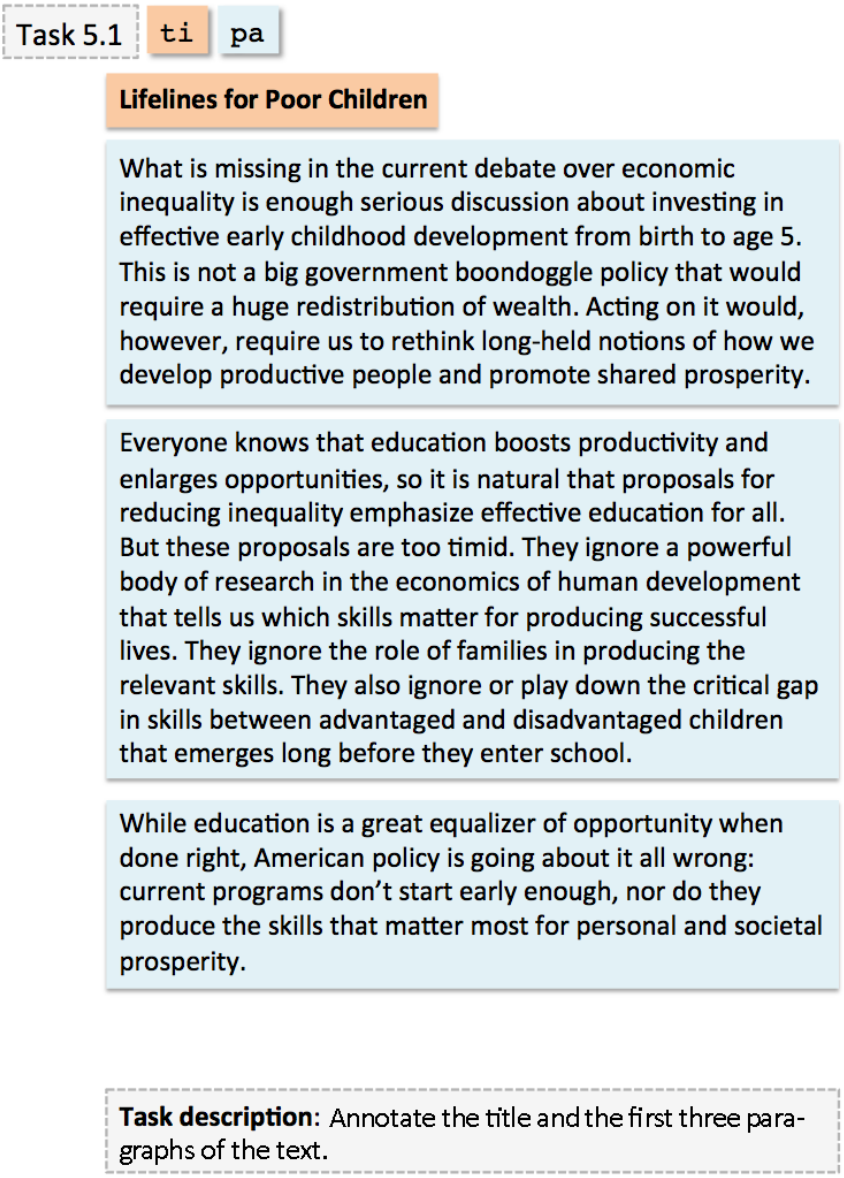
5. Calculate the number of usability problems and the number of violated heuristics and add the tool to the overview spreadsheet

6. Archive documents and files used and created during evaluation

- Screen recordings
- Screenshots
- Installation files
- Test data
- Manual / documentation
- Publications
- Spreadsheets

7. Fill in the feature matrix (comparison of all evaluated tools according to a set of attributes)

B. Solutions for annotation tasks



Task 5.1 ti pa

Lifelines for Poor Children

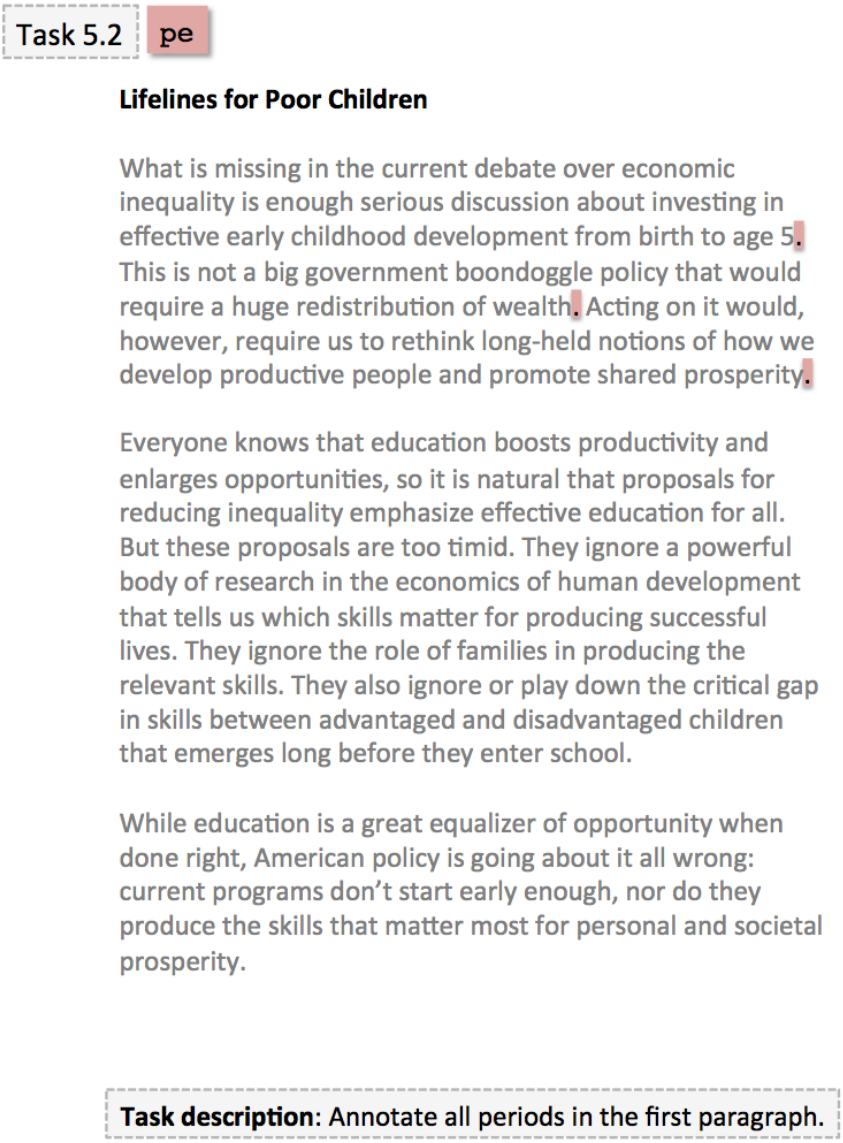
What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

Task description: Annotate the title and the first three paragraphs of the text.

Figure B.1.: Solution for Task 5.1.



Lifelines for Poor Children

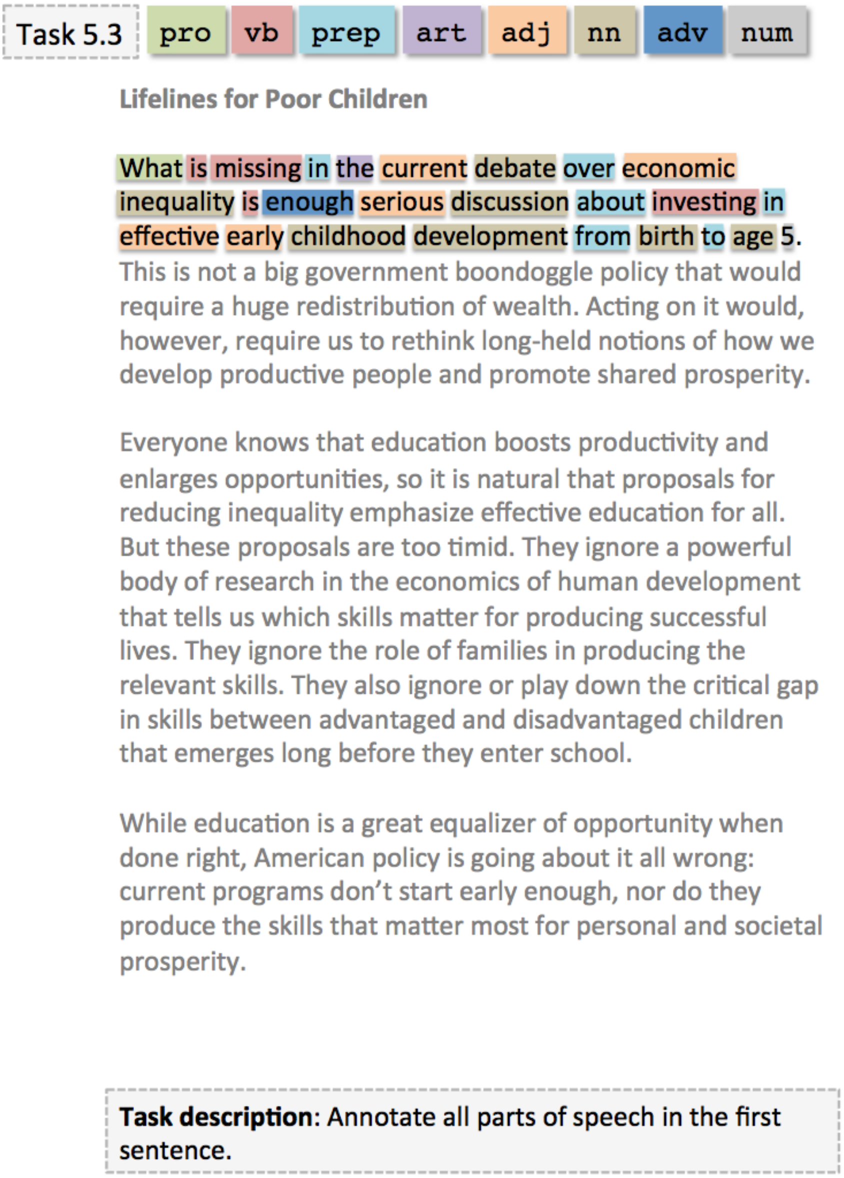
What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don’t start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

Task description: Annotate all periods in the first paragraph.

Figure B.2.: Solution for Task 5.2.



Lifelines for Poor Children

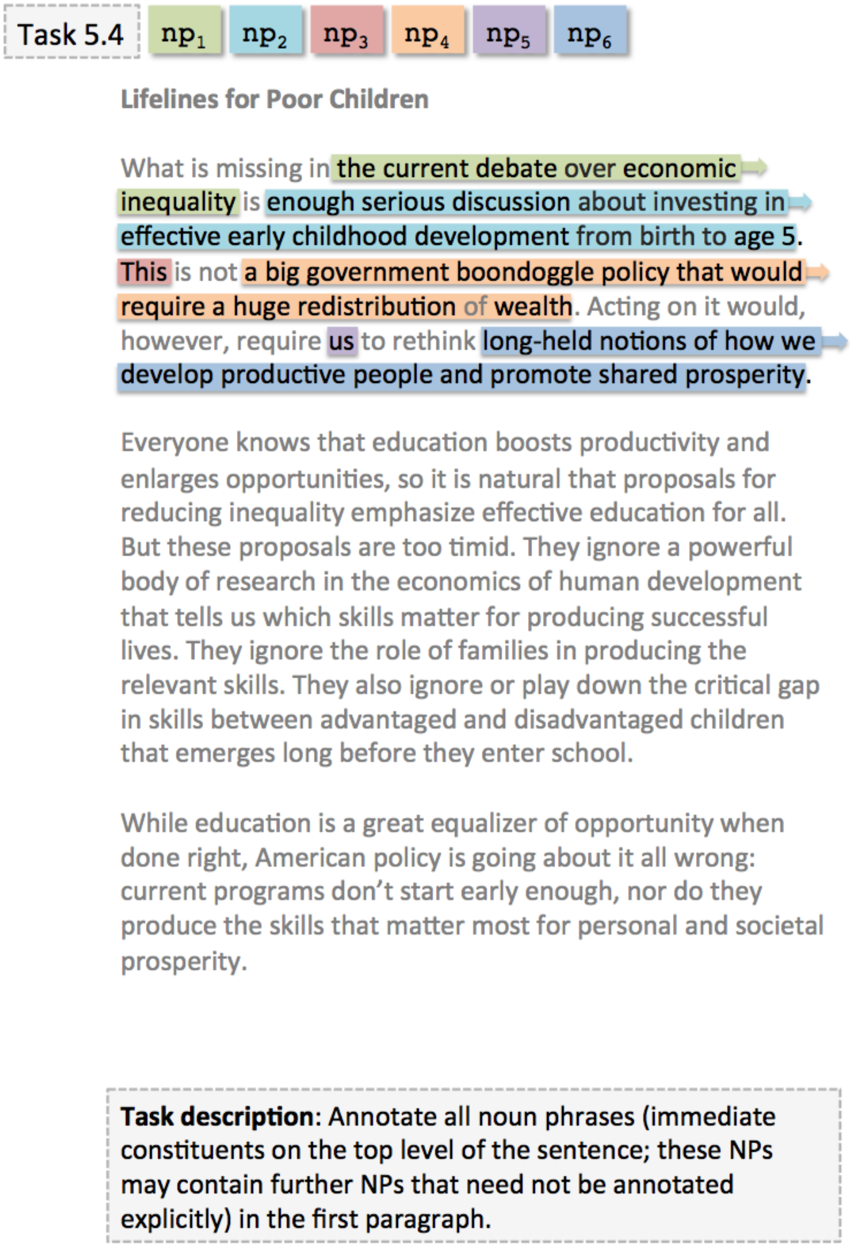
What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don’t start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

Task description: Annotate all parts of speech in the first sentence.

Figure B.3.: Solution for Task 5.3.



Lifelines for Poor Children

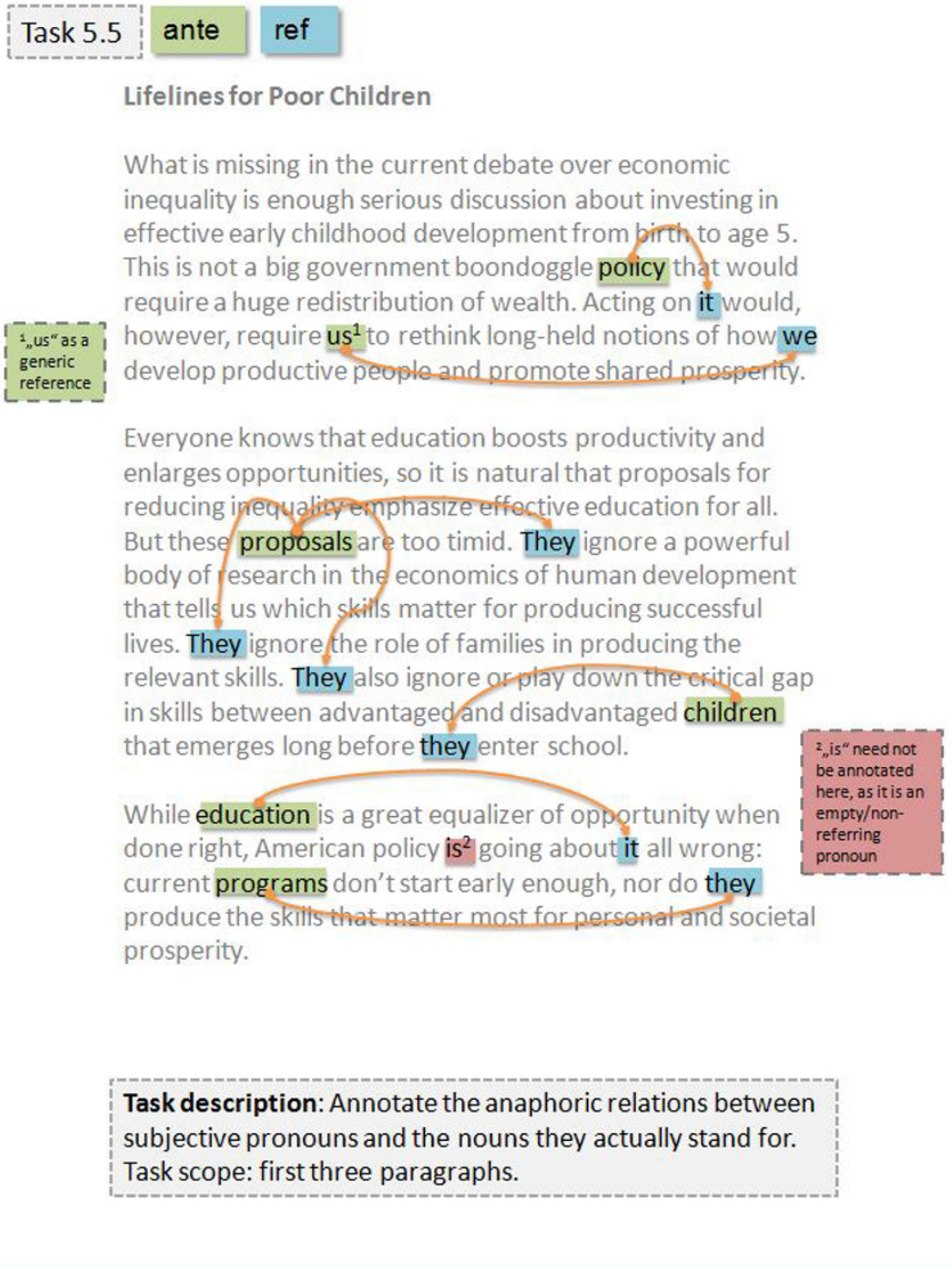
What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

Task description: Annotate all noun phrases (immediate constituents on the top level of the sentence; these NPs may contain further NPs that need not be annotated explicitly) in the first paragraph.

Figure B.4.: Solution for Task 5.4.



Task 5.5 ante ref

Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us[1] to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is[2] going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

[1] „us“ as a generic reference

[2] „is“ need not be annotated here, as it is an empty/non-referring pronoun

Task description: Annotate the anaphoric relations between subjective pronouns and the nouns they actually stand for. Task scope: first three paragraphs.

Figure B.5.: Solution for Task 5.5.

C. Reviews: Linguistic annotation tools

This section of the appendix contains a short review of all the linguistic annotation tools that were tested during the heuristic walkthrough. There is a short documentation of the basic steps that were necessary in order to achieve the six different tasks (cf. the list of tasks at the end of this paragraph) as well as a short summary of the most important characteristics of the respective tool (e.g. basic architecture, main functions, etc.). The tools are presented in the order in which they have been evaluated.

Task 1 Install and setup the annotation tool, so it can be used to achieve the consecutive tasks.

Task 2 Import the primary data into the annotation tool, so it can be viewed and annotated.

Task 3 Create an annotation scheme for the following annotation levels: structure, punctuation, parts of speech, phrases and coreference.

Task 4 View the annotation scheme during the annotation process and check if it is possible to edit the scheme inside the tool.

Task 5 Annotate the primary data on different annotation levels as defined in the annotation scheme (cf. Task 3). For every annotation subtask explore the tool's interface for creating, editing and deleting the anchor as well as the actual annotation. Also examine in detail how annotations are attached to an anchor in the primary data.

Task 6 Explore how the different levels of annotation are visualized and how they can be distinguished from each other. Also check if it is possible to view and hide selected levels of annotation.

C.1. Dexter

Task 1 – Dexter can be installed by means of a one-click installation (Java Web-Start).

Task 2 – There is a comprehensive manual in the "help menu" that contains a "read this first" section; this section tells the user that it is important to convert the primary data, so Dexter can read / process it. In order to achieve the second task, "Dexter Converter" must be downloaded. The Converter can be installed as easy as the Coder. The converter tool is basically a 3-step-dialogue that explains how to import a document and convert it to Dexter's markup format. However, the numerous encoding parameters are hard to understand - obviously, Dexter assumes that the primary data is already encoded / formatted in some way; there are some errors with the whole document (possibly the double-quotation marks cause some trouble). After reducing the text to its first three paragraphs the conversion works without problems. The conversed document can be imported in Dexter Coder without problems ("open document"). It is not necessary / possible to create a specific project, or combine multiple documents to form a corpus.

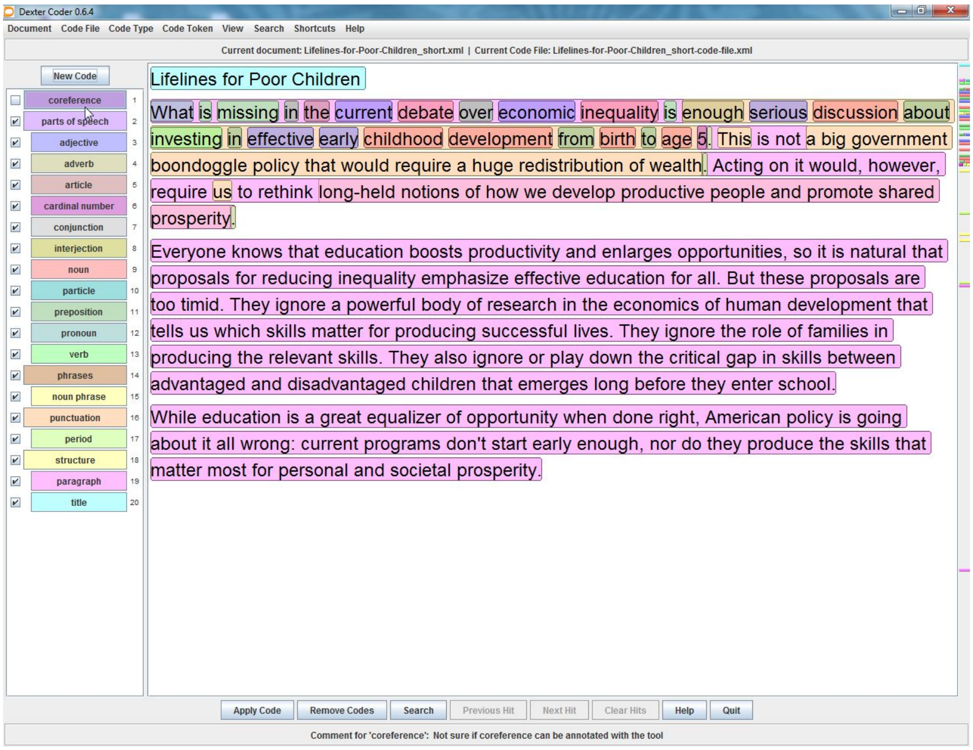
Task 3 – Schema creation happens inside the tool. Annotation items are called "codes" in Dexter Coder. The "New Code" button opens a small window where the user can name the annotation item, mark it as a subcategory of an existing annotation item, comment about the item and set its color. Each item created in this way is displayed (in alphabetical order) on the left side of the screen. Sub-items are displayed with an indention. New codes can be created with a shortcut, but only be deleted via the menu "Code Type > Delete Code Type". The coding scheme may be exported as stand-off XML. The annotation of relations and references is not supported.

Task 4 – Items (codes) from the scheme can be created, edited and deleted inside the tool. The alphabetical order of the items, however, cannot be changed. There are no shortcuts for editing and deleting codes. When right clicking a code, there is no context menu. The user rather has to navigate to the "CodeType" menu (top of the screen) every time he wants to edit or delete a code.

Task 5 – The annotation process ("coding") is very intuitive: the user selects a code from the scheme, then selects (click-drag-release OR double click a word) an anchor in the primary text. To apply the annotation the user has to click on "Code Token" > "Apply Code", or preferably use one of the three (redundant) shortcuts. Once an annotation is applied to an anchor, it can only be deleted, but not edited (e.g. the scope of the anchor). The annotation of relations between codes is not supported by Dexter Coder.

Task 6 – Any code (or even the whole parent category) can be hidden or shown via a checkbox. The annotations in the text are displayed with highlights in the code's specific color.

Summary – Dexter consists of two components that are downloaded and operated separately: (1) The "Dexter Converter" transforms the primary data into an XML / TEI-based format ("dubbed DexML"). The (2) "Dexter Coder" allows the user to import DexML-data. The Coder displays the primary data and allows the user to annotate it (stand-off XML). Dexter also provides an integrated search tool that can be used to query the annotated document.



Current document: Lifelines-for-Poor-Children_short.xml | Current Code File: Lifelines-for-Poor-Children_short-code-file.xml

Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

Comment for 'coreference': Not sure if coreference can be annotated with the tool

Figure C.1.: Screenshot of the Dexter Coder annotation tool.

C.2. CATMA

Task 1 – In its current version, CATMA is available as a web service. A previous version that can be installed locally is also available for download. CATMA was tested as a web version. The login requires a *Google* account and works without problems.

Task 2 – The tool interface shows rich functionality, including three buttons named "Create corpus", "Open document" and "Add document". The "Add Document" button calls a dialog with four steps: (1) Upload a document (via URL or upload dialog), (2) preview of the document (does not show the whole imported text) and file type setup, (3) wordlist options (language, word separators), and (4) add metadata (title, author, description, publisher). After importing the primary data, the user can create a corpus. Documents are added to the corpus via drag-and-drop.

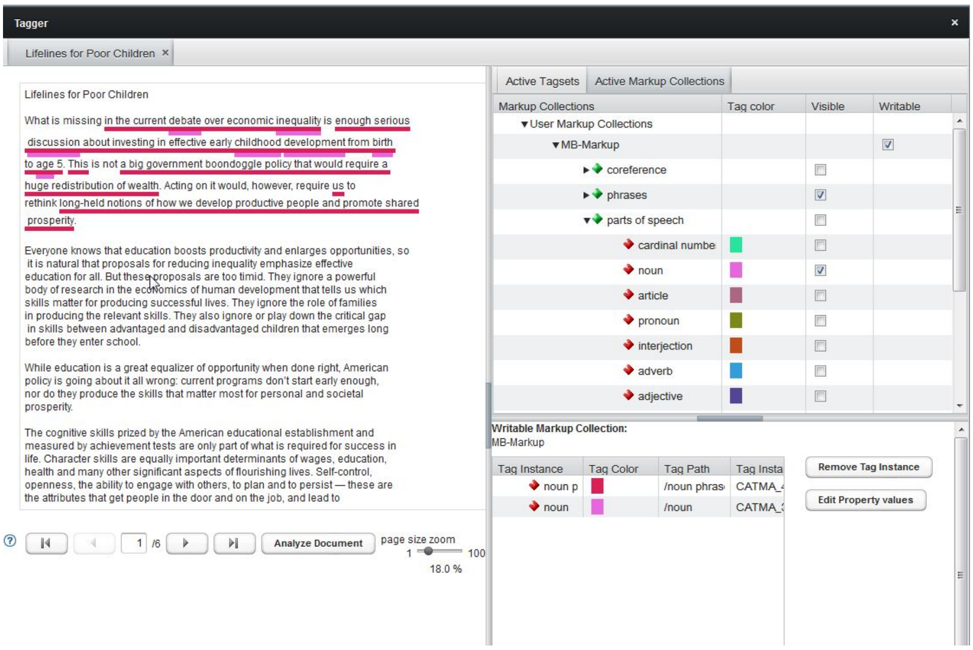
Task 3 – In CATMA, the annotation scheme is called a "markup collection". By default, there exists a "user markup collection" and a "static markup collection". Additionally, a "tag library" may be opened or created. The manual describes the difference between tags and markup as follows: (1) "tags" are the generic annotation items, whereas (2) "markup" are the tags applied to specific anchors in the document. The markup is what is actually stored as an annotation. There are many different levels for creating hierarchies when building an annotation scheme: "tag library > tag set > tag > sub tag > . . . > property > values". For each tag, CATMA suggests a distinctive color, but also allows the user to select a custom color from a broad color palette. In order to associate one or more tag sets with a document, the tag sets have to be moved into the "Tagger" window by means of drag-and-drop.

Task 4 – It is possible to change the name of an existing tag set or to delete the entire tag set. Items from the tag sets can be removed or edited (name and color). Properties and values may be added to an existing tag.

Task 5 – The annotation process works like this: (1) Select an anchor in the text (left side) and (2) click on the "tag color" from the tag set to apply the tag value to the anchor. As the text cannot be enlarged, selecting small anchors, like e.g. short words or punctuation marks, is cumbersome. If a word is double clicked, it also selects the following white space. If an anchor is clicked, the parallel annotations are displayed in an "inspector-like" window on the right side, named "Writable Markup Collection". Tags may be removed inside this window. The same applies for setting the values for a property.

Task 6 – By switching to the "Active Markup Collections" tab, any tag (or the whole parent category) can be hidden or shown via a checkbox. The annotations in the text are displayed with underlines in the tag's specific color.

Summary – CATMA is a rich web application. Many of the interaction metaphors originate from traditional desktop applications, both technically and conceptually (e.g. "drag-and-drop" and "multiple windows"). CATMA consists of a number of different modules: (1) the "Repository Manager" allows the user to save corpora and documents. (2) The "Tag Manager" is a module for creating, editing and storing annotation schemes. (3) The "Tagger" is a module that implements the actual annotation process, whereas the (4) "Analyzer" is a built-in search application that allows the user to query the annotated document, or that simply shows a list of concordances for a document. Results from the "Analyzer" can be displayed as graphs or double trees via the "Visualizer".



Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

The cognitive skills prized by the American educational establishment and measured by achievement tests are only part of what is required for success in life. Character skills are equally important determinants of wages, education, health and many other significant aspects of flourishing lives. Self-control, openness, the ability to engage with others, to plan and to persist — these are the attributes that get people in the door and on the job, and lead to

Markup Collections	Tag color	Visible	Writable
▼User Markup Collections			
▼MB-Markup			☑
▶ coreference		□	
▶ phrases		☑	
▼ parts of speech		□	
cardinal numbe	🟩	□	
noun	🟪	☑	
article	■	□	
pronoun	■	□	
interjection	🟧	□	
adverb	🟦	□	
adjective	🟪	□	

Writable Markup Collection:
MB-Markup

Tag Instance	Tag Color	Tag Path	Tag Insta
noun p	🟥	/noun phras	CATMA_4
noun	🟪	/noun	CATMA_3

Figure C.2.: Screenshot of the CATMA annotation tool.

C.3. Glozz

Task 1 – Glozz can be installed by means of a one-click installation (JAR executable). In order to store data persistently, the user needs to get an account from the developers via email (response within 24 hours).

Task 2 – The primary data can be imported via the "Import" button. A dialog asks the user to specify the paths to the primary data (*.txt), an output file (*.ac), and an annotation file (*.aa). After the import, the previously created *.ac and *.aa files have to be opened via "File > Open". After these steps, the primary data is displayed in the main window, with line breaks and indents after every paragraph. The layout looks like a print layout (serif font, top, left and right margins). The whole text can be accessed via a scrollbar, or by clicking into a macro-view of the whole document on the left side. Whenever the mouse cursor is moved somewhere in the document, the position is highlighted in the macro-view (and vice versa).

Task 3 – The annotation scheme (=annotation model) needs to be created outside of the tool, in a code editor that allows the user to create XML documents. The basic components (units, relations, schemas) are explained in the manual. It is possible to define a set of features for each item (=type). Each feature can be related to a set of values. Each type can also be marked as belonging to one or more groups (important for later show / hide feature in the tool). The scheme needs to be imported into the tool; colors for each item in the scheme need to be set in the tool. In order to display the color highlighting, the "Color mode" needs to be changed to the value "Individual Stylesheet".

Task 4 – Once imported, the annotation scheme is displayed in the tool: Units, relations and schemes are displayed in the "annotation model" pane (default: right hand side). There is also a "feature sets table" that displays features and values of an annotation item (if these have been defined in the annotation scheme). It is not possible to edit the scheme inside the tool. If the XML file is modified, it has to be re-imported. There is also a reload ("Load last job") function inside the tool.

Task 5 – Switch from "default mode" (no interaction with the primary text; read only) to one of three annotation modes. There is a mode for "unit annotation", "relation annotation" and "schema annotation" (the three basic concepts of the Glozz annotation model). Each mode can be distinguished by two sub-modes, one for the creation of the respective element, and one for editing or deleting an existing element (note: the schema annotation mode has even more sub-modes which are not described here, as they are not part of the evaluation tasks). When the user is in "create" mode, the respective annotation items are highlighted (in a color that has to be defined previously) in the annotation scheme pane.

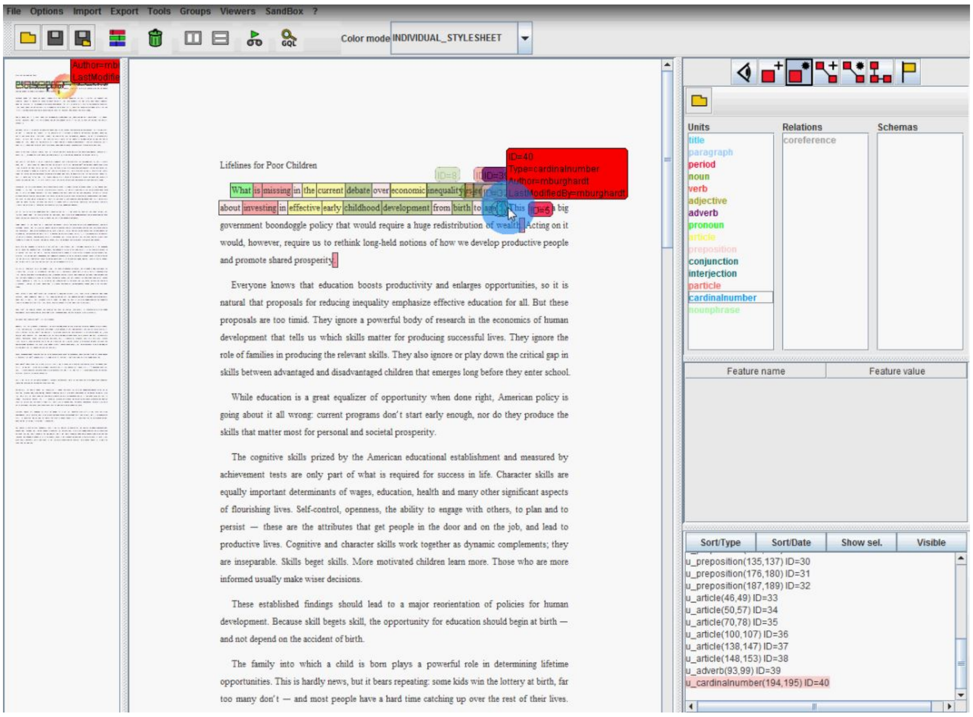
Unit annotation: The user can then specify anchors for units in several ways: (1) By drag-and-drop, (2) by clicking on the start point and on the end point of the anchor (visual aids provided), and (3), if the anchor is a single word, by double clicking on the word (this option must be enabled first in the preferences menu).

Relation annotation: The user switches to "relation annotation" mode, selects the type of relation from the annotation scheme, and then selects the two units that are to be related. Note: The user can only create relations between existing units. There is a visual hint, i.e. once the user clicks on the first element, an arrow appears that can be moved to the second element.

Task 6 – If colors have been defined by the user, and the "Individual Stylesheet" has been selected, the different annotations are highlighted in the respective color. Parallel annotations overlap with an adjustable degree of transparency ("Options > Preferences > Viewer"). Annotations are not only shown as colored highlights in the primary data, but also in a textual view in an "annotation table". Each annotation is one row in the table, containing information about the actual annotation value, beginning and end of the anchor, and a "user friendly" ID that may change from session to session.

There are several ways to show and hide annotations in the document: (1) By clicking on the "Open style editor" button it is possible to hide any of the annotation items. (2) If "groups" have been specified in the annotation scheme, it is possible to hide groups of annotation items via "Groups > Manage Visibility". Hidden items are crossed out in the annotation model. (3) Single annotated units can be hidden by double clicking them in the "annotation table". If an annotation has been hidden, it is crossed out in the table. There is also a button "Visible" that makes visible all hidden annotations again. Finally, there is a "Viewer" menu that can be used to display the annotations in two different ways: (1) Aligned with the primary data (timeline-like view) and as a (2) graph.

Summary – Glozz is a desktop application that relies on a specific meta-model which comprises *units*, *relations* and *schemas*. The tool requires several manually created files before it can be used: a corpus file, an annotation file, and an annotation model (=annotation scheme) file. The annotation scheme must be created outside of the tool, and is defined by means of XML. Besides the annotation functionality, Glozz provides a sophisticated query language (GQL) that can be used via a GUI. There are also different "Viewers" ("Grapher" and "Aligner") which allow the user to visualize and analyze the annotated document.



Color mode INDIVIDUAL_STYLESHEET

ID=40
Type=cardinalnumber
Author=mburghardt
LastModifiedBy=mburghardt

Lifelines for Poor Children

What is missing in the current debate over economic inequality [illegible] about investing in effective early childhood development from birth to [illegible] This [illegible] a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

The cognitive skills prized by the American educational establishment and measured by achievement tests are only part of what is required for success in life. Character skills are equally important determinants of wages, education, health and many other significant aspects of flourishing lives. Self-control, openness, the ability to engage with others, to plan and to persist — these are the attributes that get people in the door and on the job, and lead to productive lives. Cognitive and character skills work together as dynamic complements; they are inseparable. Skills beget skills. More motivated children learn more. Those who are more informed usually make wiser decisions.

These established findings should lead to a major reorientation of policies for human development. Because skill begets skill, the opportunity for education should begin at birth — and not depend on the accident of birth.

The family into which a child is born plays a powerful role in determining lifetime opportunities. This is hardly news, but it bears repeating: some kids win the lottery at birth, far too many don't — and most people have a hard time catching up over the rest of their lives.

```
u_preposition(135,137) ID=30
u_preposition(176,180) ID=31
u_preposition(187,189) ID=32
u_article(46,49) ID=33
u_article(50,57) ID=34
u_article(70,78) ID=35
u_article(100,107) ID=36
u_article(138,147) ID=37
u_article(148,153) ID=38
u_adverb(93,99) ID=39
u_cardinalnumber(194,195) ID=40
```

Figure C.3.: Screenshot of the Glozz annotation tool.

C.4. UAM Corpus Tool

Task 1 – UAM Corpus Tools can be installed by means of a one-click installation (installer available).

Task 2 – When starting the tool, the user is asked to create a new project or to open an existing project. When creating a new project, the user has to specify a name and the location where the project will be stored. Once the project has been created / opened, UAM Corpus Tool opens the main window. In order to import the primary data, the "Extend Corpus" button must be clicked. An assistant opens, asking the user to select one or more documents for the import to the tool. Afterwards, the file name of the document is displayed in the tool; by clicking on it, the primary data is displayed. In order to annotate the document, it must be added to the current project via "Action > Incorporate File" or alternatively "Incorporate all", if multiple documents are to be added to the project.

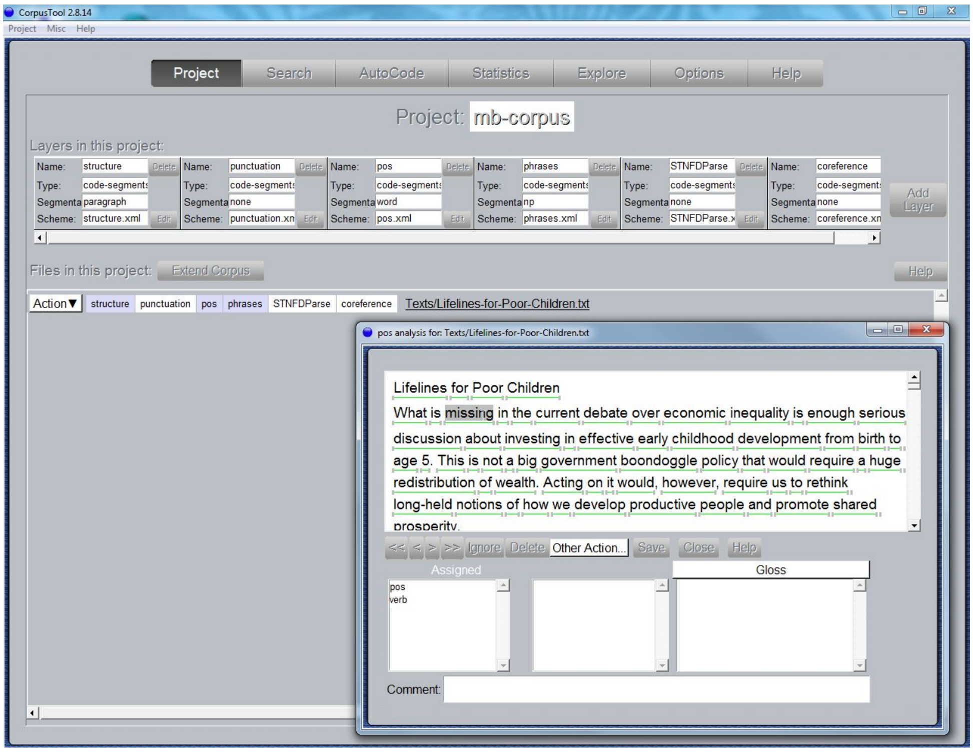
Task 3 – In order to create an annotation scheme, the user must click on the "Add Layer" button. This opens an assistant for the creation of one layer of annotation. The user has to specify the name of the annotation layer and whether the document will be annotated as a whole, or whether single segments will be annotated. Further options are "Automatic Grammar Analysis" and "Rhetorical Structure Analysis". When "Annotate Segments" is selected, the assistant displays a menu with options for "Automatic Segmentation" for paragraphs, sentences, clauses, NPs and words. Next, the type of segment that is to be created has to be specified. The assistant offers two choices: plain text segments and error segments. Finally, the user can create a new scheme or copy an existing scheme. After finalizing the layer, it is displayed in the main window with its name, type of coding object, type of automatic segmentation and the name of the XML file that is generated for storing the layer of annotation. By clicking on the "Edit" button, the user can specify annotation items on the annotation layer. A window opens with a graph-like representation of the annotation layer. By default, there is a root node (named after the layer), an edge (labeled with the layer name and the word "type"), and two children nodes named "structure 1" and "structure 2" (these are exemplary features). By clicking on the edge, a context menu opens that allows the user to add new "features" to the scheme. Each item on a layer is realized as a feature.

Task 4 – Each layer of annotation is displayed in the main window. In order to display or edit the items of a layer the user needs to click on the "Edit" button. Clicking on the "Delete" button deletes the layer with all its annotation items. By clicking (left or right) on an element in the graphical annotation scheme window, a context menu opens that allows the user to rename, delete or hierarchically move an annotation item.

Task 5 – The primary data can be annotated on the specified layer of annotation by clicking on the name of the layer, which is displayed as a text link on the left side. The primary text is displayed with the predefined segmentation. The segmentation for paragraphs, words and even NPs works very well. Segments are displayed with a green underline. Clicking on a segment (word or underline), the predefined items for this annotation layer appear in a pane beneath the text. By double clicking on an item, its value is applied to the segment. This is displayed as moving the item to the "Assigned" pane. The user may navigate from segment to segment by using the arrow buttons. There is also a "Delete" button for deleting a selected segment. Clicking on "Other Action..." makes available several actions, e.g. editing the annotation scheme or re-segmenting the document.

Task 6 – It is not possible to show annotations from multiple layers at the same time, but rather to access the primary text with only one annotation layer at once. All segments are visualized as green underlines. Parallel segments are displayed as parallel underlines. The value of the annotation is displayed in the bottom pane when it is selected. It is not possible to distinguish different annotation values in the primary text without clicking on an anchor. By clicking on "Other Action..." it is possible to view the annotations in XML format, in a structured format (only seems to work correctly for the automatic Stanford parsing layer), and a text stream visualization that shows the distribution of different annotation items graphically (different colors).

Summary – UAM Corpus Tool is a desktop application that is the successor of the previously developed *Systemic Coder*[198]. This might explain the conceptual model of the annotation scheme, which seems to be inspired by the idea of systems and features that can be related in a tree-like hierarchy. The tool provides some helpful auto-segmentation options that can help to speed up the annotation process. UAM Corpus Tool also has an "AutoCode" function that allows the user to specify rules for the automatic annotation of certain segments. There is a built-in search function that can be used to query the annotated corpus; there are also some basic statistics and concordance lists, which are useful for the exploration and analysis of the corpus data.



Project: mb-corpus

Layers in this project:

Name:	structure	punctuation	pos	phrases	STNFDParse	coreference
Type:	code-segment:	code-segment:	code-segment:	code-segment:	code-segment:	code-segment:
Segmenta	paragraph	none	word	np	none	none
Scheme:	structure.xml	punctuation.xn	pos.xml	phrases.xml	STNFDParse.x	coreference.xn

Files in this project:

structure punctuation pos phrases STNFDParse coreference Texts/Lifelines-for-Poor-Children.txt

pos analysis for: Texts/Lifelines-for-Poor-Children.txt

Lifelines for Poor Children
What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

pos
verb

Comment:

Figure C.4.: Screenshot of the UAM Corpus Tool.

[198] http://www.wagsoft.com/Coder/

C.5. Brat

Task 1 – The installation for Windows 7 seemed somewhat troublesome, because Brat requires a UNIX environment[199]; i.e. the user has to set up a virtual machine. Brat also requires a web server: The user can either set up an Apache server or use the Brat *Python standalone server*.

Installation on a Mac on the other hand is relatively simple, as both UNIX and Python (needed for Brat) are available by default. After installing Brat via the command shell, and after setting up a user and password, the standalone server can be started with one command. Brat is then available as a local web service via localhost:8001. In order to make sure Brat works properly, either Safari or Chrome should be used as web browsers. Also, pop up blockers should be disabled.

Note: Brat can also be used as a web service that runs on a public server. It is, however, not possible to add own annotation schemes to this service, as it is rather used as a sandbox for playing around with Brat.

Task 2 – There are two ways to import primary data into Brat: (1) The user has to click on the "Data" button, which opens a dialog for the import of a new document. Next, the user needs to specify a document ID, a name, and the content of the document in a web form. (2) The user can also directly save the primary text as a *.txt file in the "data" folder on the server on which Brat is running. It is necessary to create an empty annotation file which has the same file name as the primary text and the file extension (*.ann). Brat provides a log in by means of a user name and a password. It is, however, not clear how different annotators can collaboratively annotate a document and how the different annotations are managed[200].

Task 3 – In order to create an annotation scheme, the user must create a new "annotation.conf" file in the same directory where the primary data (*.txt) and the annotation file (*.ann) are stored. The annotation scheme must be created and edited in a code editor. The syntax is explained in the "configuration" section of the web site[201]. It is quite intuitive, as no XML markup is used. There are four basic types of annotation items: *entities*, *relations*, *events*, and *attributes*.

Task 4 – The scheme that was created outside the tool is automatically associated with the primary data by putting it in the same directory. When selecting an anchor in the text, the annotation scheme opens automatically. It is not possible to open the scheme without selecting an anchor. It is also not possible to edit the scheme inside Brat. The scheme displays all defined entities (not the relations). The hierarchical entities can be expanded or collapsed in a tree-like

[199] Cf. Installation hints at `http://brat.nlplab.org/installation.html`.

[200] The feature of collaborative annotation is described on the Brat website (cf. `http://brat.nlplab.org/features.html`).

[201] Cf. `http://brat.nlplab.org/configuration.html`

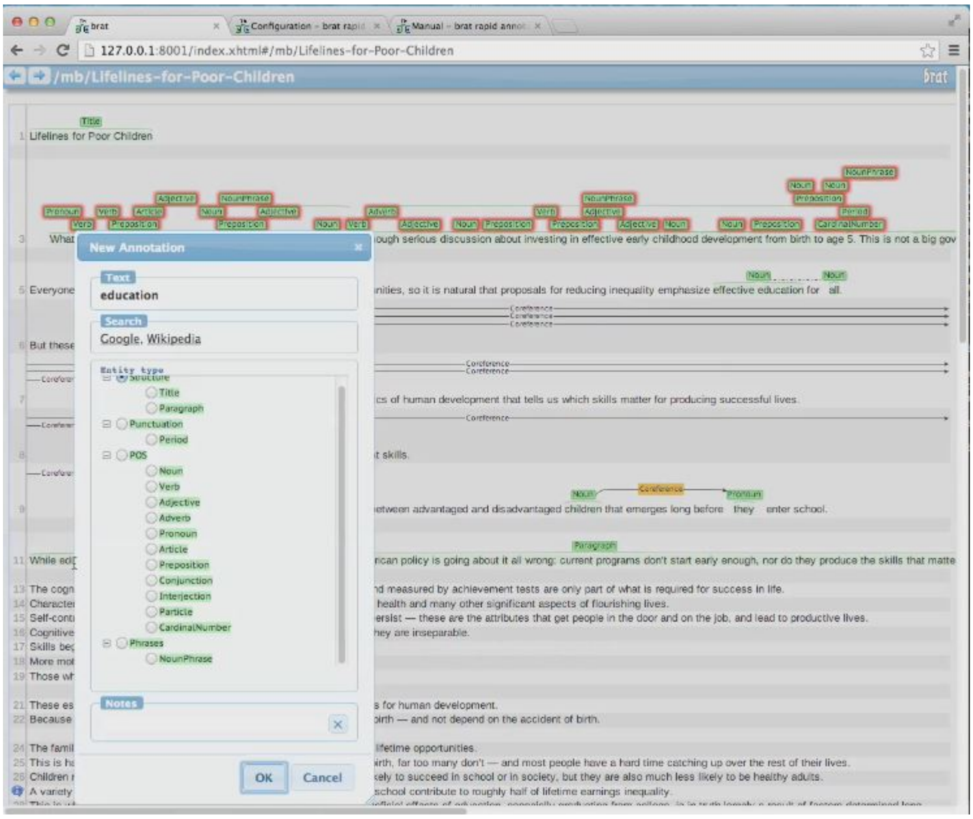
view. They are all displayed in the same standard color. The window that shows the annotation scheme can be resized to show longer lists of annotation items. Brat remembers the resizing when the scheme window is opened the next time. If something is wrong with the scheme (syntax, typos, etc.), Brat gives feedback in the form of error messages on the bottom of the screen.

Task 5 – In order to annotate, the user has to select an anchor in the primary text via click-drag-release (note: word anchors may also be selected by double clicking on the word). It is also possible to select multiple words as one anchor by using the keys CTRL and SHIFT. When releasing the mouse, an annotation window pops up which contains the items that were previously defined (cf. description of Task 4). The annotation window also displays the selected anchor as a textual string that may be looked up in Google or Wikipedia. Finally, it is possible to make notes for an annotation. An annotation item may be applied by selecting a radio button value and by clicking the "ok" button. It is also possible to speed up the annotation process by defining shortcuts for annotation items in the "kb_shortcuts.conf", and by enabling "Normal" annotation in the "Options" menu, which allows the user to apply an annotation by just selecting it and not having to confirm it each time with the "ok" button."

Relational annotations are realized by means of drag-and-drop. When the user clicks on an anchor and holds the mouse button down, an arrow head appears. The arrow head can be dragged to another anchor. Dropping the arrow head on another anchor creates a visible arrow that points from one anchor to the other.

Task 6 – The annotations are displayed in the primary data as label in a predefined color. The labels are attached to an anchor region by means of a rotated curly bracket that spans the whole anchor. The color as well as the label text can be defined in a "visual.conf" file. The colors are specified by means of HTML color codes or the respective hex-codes. By default the label text is the name of the annotation item as defined in the scheme. It is, however, possible to define shorter, alternative labels that will be displayed in case that there is not much space. Hovering over an existing annotation opens a small window that displays the anchor as a textual string as well as the annotation value and the ID of the annotation. It also displays potential annotation errors / problems, and any existing "notes" that have been made for this annotation. Relations are visualized as a directed arrow that can also be styled in the "visual.conf" file. The arrow has a label with the value of the relation and its ID. It also displays the two related entities and the type of relation in textual form. Multiple relations are arranged in a parallel fashion. If a relation spans multiple lines, the arrow spans till the end of one line and starts anew in the next line (this is repeated for multiple lines).

Summary – Brat is a web-based tool that relies on a client-server architecture. The tool supports all kinds of relational and referential annotation tasks. It is primarily used for tasks like "entity mention detection", "event extraction", "coreference resolution" etc.[202] The annotations are stored in a stand-off format that is not XML-based, but rather similar to the *BioNLP* stand-off format. Brat also comes with a built in search function. The tool provides many ways for the configuration of annotations and their visualization, which are all defined in specific files outside of the tool. Although Brat is still evolving, there is a fork of the tool called *WebAnno*[203], which augments the features of Brat and is developed independently at the TU Darmstadt.



Text: education

Search: Google, Wikipedia

Notes

1 Lifelines for Poor Children
3 What … ough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big gov
5 Everyone … nities, so it is natural that proposals for reducing inequality emphasize effective education for all.
6 But these
7 … cs of human development that tells us which skills matter for producing successful lives.
8 … t skills.
9 … etween advantaged and disadvantaged children that emerges long before they enter school.
11 While ed … rican policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matte
13 The cogn … d measured by achievement tests are only part of what is required for success in life.
14 Characte … health and many other significant aspects of flourishing lives.
15 Self-cont … ersist — these are the attributes that get people in the door and on the job, and lead to productive lives.
16 Cognitive … hey are inseparable.
17 Skills beg
18 More mo
19 Those wh
21 These es … s for human development.
22 Because … birth — and not depend on the accident of birth.
24 The famil … lifetime opportunities.
25 This is h … birth, far too many don't — and most people have a hard time catching up over the rest of their lives.
26 Children … kely to succeed in school or in society, but they are also much less likely to be healthy adults.
A variety … school contribute to roughly half of lifetime earnings inequality.

Figure C.5.: Screenshot of the Brat annotation tool.

[202] Cf. http://brat.nlplab.org/examples.html
[203] Cf. http://code.google.com/p/webanno/

C.6. MMAX2

Task 1 – MMAX2 can be downloaded as a *.zip-archive. Once the archive is extracted, MMAX2 can be started via the "mmax2.bat" file by simply clicking it (Java needs to be installed). A terminal window opens in which MMAX is executed - this window may not be closed, as it would terminate the tool (it can, however, be minimized). Note: In order to avoid complications when importing the primary data, MMAX2 should be installed somewhere on the C-drive (not on the desktop, as this creates whitespace characters in the internal project paths, which may cause troubles).

Task 2 – Upon starting the tool, the user sees an empty main window with a menu bar. The only available menu items (all others are grey) are "File", "Tools" and "Info". The file menu only allows the user to load saved MMAX2 projects, to save the current project (which does not make much sense at this point in time), or to exit the tool. Obviously, the primary data has to be imported in some other way. The "Tools" menu contains an item called "project wizard" (all other options in this menu are grey, which means they are not available). Clicking on "project wizard" opens a new window with four basic sections:

(1) First, an input file has to be selected. The encoding (default: ASCII) can be set to different formats. It is also possible to select XML documents as input. Clicking on "Analyse File" imports the selected file and previews a fragment of it in a new window.

(2) After this step, the previously unavailable "Tokenize" button becomes available. The tokenizer offers a lot of options and parameters, but can also be used with the default settings (note: this becomes obvious through trial and error).

(3) Next, at least one "markable level" needs to be defined in order to create a project. According to the manual, "markable levels" in MMAX2 are the basic annotation levels. Interestingly, the definition of annotation levels happens in the same dialog that is also used for importing the primary data and for creating the project files. In the strict sense, this is not really a "project wizard", but rather the opposite of a step-by-step guide through a complex process (which a wizard typically is). In fact, many different aspects are mixed together in one dialog. Markable levels can be assigned a color and a visual delimiter (round or square brackets). It is also possible to define words as predefined anchors for a markable level (e.g. for POS).

(4) After at least one "markable level" has been added, the MMAX2 project can be created. This includes the creation of many different files: "project file", "words file", "base data file", "style file", "markable file", "scheme file", and "customization file". It is possible to select one generic project path (i.e. select the path to a previously created folder) for all the necessary project files by clicking on the "use for all" option next to the "project path" form. Note: The folder that will con-

tain the project files should be created in advance. It should also be created on the C-drive, to avoid problems with whitespace characters in the project paths.

After the creation of the project, the primary data is displayed in the main window. Once a project has been created, it can be loaded via the "File > Load" dialog for future use. MMAX2 asks the user if he wants to validate the annotations whenever a project file is opened.

Task 3 – In MMAX2 all annotation levels must be declared at the start of the project, inside the project wizard. If levels are added or removed during the process, the different project files (e.g. "default style", "common paths", etc.) need to be adjusted manually (i.e. edit XML files). The actual definition of the items for each annotation level happens outside the tool. The project wizard automatically creates an empty XML file for each annotation level that can be adjusted by adding new items (the syntax is described in the manual; there are also predefined examples that may be adapted). It is possible to define "attributes" (regular annotation items) and "relations". Attributes comprise an ID, a name, and a set of values, which may be free text, a nominal list (drop-down list of all values) or nominal buttons (horizontal alignment of all values with radio buttons). Relations may be of the type "markable set" (group items)[204] and "markable pointers" (point from one item to another).

Task 4 – There is a main window that displays the primary data and a smaller "attribute window" that displays the annotation scheme. There is also a tab for each level of annotation. The annotation scheme can, however, only be modified outside of MMAX2, in an XML editor. In order to update the annotation schemes in MMAX2, there is the option "Panel > Update current panel" in the attribute window.

Task 5 – In MMAX2, anchors are called "markables". Markables are rendered in the primary data and may contain annotations on multiple, parallel levels. The annotation process in MMAX2 relies on mouse actions (left clicks, right clicks, click-drag-release) alone:

Creating an annotation

- Select a markable (=anchor) by double clicking a single word or by clicking-and-releasing over a larger span of text.
- Choose the level of annotation for the selected markable from a pop-up menu. The markable is now displayed with a yellow background when it is clicked.
- Select a value from the "attribute window" and confirm the annotation by clicking the "Apply" button (there is also an "Auto-apply" option, which makes explicit applying via the button unnecessary). The annotation value for an anchor will always be displayed in the attribute window.

[204] According to the sample schemes, coreference in this evaluation was modeled as a "markable set".

Deleting an annotation

- In order to delete an existing markable, right click it, and select "Delete this markable" from the pop-up menu.

Changing the scope of an anchor

- Expand scope: Select an existing markable (left click), select another word or span of text (click-drag-release) and confirm the extension of the markable by clicking on the pop-up message "Add to this markable"
- Reduce scope: This action implies that there are at least two different words in one markable. Select the word that will be kept in the markable (left click) and select the word that will be removed by means of click-drag-release. Confirm the removing by clicking on "Remove from this markable".

Creating a relation

- Create two or more "markables" on the annotation level that was defined as a "relation".
- Select the first markable (left click), then right click on another markable (on this level of annotation) to create a relation.
- The relation is displayed as a colored arc.

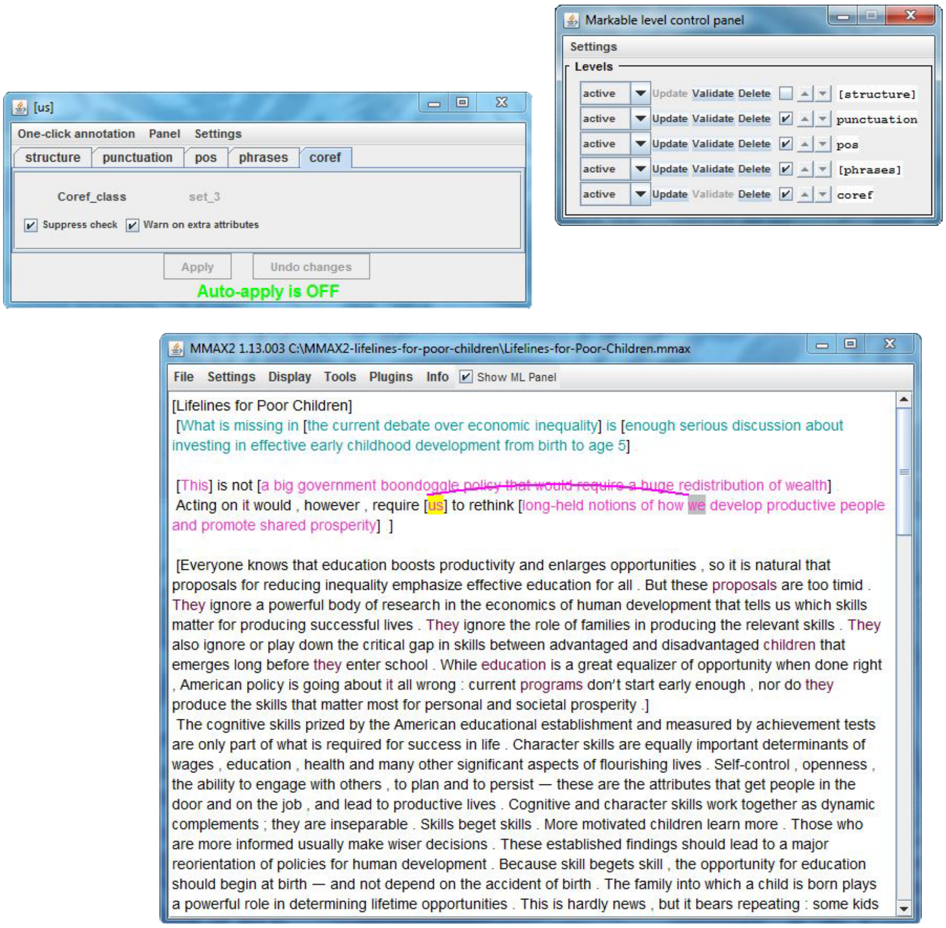
Deleting a relation

- To delete a relation, click one of the markables that are in the relation (to select the relation). Then right click on the markable that is to be deleted, and remove it from the relation (pop-up menu).

One special feature of MMAX2 is a mode in which "base data" (=primary data) can be edited. This mode needs to be activated via "Settings > Enable base data editing". By holding the CTRL-key and right clicking on a word, a pop-up menu allows the user to edit the word. There is also a feature called "one click annotation", which is also briefly described in the manual – it did, however, not become clear how this feature can be used effectively during the evaluation.

Task 6 – The markables are displayed in the main window with the color and the "handles" that were defined via the project wizard. It is possible to write a customized stylesheet by means of XSL (outside the tool). If defined in the wizard, there are line breaks after some markables. The primary data may be adjusted in terms of font family, font size or line height via "Display > Font". The different levels of annotations are displayed as an overview in the "markable level control panel". Each level can be displayed or hidden. It is also possible to set a level to "inactive", which means it is not available during the annotation process. If all levels are set to "visible", higher order levels cover lower order levels, i.e. it is not possible to display parallel levels at the same time.

Summary – MMAX2 is a Java-based tool that allows the user to define arbitrary levels of annotation in the form of external XML files. The tool can import data that already has XML markup (*markup awareness*). One characteristic of MMAX2 are its versatile features to define relations between items of annotations. Another characteristic is a mode in which base data can be edited inside the tool. There are many ways for individual styling of the primary data as well as the annotations, either with display options inside the tool, or with sophisticated XSL stylesheets that can be specified outside the tool. It is also possible to analyze the annotations inside the tool with the "markable browser" or by using a "query console", which supports a specific query language called (MMAXQL).


[us]
One-click annotation
Panel
Settings
structure
punctuation
pos
phrases
coref
☑ Suppress check
☑ Warn on extra attributes
Apply
Undo changes
Auto-apply is OFF
Markable level control panel
Settings
Levels
active Update Validate Delete □ [structure]
active Update Validate Delete ☑ punctuation
active Update Validate Delete ☑ pos
active Update Validate Delete ☑ [phrases]
active Update Validate Delete ☑ coref
MMAX2 1.13.003 C:\MMAX2-lifelines-for-poor-children\Lifelines-for-Poor-Children.mmax
File
Settings
Display
Tools
Plugins
Info
☑ Show ML Panel

Coref_class set_3

[Lifelines for Poor Children]
[What is missing in [the current debate over economic inequality] is [enough serious discussion about investing in effective early childhood development from birth to age 5] .

[This] is not [a big government boondoggle policy that would require a huge redistribution of wealth] . Acting on it would , however , require [us] to rethink [long-held notions of how we develop productive people and promote shared prosperity] .]

[Everyone knows that education boosts productivity and enlarges opportunities , so it is natural that proposals for reducing inequality emphasize effective education for all . But these proposals are too timid . They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives . They ignore the role of families in producing the relevant skills . They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school . While education is a great equalizer of opportunity when done right , American policy is going about it all wrong : current programs don't start early enough , nor do they produce the skills that matter most for personal and societal prosperity .]
The cognitive skills prized by the American educational establishment and measured by achievement tests are only part of what is required for success in life . Character skills are equally important determinants of wages , education , health and many other significant aspects of flourishing lives . Self-control , openness , the ability to engage with others , to plan and to persist — these are the attributes that get people in the door and on the job , and lead to productive lives . Cognitive and character skills work together as dynamic complements ; they are inseparable . Skills beget skills . More motivated children learn more . Those who are more informed usually make wiser decisions . These established findings should lead to a major reorientation of policies for human development . Because skill begets skill , the opportunity for education should begin at birth — and not depend on the accident of birth . The family into which a child is born plays a powerful role in determining lifetime opportunities . This is hardly news , but it bears repeating : some kids

Figure C.6.: Screenshot of the MMAX2 annotation tool.

C.7. WordFreak

Task 1 – WordFreak can be installed by means of a one-click installation (JAR executable).

Task 2 – The file menu does not contain an option to import primary data, but there is an "Add..." option in the project menu. The same option is also available as a large "Add" button on the top right of the main window. The tool asks if an annotation file should automatically be created for the imported primary data. After this step, the text document appears in a file tree visualization. The document has to be loaded via the "load" option (large button on the right). After being loaded, a little green *check mark* appears next to the text. In order to display a text, the user first has to select a (predefined) annotation scheme from the "Annotation" menu "Set annotation...". Next, one of several available viewers has to be selected from the "Viewer" menu. In order to display the document as plain text, select the "text" option. A small tab named "Text" appears in the project window, which contains the primary data. The encoding, font family and font size can be set via the "Font" menu – changes are applied live, i.e. the user does not need to reopen or reload the document.

Task 3 – In WordFreak, there are a number of predefined annotation schemes ("Annotation > Set Annotation") that can be used for annotation:

- Constituent
- Document
- Generic
- NPCoref
- POS
- Paragraph
- PredArg
- Sentence
- TextCorrection
- Token

The "Paragraph" tag set can be used to annotate the basic structure of the document (note: there is only a "paragraph" tag, no "title" tag). The "POS" scheme contains the *Penn Treebank* tag set for the annotation of parts of speech. It also allows the user to annotate punctuation and generic tokens. The "Constituent" tag set contains tags for the annotation of phrases. Coreference can be annotated with the "NPCoref" tag set.

According to the WordFreak help file ("Developer Help > Plugin Development Overview") it is possible to add new annotation scheme plugins by creating XML files that contain the necessary information:

> AnnotationScheme plugins can be defined with XML files. This functionality is somewhat recent and is poorly documented. Again, we recommend that you start with an existing .wfplugin.xml file and modify it to suit you needs. (WordFreak, ver. 2.2; Help: Developing Plugins, para. 5)

As most of the annotation tasks of the evaluation could be achieved with the predefined schemes, and as the creation of new schemes is quite cumbersome, no new schemes were created for this evaluation.

Task 4 – The predefined annotation schemes can be displayed inside the tool. They open as a new window that can be moved and resized. The schemes cannot be edited. It is only possible to display one annotation scheme at a time. There seems to be an overlap between the schemes "POS" (only parts of speech) and "Constituent", which contains several subordinate tag sets, e.g. "POS", "Constituents" and "Null Elements". Each subordinate tag set can be accessed via a tab.

Task 5 – The annotation process in WordFreak is the same for all annotation schemes: The primary text is displayed (possibly in different views) in the main window, the annotation scheme (one at a time) is displayed in another window. All items (= tags) from a scheme are displayed as small buttons. In some schemes, there are subordinate tag sets that can be accessed via tabs. The anchors are selected by using click-drag-release interaction. The annotation is applied by clicking on the respective "item" button in the annotation scheme window. It is also possible to define anchors without an immediate annotation manually via the "plus" button. The same can be achieved automatically via the option "Tagger > Set Tagger > Simple Token". Anchors (and existing annotations) are deleted via the "minus" button. It is possible to navigate from one anchor to the preceding or succeeding anchor via arrow buttons. Four other arrow buttons can be used to increase or decrease the left or right scope of an existing (selected) anchor.

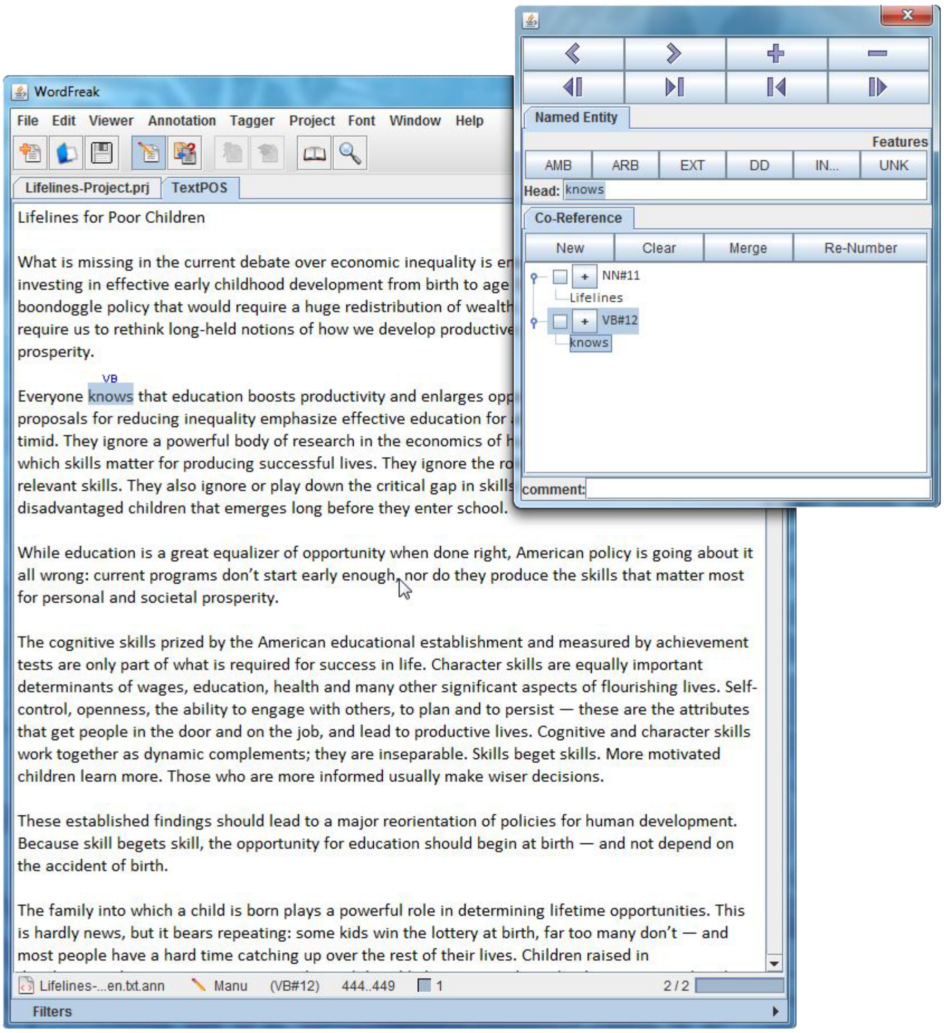
Task 6 – In WordFreak, it is possible to display the primary data and potential annotations in different views. These can be selected in the "Viewer" menu. A tab opens in the main window for each "view" (note: the text is only displayed if an annotation scheme is selected from the "Annotation"). The following views are available:

- Concordance: concordance list for every annotated item
- Table: tabular view of the annotated data with columns for text, type, annotator, comment, id, etc.
- Text: plain text view

- TextPOS: same as the text view, but with POS values above the annotated anchors
- Tree: tree-like visualization of the annotated document (text that has not been defined as an anchor is displayed in grey)
- TreeTable: tabular view with two columns; on the left side is a tree-like visualization of the annotated document, on the right side are the annotation values

It is not possible to display multiple, parallel annotations. Annotations are displayed in three basic ways: (1) above the anchor (cg. TextPOS view), (2) before the anchor (cf. Tree view), or (3) in a separate column (cf. Table view). The annotation values of an anchor are also displayed in the footer of the main window when the anchors are clicked.

Summary – WordFreak is a Java-based tool for both, manual and automatic annotation of text documents. Manual annotation is restricted to predefined levels of annotation such as POS. WordFreak can also be extended with a number of plugins (ACE, LingPipe, MUC, OpenNLP) to increase its automatic annotation functionality. It is not possible to analyze the data inside WordFreak, i.e. there is no built-in search or query functionality.



Lifelines for Poor Children

What is missing in the current debate over economic inequality is en
investing in effective early childhood development from birth to age
boondoggle policy that would require a huge redistribution of wealth
require us to rethink long-held notions of how we develop productive
prosperity.

Everyone knows that education boosts productivity and enlarges opp
proposals for reducing inequality emphasize effective education for
timid. They ignore a powerful body of research in the economics of h
which skills matter for producing successful lives. They ignore the ro
relevant skills. They also ignore or play down the critical gap in skills
disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

The cognitive skills prized by the American educational establishment and measured by achievement tests are only part of what is required for success in life. Character skills are equally important determinants of wages, education, health and many other significant aspects of flourishing lives. Self-control, openness, the ability to engage with others, to plan and to persist — these are the attributes that get people in the door and on the job, and lead to productive lives. Cognitive and character skills work together as dynamic complements; they are inseparable. Skills beget skills. More motivated children learn more. Those who are more informed usually make wiser decisions.

These established findings should lead to a major reorientation of policies for human development. Because skill begets skill, the opportunity for education should begin at birth — and not depend on the accident of birth.

The family into which a child is born plays a powerful role in determining lifetime opportunities. This is hardly news, but it bears repeating: some kids win the lottery at birth, far too many don't — and most people have a hard time catching up over the rest of their lives. Children raised in

Figure C.7.: Screenshot of the WordFreak annotation tool.

C.8. Analec

Task 1 – Analec can be installed by means of a one-click installation (JAR executable). Note: Analec is only available in French language.

Task 2 – Primary data can be imported to Analec via the "Documents > Importer du texte brute" menu. A dialog allows the user to select a *.txt file and to set the encoding ("Codage des charatéres") of the document. The text is then displayed in the main window, with line breaks after each paragraph.

Task 3 – Annotation schemes can be created inside the tool "Structure > Gestion de la structure". The scheme can be defined in a window that is separated in three different areas. These areas reflect the basic components that can be used to create schemes: units, relations, schemas [205]. In each area, new annotation

[205] Note: These are the same components that are defined in the Glozz (cf. section C.3) anno-

items can be created by means of a hierarchical file tree. The hierarchy can be interpreted as follows:

Root node "TYPES" node (default node, cannot be changed)

1st level A node on this level describes an annotation scheme (note: it is possible to define several schemes)

2nd level A node on this level describes a level of annotation

3rd level A node on this level describes an item on one of the levels of annotation

Right clicking on one of the nodes allows the user to add further child nodes or to delete the node. *Schemas* can be used to group together annotations to larger clusters (as described in the Glozz manual). As this function is not needed to achieve the tasks described in this evaluation study, no annotation items will be defined in the "Schémas" area. The window can be closed after the scheme has been defined.

Task 4 – In order to display the previously defined annotation scheme, the user has to switch into one of three annotation modes: "unit annotation", "relation annotation", or "schema annotation". The mode for "unit annotation", which is needed for Tasks 3.1 - 3.4, is activated via the "Unités > Gestion des unités" menu.

Once the "unit annotation" mode is activated, a number of form fields and buttons appears on the top of the main window. First, an annotation scheme has to be selected from a drop-down list. The annotation scheme (with its separate levels) is displayed on the bottom of the main window once an anchor has been created (cf. Task 5). The different levels of annotation are displayed in the same order as defined in the annotation scheme GUI (cf. Task 3). The items of each annotation level can be accessed via a drop-down list.

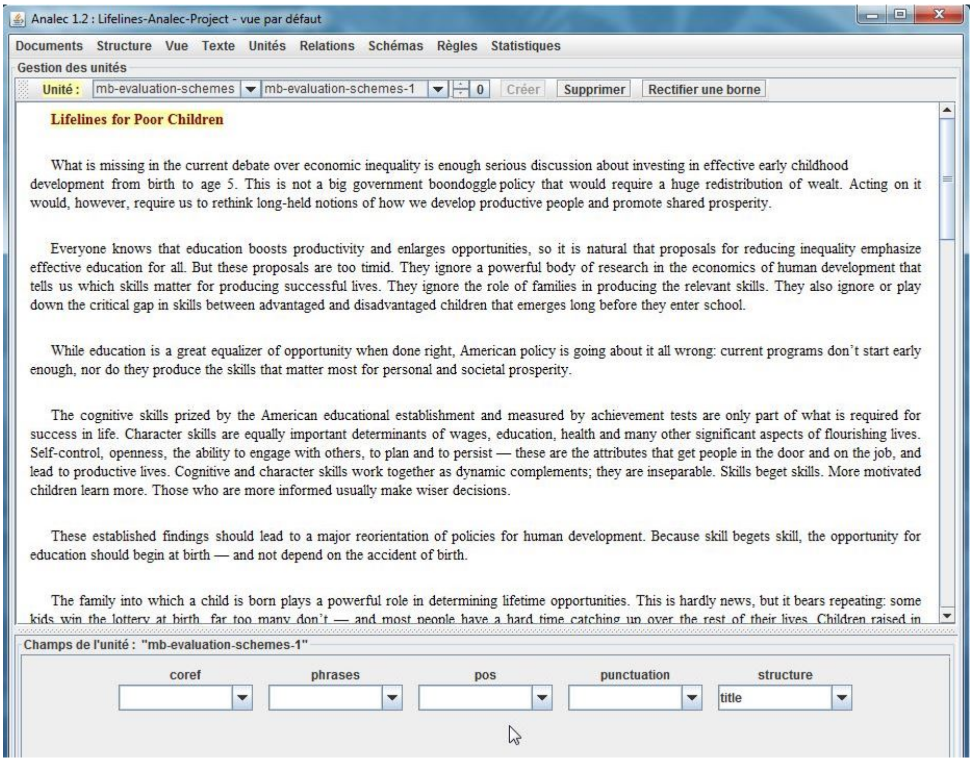
Task 5 – As described in Task 4, one of three annotation modes has to be activated before the annotation process can be started. Once a mode and an annotation scheme have been selected, the user has to select a span of text from the primary data by means of click-drag-release (for multiple words) or by double clicking on a word. After this selection, the creation of the actual anchor has to be confirmed by clicking on the "Créer" button on top of the main window. After this step, the annotation scheme appears on the bottom of the main window. The user may select values for one or more levels of annotation from a drop-down list. There is no need to confirm the annotation; once the user selects another span of text, or just clicks on an empty space, the annotation scheme disappears and the selected annotation values are saved automatically. Each anchor is assigned an ID that can be selected from a drop-down list on top of the main window ("identifiant"). Anchors (and associated annotations) can be deleted by selecting the anchor and by clicking the "Supprimer" button. The scope of an existing anchor can be adjusted by clicking the "Rectifier une borne" button.

tation model. There is also an import / export function for documents annotated in Glozz format.

Before coreference can be annotated, the antecedents and referents (defined as items in the units scheme) have to be annotated as single anchors. In order to annotate coreference, the user has to switch into the "relation annotation" mode (coreference has been defined as a relation). After two anchors ("Elément 1" and "Elément 2") have been selected, a relation ("Relation") can be created: the coreference scheme is displayed at the bottom of the main window and a relation type can be selected from a drop-down list.

Task 6 – Clicking on an anchor shows the associated span of text with a default colored background. The color for each annotation item can be specified in "Vue > Gestion de la vue > Coloriage d'éléments". Previously applied annotations to that anchor are shown in the annotation scheme window at the bottom of the main window. There are no explicit visualizations for coreference relations.

Summary – Analec was primarily designed for manual annotation, i.e. it has no automatic pre-processing or annotation features. The annotation model is the same as described for Glozz (units, relations, schemes). Analec provides statistical analysis functionality as well as sophisticated visualizations for the analysis of the annotated data (e.g. "reference chains").



Unité : mb-evaluation-schemes mb-evaluation-schemes-1 0

Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealt. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

The cognitive skills prized by the American educational establishment and measured by achievement tests are only part of what is required for success in life. Character skills are equally important determinants of wages, education, health and many other significant aspects of flourishing lives. Self-control, openness, the ability to engage with others, to plan and to persist — these are the attributes that get people in the door and on the job, and lead to productive lives. Cognitive and character skills work together as dynamic complements; they are inseparable. Skills beget skills. More motivated children learn more. Those who are more informed usually make wiser decisions.

These established findings should lead to a major reorientation of policies for human development. Because skill begets skill, the opportunity for education should begin at birth — and not depend on the accident of birth.

The family into which a child is born plays a powerful role in determining lifetime opportunities. This is hardly news, but it bears repeating: some kids win the lottery at birth, far too many don't — and most people have a hard time catching up over the rest of their lives. Children raised in

coref	phrases	pos	punctuation	structure
				title

Figure C.8.: Screenshot of the Analec annotation tool.

C.9. WebAnno

Task 1 – WebAnno is a web-based tool that requires an Apache Tomcat web server to host the application, and a MySQL database to store the data. There is also an all-in-one version available for download, which does not require the user to set up a server or a database. While the developers suggest to use this version for testing only rather than for productive use in a real annotation scenario, it seems appropriate for the single-user tasks of this evaluation study. The all-in-one-version has to be executed from the command line, as described in the online manual[206]. The tool can be accessed via the localhost on port 8080 (Default account: user = admin / password = admin).

Task 2 – On the start screen, WebAnno provides a number of options to choose from: "Annotation", "Curation ", "Correction" (at the moment of evaluation still under development), "Projects", "Monitoring", "Manage users", and "Crowd-Source" (at the moment of evaluation still under development). An online manual explains every option. In order to be able to import primary data to the tool a project has to be created via the "Projects" menu item. The user is required to set a name for the project, which will be displayed in the list of projects. Documents can be added to a project by clicking on the project name. Once a project has been selected, a second window with a number of tabs appears. The user needs to click on the "Documents" tab, which opens an "Import new documents" dialog. During this dialog, a file format for the primary data can be specified. Once the document is imported (confirm action with the "Import document" button), its name is displayed in the documents list. In order to display the primary data during annotation mode, the user has to jump back to the main navigation window, and then click on "Annotation". An import dialog for projects and documents opens, where the previously imported document can be selected. Clicking on the "Open" button loads the primary data into the annotation window (this may take some seconds). The primary data is displayed in a similar fashion as in Brat (line numbers, line breaks). In "Settings" the number of lines per "page" can be adjusted (default: 10). There are also controls for navigating the "pages" ("first", "previous", "go to page", "next", "last") and documents (if there are multiple documents in one project).

In WebAnno it is also possible to define different users inside the tool. Users can be assigned different roles / rights (user vs. admin). The "plain users" only see the annotation option when logged in while the admin user has access to all the options. The admin may also monitor the annotation progress of different users on different projects. It is even possible to monitor progress on a particular level of annotation. The curation mode allows the user to compare the annotations of different annotators, and to merge them.

[206] Cf. `https://code.google.com/p/webanno/wiki/WebAnnoStandalone`

Task 3 – Annotations schemes (tag sets) are created in the "Projects" window, via the tab "Tag sets". The user can create tag sets on six different layers which behave differently in terms of anchor creation:

- The "named entity" layer can have anchors that span several tokens within one sentence
- The "lemma" layer can have anchors that span a single token
- The "pos" layer can have anchors that span one token; POS anchors can be connected via a relation that is defined on the "dependency" layer (only within the same sentence)
- The "coreference type" layer can have anchors that span one or more tokens; the type anchors can be connected via a relation that is defined on the "coreference" layer

WebAnno provides default tag sets for each of the six layers. As only one tag set is allowed per layer, the user has to delete the existing tag sets in order to create new tag sets. It was not possible to create all tag sets required for the evaluation task at once, as some annotation levels require the same "layer". Tag sets may also be imported or exported in JSON format. Once a new tag set has been created (parameters that need to be defined: "name", "language", "layer", and an optional "description") new tags can be added via a form (parameters for tag: "name" and an optional "description").

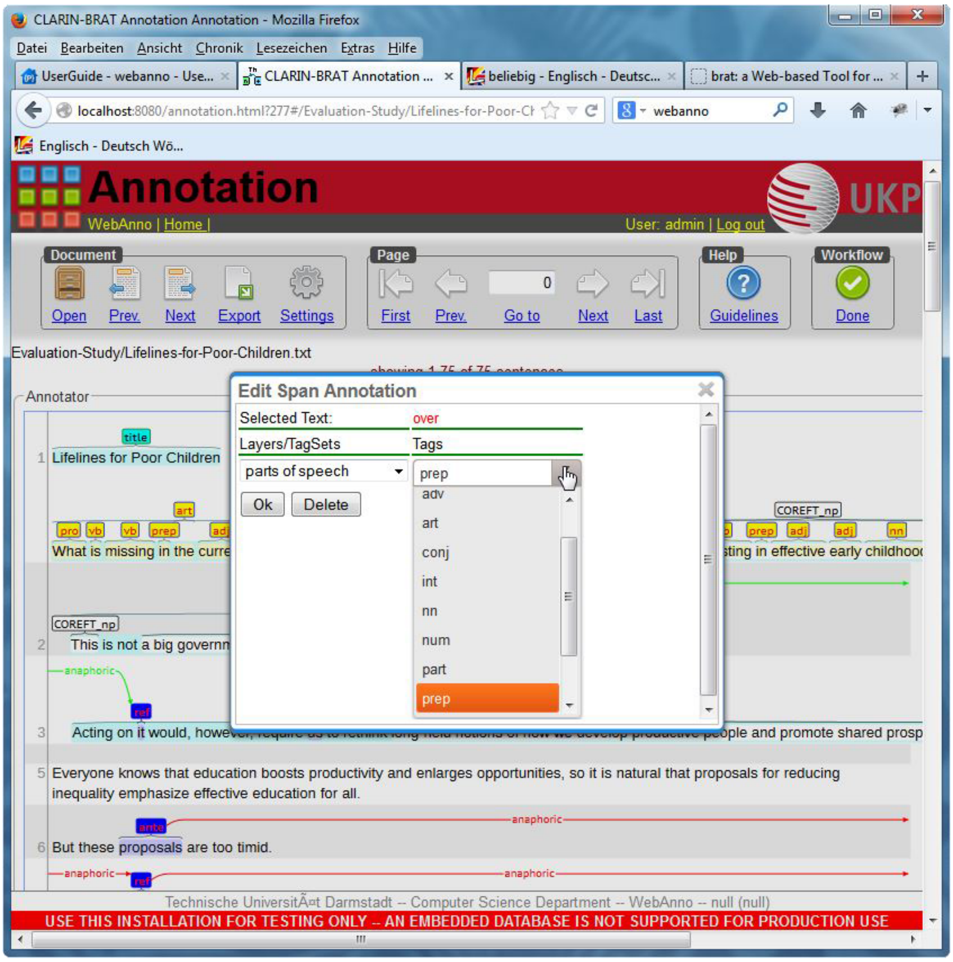
Task 4 – The annotation scheme can only be displayed passively during the annotation process, i.e. it is not possible to edit the scheme during the annotation process. To modify the scheme, the user has to switch from the "Annotation" window to the "Projects" window. The scheme and the corresponding items are only displayed during the annotation process if an anchor is selected, or if the user clicks on an existing anchor. The annotation scheme opens in a separate window, and closes once the annotation has been applied. In the "Settings" menu, the user can specify which annotation schemes are displayed in the annotation window (annotations that have already been applied on this level of annotation are also displayed in the primary data).

Task 5 – In order to annotate, the user has to select an anchor in the primary text via click-drag-release (word anchors may also be selected by double clicking the word). When releasing the mouse, an annotation window pops up which contains two drop-down menus: the first menu contains all the annotation levels that were defined (and that were activated in the "Settings" menu), the second menu contains the annotation items (=tags) of the selected annotation level. The annotation window also displays the selected anchor as a textual string. An annotation item may be applied by selecting a value from the drop-down menu, and by clicking on the "ok" button to confirm the action. By clicking on an existing annotation, the annotation window pops up and allows the user to modify the tag value or to delete both, anchor and annotation.

Relational annotations are realized by means of drag-and-drop. When the user clicks on an anchor that was defined on the layer "coreference type" and holds the mouse button down, an arrow head appears. The arrow head can be dragged to another anchor. Dropping the arrow head on another anchor creates a visible arrow that points from one anchor to the other. The value of the relation can be selected from the annotation pop-up window.

Task 6 – WebAnno makes use of the visualization front end of Brat, i.e. the two tools display annotations in a similar fashion: The annotations are displayed in the primary data as a label in a predefined color. The labels are attached to an anchor region by means of a rotated curly bracket that spans the whole anchor. The label text is the name of the annotation item, as defined in the scheme. Hovering over an existing annotation opens a small window that displays the anchor as a textual string as well as the annotation value and the ID of the annotation. Relations are visualized as a directed arrow. Each relation is displayed in a distinct color. The arrow has a label with the value of the relation and its ID. It also displays the two related entities and the type of relation in textual form. If a relation spans multiple lines, the arrow spans till the end of one line, and starts anew in the next line (this is repeated for multiple lines). Via the "Settings" menu, it is possible to show or hide annotations from the different annotation levels.

Summary – WebAnno is a modern, web-based tool that extends the functionality of the Brat annotation tool. The tool utilizes a client-server architecture and can be accessed via a web browser. All data is stored on a central database server. WebAnno is different from most of the other tools, as it provides a number of additional functions that are useful for collaborative annotation projects. These functions include the option to define annotation guidelines that can be displayed during the annotation process, or to manage different roles / annotators. Also, the annotation workflow can be monitored and managed, i.e. annotations can be submitted for curation or correction. WebAnno is one of the few tools that actually implements an intuitive multi-annotator workflow. Different users can log into the annotation tool and annotate the document on different layers. An admin may then compare the annotations, or merge and correct them. Another distinctive feature is the option to export annotation tasks to a crowdsourcing platform (at the time of evaluation this feature did not seem to work).



Evaluation-Study/Lifelines-for-Poor-Children.txt

Selected Text: over

Layers/TagSets: parts of speech

Tags: prep

1 Lifelines for Poor Children

What is missing in the curre

...sting in effective early childhood

2 This is not a big governn

3 Acting on it would, howev... ...ple and promote shared prosp

5 Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all.

6 But these proposals are too timid.

Technische UniversitÃ¤t Darmstadt -- Computer Science Department -- WebAnno -- null (null)

USE THIS INSTALLATION FOR TESTING ONLY -- AN EMBEDDED DATABASE IS NOT SUPPORTED FOR PRODUCTION USE

Figure C.9.: Screenshot of the WebAnno annotation tool.

C.10. Knowtator

Task 1 – As Knowtator is a plugin for the knowledge modeling framework Protégé, the user first of all has to download the Protégé installer. Next, the Knowtator plugin files (in *.zip format) need to be downloaded and extracted to the plugins directory of the Protégé installation. In order to activate Knowtator, a new project needs to be created and saved in Protégé – after that, the Knowtator plugin can be activated via "Project > Configure > Tab Widgets", check box "Knowtator". A new tab "Knowtator" as well as a menu item "Knowtator" appear in the main navigation, and signal the successful setup of the Knowtator annotation tool[207].

[207] There is a detailed installation tutorial on the Knowtator website (`http://knowtator.sourceforge.net/install.shtml`) that describes every step for the setup and configuration in great detail.

Task 2 – Primary data can be imported to an existing Knowtator project by clicking on the folder icon that is positioned on the right symbol bar (tooltip: "Open text source collection"). The dialog can only import folders, no single *.txt documents – therefore the exemplary evaluation text file needs to be put in a folder, which is then imported via the "open text" dialog. The font size of the primary data (as well as the font size of menus etc.) can be modified via "Window > Increase / decrease font size". In Protégé / Knowtator it is also possible to define different annotators and annotation teams that can be assigned to an annotation. Inter-annotator agreement may be calculated inside the tool.

Task 3 – Being a plugin of the Protégé platform, Knowtator builds on Protégé's "knowledge" model: The basic elements of this model are *classes* (= objects from the domain) and *slots* (attributes for classes). Classes can be created in the tab "Classes". The corresponding "class browser" allows the user to create, modify or delete classes in a hierarchical file tree view. A new class is added as a child node of the selected parent node. The hierarchy of nodes (=classes) in the tree can be modified by means of drag-and-drop. Clicking on a node also opens a context menu that allows the user to specify the "name" and "role" of the respective class, as well as to generate optional "slots". In order to implement coreference annotation, the main class "coreference" gets two subclasses "antecedent" and "referent" as well as a slot titled "coreference-relation". The slot (which needs to be of type "instance") can take a number of "allowed classes", which may later be part of a coreference relation annotation. By setting the cardinality to "multiple", it is possible to relate several referents to one antecedent.

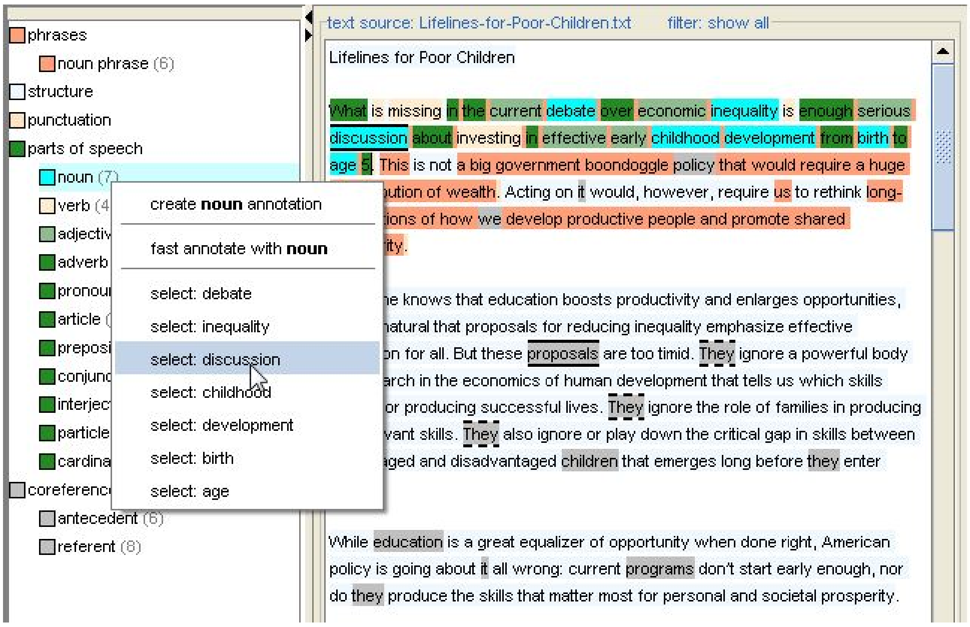
Task 4 – The annotation scheme can be viewed and edited during the annotation process, by switching to the "Classes" tab. Every class needs to be activated manually for the Knowtator plugin via the menu items "Knowtator > Configure". All previously defined annotations schemes can be imported in the field "root classes" (child nodes are automatically imported if the parent node was added). It is also possible to modify the order of annotation schemes and items. After this step, the "classes" appear in the left pane of the Knowtator main window.

Task 5 – In order to apply an annotation, the user is required to define an anchor. This is either achieved by means of click-drag-release for larger scopes, or by double clicking on a single word. Clicking on an annotation item from the scheme opens a context menu that asks the user if he wants to create an annotation for the selected anchor. There is also an option for "fast annotation", which selects the annotation item and automatically applies the value to every anchor scope the user defines. The "fast annotation mode" can be ended by clicking on the "quit" button that appears under the tabs while being in "fast annotation mode". Next to the quit button are previously selected "fast annotation values", which enable quick switching of annotation values in "fast annotation mode".

Clicking on an item from the annotation scheme also shows the anchors of previous annotations, providing a quick overview of already existing annotations for the currently selected value. The number of existing annotations for each item is shown as an integer value in the scheme. Annotations may be deleted by clicking on an existing anchor, and by clicking the "delete" button in the right pane. It is also possible to modify the scope of an anchor with arrow icons. If there are multiple annotations for one anchor, a click on the anchor opens a context menu that lists all parallel annotations. Coreference annotation is realized by adding a coreference-relation (in the right pane) to a previously created antecedent and a referent anchor (all available anchors are shown in a list).

Task 6 – It is possible to define distinctive colors for a layer of annotation, or for each item on that layer. Colors may be specified by means of RGB values, or by selecting color names from the predefined color palette (e.g. "salmon"). Different layers or even single items of annotation can be shown and hidden by means of the "filter" mechanic. Knowtator allows the user to define tailored filters for the visualization of annotation. By default, there are two filters available: show all / hide all.

Summary – Knowtator is designed as a plugin for the knowledge modeling framework Protégé that can be used as a generic text annotation tool. Knowtator also relies on the conceptual model of Protégé, which is reflected in schema design. The "fast annotation mode" is useful to speed up the annotation process for identical annotation values. Knowtator also allows the user to manage annotator roles and to calculate inter-annotator agreement.



Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge
ution of wealth. Acting on it would, however, require us to rethink long-
ions of how we develop productive people and promote shared
ity.

ne knows that education boosts productivity and enlarges opportunities,
natural that proposals for reducing inequality emphasize effective
on for all. But these proposals are too timid. They ignore a powerful body
arch in the economics of human development that tells us which skills
or producing successful lives. They ignore the role of families in producing
vant skills. They also ignore or play down the critical gap in skills between
aged and disadvantaged children that emerges long before they enter

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

Figure C.10.: Screenshot of the Knowtator annotation tool.

C.11. GATE

Task 1 – GATE can be downloaded as an executable file that may be installed with one click. During the installation process the user can optionally install the standard user guide, a detailed developer documentation and the source code of GATE. At the end of the installation the user can generate an "installation script" for future installations.

Task 2 – GATE distinguishes "Applications", "Language Resources" and "Processing Resources". In order to import primary data the user has to right click "Language Resources" and then select "New > GATE Document". In the following import dialog the user has to specify a name as well as the encoding (e.g. UTF-8). In the field "sourceURL" the path to the primary data is specified. After clicking "Ok", the imported document can be accessed (via double click) as a child-node of the "Language Resources" node in the left pane.

Tasks 3-5 – GATE has two options for creating an annotation scheme: The first option is to create an annotation scheme in XML Schema syntax by using an external XML editor, and import the scheme via "Language Resources > right click > New > Annotation Schema". The second option is to create an annotation scheme inside GATE in an ad hoc manner, during the annotation process. A new level of annotation can be created in the tab "Annotation Sets", by specifying a name for the level in the empty form field and by clicking on the "New" button in the bottom left corner. By default, GATE creates a level "Original markups", which contains existing markup of the primary data. If the primary data is just plain text, only the "paragraphs" are annotated in "Original markups".

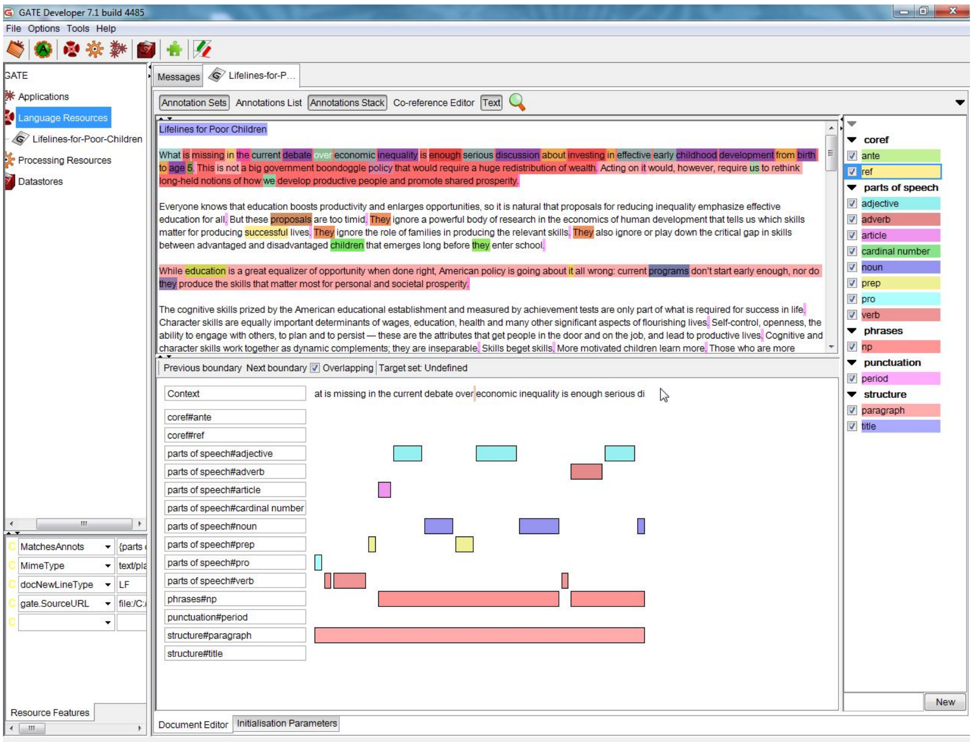
Selecting an anchor scope and hovering over the selection opens a pop-up menu which allows the user to either define a new annotation value, or to select an existing value from a drop-down list. The annotation value is applied by closing the pop-up window (x). The newly created annotation value is displayed on the right "Annotation Sets" pane. It has a default color that can be changed by right clicking the item and by clicking "Change color". Right clicking an item and choosing "Delete" deletes the item from the annotation scheme. All items can be set visible or hidden by means of a checkbox.

Hovering over an annotated anchor opens the pop-up annotation window. The user may modify the scope of the anchor by means of small arrows. It is also possible to modify the annotation value, or to delete the annotation for the selected anchor. Hovering over an anchor with multiple annotations opens a context menu with all annotation values, from which the user can choose which one he wants to edit.

Coreference annotation is realized via the "Co-reference editor" tab. GATE allows the user to create chains of coreference relations that may include multiple annotated elements. The chains automatically get the name of the longest anchor that is part of the chain. Like annotation items, chains are displayed in different colors, and can be set to visible / hidden.

Task 6 – As was described before, GATE automatically applies colors to each annotation item and allows the user to show / hide items in the primary data via check boxes. Besides the color highlight visualization in the primary data, GATE provides two alternative views on annotations: The "Annotations List" tab displays the annotations in a structured, tabular view, with the following parameters: "Type" (annotation item), "Set" (annotation level), "Start" and "End" of anchor scope, "Id" and additional "Features" of the annotation. The second view, the "Annotations Stack", displays annotations for a selected span of primary data in a parallel layer fashion. The stack view is convenient for visualizing parallel, overlapping annotations.

Summary – GATE is a complex framework for text annotation and analysis. GATE's focus lies on automatic annotation and processing functions (comparable to UIMA), but it also has a sophisticated, manual annotation component. Although GATE can be expanded with a great collection of available plugins, the default interface is relatively simple and does not distract the manual annotator. The real strength of the framework, however, lies in a combination of manual and automatic annotation. Among the automatic tools are *tokenizers*, *sentence splitters*, *gazetteers*, *transducers*, *taggers*, etc.



Lifelines for Poor Children

What is missing in the current debate over economic inequality is enough serious discussion about investing in effective early childhood development from birth to age 5. This is not a big government boondoggle policy that would require a huge redistribution of wealth. Acting on it would, however, require us to rethink long-held notions of how we develop productive people and promote shared prosperity.

Everyone knows that education boosts productivity and enlarges opportunities, so it is natural that proposals for reducing inequality emphasize effective education for all. But these proposals are too timid. They ignore a powerful body of research in the economics of human development that tells us which skills matter for producing successful lives. They ignore the role of families in producing the relevant skills. They also ignore or play down the critical gap in skills between advantaged and disadvantaged children that emerges long before they enter school.

While education is a great equalizer of opportunity when done right, American policy is going about it all wrong: current programs don't start early enough, nor do they produce the skills that matter most for personal and societal prosperity.

The cognitive skills prized by the American educational establishment and measured by achievement tests are only part of what is required for success in life. Character skills are equally important determinants of wages, education, health and many other significant aspects of flourishing lives. Self-control, openness, the ability to engage with others, to plan and to persist — these are the attributes that get people in the door and on the job, and lead to productive lives. Cognitive and character skills work together as dynamic complements; they are inseparable. Skills beget skills. More motivated children learn more. Those who are more

MatchesAnnots	{parts
MimeType	text/pla
docNewLineType	LF
gate.SourceURL	file:/C:

Context: at is missing in the current debate over economic inequality is enough serious di

coref#ante
coref#ref
parts of speech#adjective
parts of speech#adverb
parts of speech#article
parts of speech#cardinal number
parts of speech#noun
parts of speech#prep
parts of speech#pro
parts of speech#verb
phrases#np
punctuation#period
structure#paragraph
structure#title

Figure C.11.: Screenshot of the GATE annotation tool.

D. Usability problems

D.1. Dexter

Source `http://www.dextercoder.org/index.html`

Date of download October 2, 2013

Date of evaluation October 2, 2013

Number of usability problems 14

Number of specific strengths 3

Usability problems

ID	Problem	Description	Violated heuristics										Severity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
DEX01	**Forced pre-processing of primary data**	Although pre-processing can facilitate the annotation process in general, it should be optional, because the parameters that have to be set are oftentimes complex.					1	1					2	Primary data
DEX02	**Multiple tools**	The conversion function should be included in the annotation tool. The installation of two separate tools (for conversion and annotation) creates unnecessary complexity.					1	1					2	Primary data
DEX03	**Conversion tool complexity**	The conversion tool offers too many parameters. Although there are tooltips for each of the parameters, the user might be overwhelmed by the abundance of options.					1	1		1			3	Primary data
DEX04	**Parsing errors**	The conversion tool seems to have problems when parsing documents that contain double quotation marks. It is not clear to the user how these errors can be avoided.			1		1				1		4	Primary data
DEX05	**Two-step "open file" dialog**	When starting Dexter Code anew the user needs to open both, the primary document and the coding file, with two different "open"-dialogs. (*Other tools allow the user to save both files in one "project" etc.)					1	1					1	Primary data
DEX06	**No easy manipulation of items in the annotation scheme**	It is not possible to create, delete or edit an item in the annotation scheme via a shortcut or a right-click context menu. The user has to navigate to the top menu bar every time he wants to edit or delete an item. The creation of a nested, subordinated item would also be facilitated if it would be possible to click directly in the scheme and create a new item.			1			1	1				2	Annotation scheme
DEX07	**No customized color palette for annotation items**	It is not possible to define custom colors for annotation items – the user has to choose from a total of 28 predefined colors.			1			1					1	Annotation scheme
DEX08	**Fixed order of items in the annotation scheme**	The order of items in the annotation scheme is alphabetically (for each hierarchical level) and cannot be changed. Ideally, it would be possible to change the order of elements via drag-and-drop.							1				3	Annotation scheme
DEX09	**Obligatory display of all items in the annotation scheme at the same time**	Every item in the annotation scheme is displayed to the user. It might be useful to minimize certain categories and their sub-items from the display, to reduce complexity and distraction during the annotation process.								1			2	Annotation scheme

DEX10	**Insufficient visualization of hierarchical structure in the annotation scheme**	The hierarchical dependencies are only visualized by minor indents. A "file tree"-like visualization with "branches" would be better. The centered text in the "codes" visually conflicts with the left-side indentions.								1			2	Annotation scheme
DEX11	**No custom shortcuts**	Although the availability of predefined shortcuts is a good thing, it is disadvantageous that the shortcuts cannot be customized. This is especially problematic for code items, as they cannot be changed and the shortcuts are automatically assigned running numbers starting, from 1.							1				2	Annotation process
DEX12	**Unclear "delete annotations" function**	When the user clicks on an anchor, he sees all parallel annotations for it in a context menu. By clicking on one of these annotations, he deletes it. The delete function is not indicated by an icon or some other kind of hint.	1				1	1					2	Annotation process
DEX13	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone.			1	1			1				3	Annotation process
DEX14	**Insufficient visualization of parallel annotations in the primary data**	Annotations are visualized with colored (limited color palette) highlights. It is hard to distinguish multiple, parallel levels of annotation by means of overlapping, colored highlights. The purpose of the small column on the right side of the screen is unclear.								1			2	Annotation visualization
14	**Usability problems**		1	0	4	1	6	7	4	4	1	0	2,2	

0	**Installation**
0	**General UI**
5	**Primary data**
5	**Annotation scheme**
3	**Annotation process**
1	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-DEX01	**Shortcuts**	Shortcuts for applying annotations speed up the annotation process significantly.							1					General UI
S-DEX02	**Preview primary data**	After the primary data has been imported to the converter tool, it can be previewed with and without conversion.	1				1							Primary data
S-DEX03	**Annotation scheme creation and modification via GUI**	The annotation scheme can be created and edited directly in the tool, by means of graphical elements and a simple form window.							1					Annotation scheme
3	**Specific strengths**		1	0	0	0	1	0	2	0	0	0		

0	**Installation**
1	**General UI**
1	**Primary data**
1	**Annotation scheme**
0	**Annotation process**
0	**Annotation visualization**

D.2. CATMA

Source `http://www.catma.de/`

Date of download October 7, 2013

Date of evaluation October 7, 2013

Number of usability problems 19

Number of specific strengths 5

Usability problems

ID	Problem	Description	Violated heuristics										Seve-rity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
CAT01	**Refresh button is unclear**	The function of the "refresh" button in the top right corner of the "repository" and "tag manager" modules is not clear.					1	1					1	General UI
CAT02	**Multiple windows**	Each tool / module opens in a new window. This does not feel natural in the domain of web applications. Also, moving and resizing the windows takes rather long (due to the client-server architecture?). If multiple windows are used, the user should be able to lock them on some kind of grid.				1	1			1			2	General UI
CAT03	**Hints disappear to fast**	On some occasions, textual hints on how to proceed with the tool appear in the bottom left corner. This position is not in the focus of the user. Also, the message text disappears really fast. Users generally will not be able to read the hint, but rather have a bad feeling of having missed some important information.										1	2	General UI
CAT04	**Hints are too generic and too small**	There are some ?-icons scattered across the interface. These icons are very small and can be easily overseen. They look like buttons, but cannot be clicked. They reveal rather long, generic hints on how to proceed with the tool. Tooltips should be contextual, on a specific UI element, and explain what the respective element does.										1	2	General UI
CAT05	**Double clicking on a document or a tagset**	Sometimes, double clicking on documents or tagsets does open them, sometimes not (inconsistent behavior). At the same time, there are "Open Markup Collection" / "Open Tagset Library" buttons that do the same job (redundancy).				1							2	General UI
CAT06	**"Page size zoom" function is unclear**	There is no way of adjusting the font size. The "page size zoom"-function implies to zoom the text, but actually adjusts how much text is displayed in the document pane. The wording is misleading.		1			1	1					3	General UI
CAT07	**Create corpus button is unclear**	It is not clear whether it is necessary to create a corpus first, or if a document can be "added" to the repository without being part of a corpus. It is not clear whether a document can be added to a corpus at a later point in time.					1						2	Primary data
CAT08	**Adding documents to corpus is unclear**	It is not clear how documents can be added to an existing corpus. Drag-and-drop of a document from the "documents pane" to the "corpora pane" is not intuitive (especially in the web context).				1	1						2	Primary data

CAT09	**Preview only shows fragment of document**	Once a document has been added, the preview only shows a fragment of the primary data. It ends abruptly in the middle of a word, conveying the impression that something went wrong when importing the document.	1				1						3	Primary data
CAT10	**Annotation scheme hierarchy is unclear**	There seem to be several concepts for creating a hierarchical, multi-level annotation scheme. It is, however, not clear which concepts to choose: "tag library > tagset > tag > property > values"; tag library, tag set and tag (with sub tags) can be used equally to model different levels of annotation with respective items.		1			1	1					3	Annotation scheme
CAT11	**Wording for creating an annotation scheme is unclear / ambiguous**	The difference between "user markup collection" and a "static markup collection" and "tagset library" is not clear / intuitive.		1		1	1	1					3	Annotation scheme
CAT12	**Relating a tagset to a document is unclear**	It is not clear how tagsets can be related to a document. Drag-and-drop is neither intuitive (especially in the web context) nor practical, as multiple windows have to be opened (cf. "multiple windows"-problem).				1	1						3	Annotation scheme
CAT13	**Annotation process is not intuitive**	Although described in the manual, it is not intuitive to select an anchor in the left "document pane" and then apply an annotation from the tagset by clicking on its color symbol.							1				2	Annotation process
CAT14	**No shortcuts for applying annotation values**	There are no shortcuts for applying a tag to a selected anchor. This is very impractical, as the user has to click on a tag after each anchor selection.							1				2	Annotation process
CAT15	**Double clicking on a word selects whitespaces**	It is cumbersome to select small anchors, such as short words or punctuation, because the text is displayed in a rather small, default font size. Double clicking on a word does not only select the word, but also the subsequent whitespace, which creates erroneous anchors.				1	1		1				2	Annotation process
CAT16	**Deleting annotations is impractical**	If an anchor is clicked, the parallel annotations are displayed in an "inspector-like" window on the right, named "Writable Markup Collection". Tags may be removed inside this window. The same applies for setting the values for a property. This is not intuitive (a context menu in the document would be better).							1	1			3	Annotation process
CAT17	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone.			1	1			1				3	Annotation process
CAT18	**Show / hide annotations function is not accessible**	It is not intuitive to distinguish between "active tagsets" and "active markup collections". The show / hide function for items from the annotation scheme is only available in the "active markup collections" tab, which is not displayed by default.					1	1					3	Annotation visualization

CAT19	**Collapsed annotation underlines**	Underlines collapse into each other - they should rather form a straight line for each level of annotation.								1			2	Annotation visualization
19	**Usability problems**		1	3	1	7	11	5	5	3	0	2	2,4	

0	**Installation**
6	**General UI**
3	**Primary data**
3	**Annotation scheme**
5	**Annotation process**
2	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-CAT01	**Ready-to-use web service**	The tool does not require the user to install or setup anything at all, as it is available as a web service. All data is stored on the CATMA server. The tool can be accessed via a web browser and a Google account.					1							Installation
S-CAT02	**Meta data for primary data**	It is possible to define title, author, description and publisher for the primary data document.	1					1						Primary data
S-CAT03	**Annotation scheme creation and modification via GUI**	The annotation scheme can be created and edited directly in the tool, by means of graphical elements and a simple form window.							1					Annotation scheme
S-CAT04	**Show and hide annotations on different levels of detail**	It is possible to show and hide whole levels of annotations as well as single annotation items.								1				Annotation visualization
S-CAT05	**Visualization of annotations as underlines**	Underlining the anchors in the color of the respective tag is a good way to visualize multiple, parallel annotations.								1				Annotation visualization
5	**Specific strengths**		1	0	0	0	1	1	1	2	0	0		

1	**Installation**
0	**General UI**
1	**Primary data**
1	**Annotation scheme**
0	**Annotation process**
2	**Annotation visualization**

D.3. Glozz

Source `http://www.glozz.org/`

Date of download October 8, 2013

Date of evaluation October 10, 2013

Number of usability problems 17

Number of specific strengths 9

Usability problems

ID	Problem	Description	Violated heuristics										Seve-rity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
GLO01	**"Load last job" is unclear**	"Load last job" does not sound like reloading the recent files; it is unclear what the function actually does. The manual does not describe the function sufficiently.					1	1					3	General UI
GLO02	**Cumbersome import process**	Importing the primary data is very cumbersome. Although the "Import button" is clearly visible, the following import dialog is rather irritating: The user not only has to import the primary data (*.txt), but also needs to specify the paths to an output file (*.ac) and an annotation file (*.aa). It is not clear what these files do, whether they already exist, or whether they have to be created manually. It is also not clear where to store the files.		1			1		1			1	4	Primary data
GLO03	**No feedback after import**	After the *.ac and *.aa files have been created, the import dialog closes, but the imported text is not displayed. The user has to click on "file > open", to start a dialog that requires to browse the respective *.ac and *.aa files. This is redundant - after having created those files in the previous step, it would be more comfortable to display the document directly, and not having to define the same path again to achieve this effect.	1				1		1	1		1	4	Primary data
GLO04	**Annotation scheme cannot be created / edited inside the tool**	The annotation scheme needs to be created outside of the tool, in a code editor. If the user changes the scheme outside of the tool, he has to "reload" the scheme.					1		1				4	Annotation scheme
GLO05	**Annotation scheme must be defined as XML**	The annotation model (= scheme) is defined in XML syntax. This can be cumbersome for annotators without knowledge about markup languages (e.g. type names without whitespace, etc.).					1		1				2	Annotation scheme
GLO06	**Some "type" names are not working**	There seem to be problems with the type names "title" and "paragraph" - they are displayed in the tool, but cannot be used as valid annotations.		1			1				1		4	Annotation scheme
GLO07	**Wording for creating an annotation scheme is not intuitive**	The tool makes use of an annotation model that relies on units (= anchors), relations (= relations), and schemas (= groups of anchors, relations and other schemas). An annotation tool should be theory-neutral and use intuitive concepts.		1		1							2	Annotation scheme
GLO08	**No hierarchy in annotation scheme**	It is not possible to implement a hierarchy of elements and subelements in the annotation scheme, i.e. all elements from different annotation levels are displayed in the same pane, with no visual demarcation.								1			2	Annotation scheme

GLO09	**Display of all items in the annotation scheme at the same time**	Any item that is in the annotation scheme is displayed to the user. It might be useful to minimize certain categories and their sub-items from the display, to reduce complexity and distraction during the annotation process.								1			3	Annotation scheme
GLO10	**Double clicking on a word does not select it per default**	This is a conventionalized shortcut that can speed up the annotation process. Here, the shortcut is disabled by default, and has to be explicitly turned on in the preferences.				1			1				2	Annotation process
GLO11	**Switching to different annotation modes is impractical**	Whenever the user wants to annotate, he has to select the *annotation* mode. Whenever he wants to change an annotation, he has to switch to *edit* mode. This is impractical - annotation modes should rather be activated automatically whenever the user selects an item from the respective annotation scheme. The *change* / *delete* mode could be toggled in another way (e.g. by a right click).					1						4	Annotation process
GLO12	**Font size of the primary data is too small and cannot be changed**	The font size of the primary data is too small and cannot be enlarged. This makes the annotation of small anchors (single character and short words) a very cumbersome task.					1						2	Primary data
GLO13	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone			1	1			1				3	Annotation process
GLO14	**Display of primary data is broken**	At some point, after a couple of annotations were added, and the document was "reloaded", the display of the primary data was broken, i.e. the tool displayed different font sizes, arbitrary linebreaks after single characters, words and phrases, as well as centered text.			1						1		4	Primary data
GLO15	**Styling of the annotation scheme as an extra step**	The items of the annotation scheme need to be styled inside the tool, as an extra step. If the user wants to display an item in a specific color, he may define it in the "Style editor". This is not practical, as the user typically wants every item in a distinctive color. Items should have different color by default. Also, the "individual stylesheet" must be selected in the "color mode" menu.							1	1			4	Annotation visualization
GLO16	**Show / hide annotations is not accessible / intuitive**	Although it is nice to be able to show / hide single units, single annotation items and whole groups of items, it is disadvantageous to scatter these functions among three different places in the interface. Show / hide groups is in a menu "Groups" that only has one menu item: "Manage visibility".						1	1	1			4	Annotation visualization

GLO17	**Insufficient visualization of parallel annotations in the primary data**	It is hard to distinguish multiple, parallel levels of annotation by means of overlapping, colored highlights.								1			2	Annotation visualization
17	**Usability problems**		1	3	2	3	7	1	8	5	2	2	3,2	

0	**Installation**
1	**General UI**
4	**Primary data**
6	**Annotation scheme**
3	**Annotation process**
3	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-GLO01	**Sandbox with examples**	Predefined examples that can be loaded are nice to get a feel for what can be done with the tool.					1					1		General UI
S-GLO02	**Visual support when defining anchors**	There are little flags when clicking in the primary data (to create and anchor scorpe) that say "begin" and "end" - this is good user feedback.					1	1						Annotation process
S-GLO03	**Visual support when editing anchors**	Once the user is in "editing" mode, he not only can delete annotations, but also edit the anchor of existing ones. There are visual aids in the form of small circles. These circles can be dragged like handles, and allow the user to adjust the scope of an anchor.					1	1						Annotation process
S-GLO04	**Relations can be created with two clicks and a visual aid**	To create a relation, the user only has to select the two anchors he wants to relate. There is a visual aid in the form of an arrow that helps to "draw" the relation.				1	1		1					Annotation process
S-GLO05	**Delete annotations with DEL key**	Deleting annotations with the DEL key complies to standardized behavior in other systems and can speed up the annotation process.				1			1					Annotation process
S-GLO06	**Show and hide annotations on different levels of detail**	It is possible to show and hide whole levels of annotations as well as single annotation items.								1				Annotation visualization
S-GLO07	**Macro-view and positional syncing**	A thumbnail-version of the document (macro-view) allows the user to quickly navigate through long documents. The whole text can be accessed via a scrollbar or by clicking in the macro-view of the document on the left side. If the user moves the mouse cursor somewhere in the document, the position is highlighted in the macro-view (and vice versa).							1					Annotation visualization
S-GLO08	**Tabular view of annotations**	Annotations are not only displayed as colored highlights in the primary document, but also in a tabular view, containing information about the annotation's value, its scope and its ID.							1					Annotation visualization

S-GLO09	**Good visualization for annotations that span several rows**	If an anchor spans e.g. two rows in the primary data, the area is highlighted in one color, with a visual demarcation to nearby anchors.								1				Annotation visualization
9	**Specific strengths**		0	0	0	0	4	2	4	2	0	1		

0	**Installation**
1	**General UI**
0	**Primary data**
0	**Annotation scheme**
4	**Annotation process**
4	**Annotation visualization**

D.4. UAM Corpus Tool

Source `http://www.wagsoft.com/CorpusTool/`

Date of download October 11, 2013

Date of evaluation October 11, 2013

Number of usability problems 21

Number of specific strengths 3

Usability problems

ID	Problem	Description	Violated heuristics										Seve-rity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
UAM01	**No conventionalized design of UI elements**	Buttons and other UI elements do not conform to established design conventions and look-and-feel guidelines. The user has troubles identifying interaction elements. Examples: (1) The buttons are rendered in light grey and thus look as they are *inactive.* This holds the user from clicking on them. (2) Some buttons, like e.g. the "edit" (layer) button are arranged in a rather irritating way. They convey false functionality, like e.g. belonging to a free text field.	1			1	1	1					4	General UI
UAM02	**Redundant controls**	Many controls are redundant. This does not make it easier for the user to find a specific function, but rather generates overhead by providing too many (similar) options. Example: multiple "help" buttons.				1	1			1			2	General UI
UAM03	**Help function uses advanced language**	The help function uses topics that are not easily understood by new users.		1								1	2	General UI
UAM04	**Overall wording is not intuitive**	The general wording is not easy to understand / not very meaningful. Examples: "(Un-)Incorporate files", "Extend corpus", "System / features", etc.		1			1	1					3	General UI
UAM05	**Importing files is not standardized**	It is not clear at first sight how a text can be imported. The "Project" menu is misleading, as the user would expect something like "File" in this position. Import files actually means "import previous projects". The import mechanism is not clearly visible; the wording "Extend corpus" is not easy to interpret and not quite conventional.				1	1						3	Primary data
UAM06	**"Corpus" vs. "project" is unclear**	It is not entirely clear what is the difference between a "project", that has to be created in the beginning, and a "corpus", to which documents need to be added before they can be annotated.		1			1						2	Primary data
UAM07	**Incorporating text into a corpus is impractical**	After a text has been imported to the tool, it has to be "incorporated" to a corpus before it can be actually annotated.					1	1	1	1			2	Primary data
UAM08	**Imported document can be edited, but changes are not saved**	Once a document has been imported into the tool it can be opened. The user can also edit the text (delete / insert characters). These changes are, however, not saved once the user closes this document "preview". This can be quite irritating, as the user might think he can edit the primary data inside the tool.	1				1						2	Primary data

UAM09	**Too many unexplained options in the assistant for annotation scheme creation**	The wizard for the creation of a new annotation layer contains many options. Some of them are explained, some are not. It is not always clear what an option does, or what are the implications when selecting it.					1	1				1	3	Annotation scheme
UAM10	**Creation of an (empty) annotation layer and creation of actual annotation items as two different steps**	The wizard for creating a new annotation layer does not contain a step which allows the user to add new tags to the layer. It is not clear why the definition of tags requires an additional step (click on "edit" layer).					1		1	1			2	Annotation scheme
UAM11	**Default tags in the annotation scheme are irritating**	The default tags in the annotation scheme are irritating. It is no clear that they are mere examples of a basic scheme.	1				1	1					3	Annotation scheme
UAM12	**Annotation scheme editor uses rather specific concepts / metaphors**	The graphical annotation scheme editor is a good idea, but the specific metaphors *(systems, features, depth, types, graphs, conditions)* are not clear to non-RST linguists.		1			1	1					1	Annotation scheme
UAM13	**Interaction with the graphical scheme editor is not intuitive**	It is not clear how to interact with the graphical scheme immediately.					1	1					4	Annotation scheme
UAM14	**Insufficient visualization of hierarchical structure in the annotation scheme**	Although the tool allows the user to define a hierarchically structured annotation scheme, it only displays one level at a time during the annotation process. The user has to click through different levels of annotation.								1			2	Annotation scheme
UAM15	**Mida's touch problem during the annotation process**	"Mida's touch problem": every click-drag-release action creates a new anchor selection.	1				1		1				3	Annotation process
UAM16	**Applying annotations via double click (in the scheme) is not intuitive**	The application of an annotation value via double click is not intuitive. Furthermore, the nearby save button suggests that it could be used to save an annotation to a selected anchor. Actually, it saves the whole project.					1	1					3	Annotation process
UAM17	**Limited space for displaying the annotation items from the scheme**	There is only limited space for annotation items in the scheme pane. If there are many items, the user may have to scroll down to select it.						1	1				3	Annotation scheme
UAM18	**Handles for editing an anchor scope are too small**	The handles for editing an anchor scope are to small, which leads to many errors.	1			1			1				2	Annotation process
UAM19	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone.			1	1			1				3	Annotation process
UAM20	**Annotations cannot be visually distinguished without clicking on them**	All annotations are visualized with a green underlining. It is not possible to visually distinguish different annotation items from each other without clicking on them. If the user clicks on an annotation, its value is displayed in the bottom annotation pane.							1	1			2	Annotation visualization

UAM21	**Only one layer of annotation can be viewed at a time**	It is only possible to view one annotation layer at a time. For certain annotation scenarios it might be helpful to view more than one layer at the same time.	1					1					2	Annotation visualization
21	**Usability problems**		6	4	1	5	14	10	7	5	0	2	2,5	

0	**Installation**
4	**General UI**
4	**Primary data**
7	**Annotation scheme**
4	**Annotation process**
2	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-UAM01	**Visual annotation scheme editor**	There is a visual annotation scheme creator and editor within the tool that can be accessed at any time during the annotation process.	1						1					Annotation scheme
S-UAM02	**View XML code during the annotation process**	It is possible to show the XML code that is generated during the annotation process. This can be an interesting "feedback" for people who know about XML, and who would like to know what is happening in the background while annotating via the GUI.	1											Annotation visualization
S-UAM03	**Automatic segmentation (anchor creation)**	Automatic segmentation works really good for paragraphs, clauses, NPs and words. This can speed up the actual annotation process significantly. It is also possible to navigate between the segments by using arrow buttons during the annotation process.							1					Annotation process
3	**Specific strengths**		2	0	0	0	0	0	2	0	0	0		

0	**Installation**
0	**General UI**
0	**Primary data**
1	**Annotation scheme**
1	**Annotation process**
1	**Annotation visualization**

D.5. Brat

Source `http://brat.nlplab.org/`

Date of download October 15, 2013

Date of evaluation October 15, 2013

Number of usability problems 24

Number of specific strengths 13

Usability problems

ID	Problem	Description	Violated heuristics										Seve-rity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
BRA01	**Installation requires technical skill (UNIX / server)**	In order to create an individual annotation scheme in Brat, and to be able to tailor the tool to specific needs, it is necessary to host it on a private server - this requires technical skill in UNIX environments and basic knowledge about web servers and Python.		1			1						2	Installation
BRA02	**No save button**	The project is automatically saved, but there is no save button that explicitly allows the user to save a document and its annotations.	1			1							3	General UI
BRA03	**Main navigation is hidden by default**	The main navigation bar on the top of the screen should not be hidden by default, but rather have a function that allows the user to minimize / hide it if desired.	1				1						3	General UI
BRA04	**Backward / forward navigation is confusing**	The function of the two arrows that allow the user to navigate back and forth through a collection of documents is not understood intuively. The arrows rather look like an undo / redo function. Also, the arrows are positioned directly under the browser's navigation buttons; the user might confuse the different controls.				1	1						2	General UI
BRA05	**Wording and structure of controls is not intuitive**	The buttons in the main navigation bar that are labeled with "Collection" and "Data" are not intuitive. It is not clear why "collections" (of documents) are different from "data". "Collections" allow the user to open imported documents. "Data" contains an import feature for documents, but also other functions such as "export", "compare" and "automatic annotation". These functions are arranged in arbitrary order, and some of them are only placeholders, as they are not even implemented yet. Functions like "open document" or "import document" have been conventionalized for numerous tools.		1		1	1	1					4	General UI
BRA06	**Inconsistent use of tooltips**	While there are tooltips for some functions, they are not available for every function, which is inconsistent.				1						1	2	Genreal UI
BRA07	**Error messages disappear to quick**	Error messages appear on the bottom of the screen, but disappear too quick. The little blue button (i) does not convey that its function is to show those messages again. The button cannot be pressed, but only works "on hover".	1								1		2	General UI

BRA08	**Documentation is lacking important information**	Although the manual explains some of the basics of Brat, it does not explain how to create an individual annotation scheme, or how to configurate the tool to one's specific needs. This information is given on the web site, on a page named "configuration.html", which is only accessible via the site map.					1					1	4	General UI
BRA09	**Importing primary data is not intuitive**	The two ways for importing data are both not intuitive: (1) By clicking on the "Data" button, a dialog opens that allows the user to import a document. The user has to specifiy a document name and an ID, and then needs to paste the document content in a form field. The user does not really know what to do here. It is also not clear that the document is stored in the "collection". (2) If a user saves a document directly as a *.txt file in the "data" folder of the Brat installation, he needs to create an empty annotation file (*.ann) with the same file name as the primary data. This could by automized, to prevent errors and misunderstandings.	1				1		1				4	Primary data
BRA10	**Annotation scheme cannot be created / edited inside the tool**	The annotation scheme needs to be created outside of the tool, in a code editor. If the user changes the scheme outside of the tool, he has to refresh the browser.					1		1				3	Annotation scheme
BRA11	**No whitespaces for annotation items allowed**	Although the syntax is not XML-based, annotation items that are defined in the scheme may not contain whitespaces. The error messages state a parsing error.					1				1		3	Annotation scheme
BRA12	**Placeholder for unused annotation types**	There are four basic annotation types: *entities, relations, events* and *attributes*. Even if events and attributes are not used, there has to be a placeholder for them in the annotation scheme (there is a hint in the configuration tutorial).					1						2	Annotation scheme
BRA13	**Annotation scheme is only visible after selecting an anchor**	The annotation scheme can only be viewed after an anchor has been selected.	1		1								1	Annotation scheme
BRA14	**Colors for annotations cannot be defined / edited inside the tool**	Each annotation item has the same color by default. Colors can only be specified outside of the tool, in a "visual.conf" file, as HTML or hex code.						1	1				4	Annotation scheme
BRA15	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone.			1	1			1				3	Annotation process
BRA16	**Delay when applying / deleting an annotation**	There is a significant delay when applying or deleting an annotation.	1						1				3	Annotation process

BRA17	**Annotation errors for nested annotations**	Whenever annotations are nested, an annotation error is displayed in the "on hover" context menu of the respective annotation. It is not clear if or how the error can be eliminated, or which consequences it entails.	1								1		4	Annotation process
BRA18	**Interaction for editing an annotation does not work properly**	The user needs to double click on an existing annotation to open the annotation pane, and to edit the annotation. The double click does not always work at the first try.				1							3	Annotation process
BRA19	**Wording for editing the anchor of an annotation is unclear**	"Move" is not an intuitive wording for editing the scope of an existing anchor (positive aspect: a tooltip explains what "Move" does).		1			1	1					3	Annotation process
BRA20	**Some functions are unclear**	It is not clear what the "Add Frag." function does. It is available when an existing annotation is edited (no tooltip available).		1			1	1					3	Annotation process
BRA21	**Interaction for creating relations is irritating**	Relations between two existing annotations are created by means of click (first annotation) and release (second annotation). While the first annotation is selected at its label, the second annotation has to be selected slightly underneath the label, which is not intuitive.					1		1				3	Annotation process
BRA22	**Large anchor scopes are displayed as a single horizontal line**	Even though the primary data is displayed with line breaks right after its import, lines are collapsed into one long line if an anchor scope spans multiple lines (e.g. paragraph). The user then has to scroll horizontally, to see all of the primary data.					1			1			4	Primary data
BRA23	**Confusing visualization of relations that span multiple lines**	Coreferences that span multiples lines are displayed in a confusing manner: an arrow (above the line of text) points until the end of the line and starts anew in the next line. For every line, there is a new arrow, until the anaphor is reached. This becomes even more confusing when several, parallel coreference annotations are used.								1			3	Annotation visualization
BRA24	**No way to show / hide annotations from different levels**	It is not possible to show / hide annotations from different levels; the user does see all annotations at any time.								1			3	Annotation visualization
24	**Usability problems**		7	3	2	6	12	4	6	3	3	2	3	

1	**Installation**
7	**General UI**
2	**Primary data**
5	**Annotation scheme**
7	**Annotation process**
2	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-BRA01	**Quick tutorial on first start of the tool**	There is a quick tutorial when first starting the tool, introducing the basic concepts of the tool.										1		Annotation process
S-BRA02	**Numbered lines**	The lines of the primary text are numbered for facilitated navigation.					1	1						Primary data
S-BRA03	**Row striping for primary data**	The lines of the primary data are alternating between white and light grey for improved readibility.					1			1				Primary data
S-BRA04	**Appearance of the primary data can be adjusted**	By using the browser function to adjust the font size, the primary data (as well as the annotation scheme) can be adjusted. It is also possible to adjust the appearance of the primary data via the "Options" menu, which allows the user to set the text density as "dense", "normal" or "spacious" (the width of the screen can also be set, but this does not work properly).			1					1				Primary data
S-BRA05	**Easy navigation between different primary data documents**	The user may navigate between different primary data documents that have been previously imported by means of small arrow controls.							1					Primary data
S-BRA06	**Intuitive syntax for creating annotation schemes outside the tool**	The tool does not require the user to be experienced in XML, but rather uses a syntax that is quite intuitive. Hierarchies are for instance realized by means of "tab indents". The syntax is similar to the BioNLP format.					1							Annotation scheme
S-BRA07	**Appearance of the annotation scheme window can be adjusted**	When there are many items in the annotation scheme, the window can be enlarged to display all items at once.			1									Annotation scheme
S-BRA08	**Different modes of annotation and shortcuts (accelerators)**	It is possible to define different modes of annotation: (1) in "Careful" mode, each annotation must be confirmed with an "ok" button, (2) in "Normal" mode, each annotation is applied as soon as a radio button has been selected. This is very effective for advanced users, especially if combined with the possibility to define individual shortcuts for different annotation items.							1					Annotation process
S-BRA09	**Note taking for annotations**	It is possible to write down notes for an annotation. This note is displayed when hovering over the annotation.		1				1						Annotation process
S-BRA10	**Display of anchor as text string during the annotation process (feedback)**	After an anchor has been selected, the annotation scheme opens. It also contains the selected anchor as a text string. This feedback helps the annotator to make sure that he has selected the right anchor scope.	1					1						Annotation process
S-BRA11	**Comparison mode**	It is possible to view two versions of the annotated document side by side, in order to compare differences between annotators.							1		1			Annotation process

S-BRA12	**Hovering over annotated anchors displays the annotation values**	Annotation values are displayed when the user hovers over an annotated anchor.	1							1				Annotation visualization
S-BRA13	**Annotations as labels**	Annotations are not only visualized by means of color, but also have a label with the actual annotation value. In case there is not enough space to display a longer label, it is possible to define a shorter, alternative version that will then be displayed instead.						1						Annotation visualization
13	**Specific strengths**		2	0	1	0	3	4	3	3	1	1		

0	**Installation**
0	**General UI**
4	**Primary data**
2	**Annotation scheme**
5	**Annotation process**
2	**Annotation visualization**

D.6. MMAX2

Source `http://mmax2.sourceforge.net/`

Date of download November 12, 2013

Date of evaluation November 12, 2013 (tasks 1), November 17, 2013 (tasks 2-6)

Number of usability problems 22

Number of specific strengths 4

Usability problems

ID	Problem	Description	Violated heuristics										Seve-rity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
MAX01	**Console window in the background is irritating**	The console window in the background is irritating. If the user closes it, the whole tool is closed.					1			1			2	General UI
MAX02	**Free floating windows are confusing**	The tool is organized in many separate windows. It is hard for the user to arrange them in a meaningful order. The many different windows easily overlap each other, and are thus rather confusing.				1				1			2	General UI
MAX03	**Forced pre-processing of primary data**	Although pre-processing can facilitate the annotation process in general, it should be optional, because the parameters that have to be set are oftentimes complex.					1	1					2	Primary data
MAX04	**Import of primary data is unclear**	Typically, data can be imported to a tool via the "File" menu - this is not the case for MMAX2. In fact, the user has to click on the "Tools" menu and open the "Project Wizard".				1							2	Primary data
MAX05	**Import of primary data, pre-processing and basic annotation scheme creation are intertwined in one dialog step**	The "Project Wizard" is not really a wizard. There are many parameters that must be set, but it is not clear what needs to be done in order to successfully import the primary data. It is not clear why the user needs to specify basic annotation levels at this stage of the annotation process. Import / pre-processing of primary data and scheme creation should be two different steps. It is also not clear that certain steps are necessary before another step can be taken (e.g. "analyse file" before "tokenize", or "add level" before "create project").								1			4	Primary data
MAX06	**Project wizard settings cannot be changed once a project has been created**	Once a project has been created, it is not possible to modify the parameters (e.g. name and color of annotation levels, project name, etc.) via the wizard GUI. These parameters may only be changed outside the tool, by editing the multipe project files in an XML editor.			1				1				2	Primary data
MAX07	**Tokenizer is not intuitive**	There are many parameters available for the tokenizer. It is not clear how they have to be set and how they affect the tokenization process. It is not clear that tokenization is obligatory in order to create an annotation project in MMAX2. It is also not clear that the default settings can be readily used.		1				1					2	Primary data

MAX08	**Multiple files that are required to build a project are confusing**	In order to create a project, the user is required to define paths to a number of different project files (words file, basedata, style, markable, scheme, customization). Although there is an option to define one generic project path, which can then be "used for all" files, the standard option to specify paths for each file may be confusing. The user should not be bothered with the different components of the project, but they should rather be created automatically in the background. There should be an expert-mode, in which the files can be modified (if necessary).					1			1			3	Primary data
MAX09	**Creation of an (empty) annotation layer and creation of actual annotation items as two different steps**	The wizard for creating a new annotation layer does not contain a step which allows the user to add new tags to the layer. It is not clear why the definition of tags requires an additional step (click on "edit" layer).					1		1	1			2	Annotation scheme
MAX10	**Validate prompt after immediate project creation unclear**	The validate-prompt after creating a new project is hard to understand. It is not clear what should be validated, as there has not been any scheme definition or annotation so far.		1				1					1	Annotation scheme
MAX11	**Scheme creation and modification outside the tool**	The creation of new annotation layers or the modification of existing annotation layers can only be realized outside the tool, by using an XML editor. Defining relational annotations in this fashion can be quite cumbersome.					1		1				3	Annotation scheme
MAX12	**Manual synchronization of external scheme changes within the tool**	Changes to the annotation scheme (outside the tool) have to be updated manually ("update current panel") in the tool.						1					2	Annotation scheme
MAX13	**Insufficient visualization of hierarchical structure in the annotation**	The hierarchy of annotation items is not visualized. The user has to click through the hierarchy, as only one level is displayed at a time.								1			2	Annotation scheme
MAX14	**First item from annotation scheme cannot be selected**	The first element from the annotation drop-down menu cannot be used / applied before some other element from the list has been clicked.			1		1						4	Annotation process
MAX15	**Some functions are unclear**	In the attribute window, there is a "One-click annotation" menu; it is not clear what it does or how it can be used (although described in the manual). In the attribute window, there is a settings menu - it is not clear what is the function of "anno hint".				1		1					3	Annotation process

MAX16	**Creation of anchors and annotations is cumbersome**	The process to select an annotation base and apply an actual annotation is very complicated and unintuitive / unefficient. First, the user has to select the anchor via click-drag-release in order to create a *markable*. The user also chooses on which annotation level the markable should be created. Then, the markable has to be clicked, to select it (again). From the tagset-window, a tag can be selected and applied (apply-button) to the selected markable.	1		1		1	1					3	Annotation process
MAX17	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone.			1	1			1				3	Annotation process
MAX18	**Mouse interaction is not intuitive**	The different context menus which appear on left and right clicks on either a markable, an annotated markable or a word which has not yet been selected as a markable, is not intuitive and generates serious overhead in understanding the overall contex menu concept.	1		1		1	1					4	Annotation process
MAX19	**Styling of markables only outside the tool**	Markables can be styled only by using XSL, or by using the wizard in the beginning. If the wizard is first used, it is not clear what effect the styling option will have on the later annotation (no preview).					1		1				3	Annotation visualization
MAX20	**Annotations are displayed in a confusing way on the context menus**	Annotated texts are not easy to recognize. The info in the context menu (right click) and in the annotation window is not easy to read / understand, especially for a larger number of annotations.							1	1			3	Annotation visualization
MAX21	**No display of parallel annotations**	Parallel annotations are hard to identify, as they are only displayed in the context menu. Higher level annotations cover lower level annotations. There is, however, an option to "hide" certain levels of annotation in the "markable level control panel".	1					1					4	Annotation visualization
MAX22	**Display only one relational annotation at a time**	Relational annotations can only be displayed when one of the markables (that are part of the relation) is clicked. It is not possible to display all relations at once.	1					1					2	Annotation visualization
22	**Usability problems**		4	2	5	4	9	9	6	7	0	0	2,6	

0	**Installation**
2	**General UI**
6	**Primary data**
5	**Annotation scheme**
5	**Annotation process**
4	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	**H2**	**H3**	**H4**	**H5**	**H6**	**H7**	**H8**	**H9**	**H10**		
S-MAX01	**Base-data editing mode**	Primary data can be edited in a special mode within the tool.					1							Primary data
S-MAX02	**Customizable display of primary data inside the tool**	The display of the primary data can be modified inside the tool. The parameters that can be set are *font size, font family* and *line spacing*.			1									Primary data
S-MAX03	**Linebreaks after certain markables**	It is possible to define linebreaks after certain markables, e.g. after paragraphs.						1						Primary data
S-MAX04	**Automatic validation**	Although the creation and modification of annotation schemes outside the tool is cumbersome, it is a good feature to validate the external XML schemes whenever a project (and its associated schemes) is opened.					1							Annotation scheme
4	**Specific strengths**		0	0	1	0	2	1	0	0	0	0		

0	**Installation**
0	**General UI**
3	**Primary data**
1	**Annotation scheme**
0	**Annotation process**
0	**Annotation visualization**

D.7. WordFreak

Source `http://wordfreak.sourceforge.net/`

Date of download November 22, 2013

Date of evaluation November 22, 2013

Number of usability problems 15

Number of specific strengths 8

Usability problems

ID	Problem	Description	Violated heuristics										Seve-rity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
WOR01	**Documentation is lacking important information**	There is no detailed manual, only a very basic help function inside the tool. This is problematic, as many functions and mechanics do not work intuitively and require a lot of trial and error until the user figures out how they work (some functions remain unclear, e.g. "Filters" etc.).					1					1	4	General UI
WOR02	**Redundant functions and UI elements**	"Add" files, "Viewer", "Annotation" and "Tagger" are redundantly available in the menu header and at the right side or bottom side of the window. This does not speed up the annotation process, but rather irritates the novice user.								1			1	General UI
WOR03	**Primary data cannot be added via "File" menu**	This is a convention in most software tools: import / add files via the "File" menu. Here, the user first has to create a project, to which he can then "Add" primary data.				1							1	Primary data
WOR04	**Cumbersome process for the actual display of primary data**	The process for displaying primary data involves a number of different steps: First, the primary data needs to be added and loaded. Next, a viewer has to be selected from the "Viewer" menu (there should be a default viewer that is automatically selected). The data is only displayed in the viewer if an annotation scheme has been selected via the "Annotation > Set annotation" menu.					1		1				4	Primary data
WOR05	**No definition of individual annotation schemes inside the tool**	It is not possible to define new annotation schemes inside the tool. Even the definition outside the tool (modify an XML document) is very cumbersome and poorly documented.					1					1	4	Annotation scheme
WOR06	**Display of only one annotation scheme at a time**	Only one annotation scheme can be selected at a time.		1									3	Annotation scheme
WOR07	**Poor documentation of predefined annotation schemes**	The predefined annotation schemes are poorly documented and thus hard to use for novice users. There is no comprehensive list that explains the various abbreviations for the tags. There is also no information on how the tag sets are to be used.					1					1	4	Annotation scheme
WOR08	**No selection of word via double click**	Selecting a word by double clicking it is a convention known from many word processors. WordFreak however does not support this function.				1			1				2	Annotation process

WOR09	**Multiple "arrow" buttons are confusing**	Two of the arrow buttons in the annotation scheme window are for navigating between the anchors, the other four are used to shrink or expand the scope of a selected anchor. The buttons can easily be confused, because they are grouped together (navigate as well as shrink and expand). The user might think, that the shrink / expand left buttons can be used to jump greater distances between the anchors (analogy from stereo system controls).		1			1						2	Annotation process
WOR10	**No efficient annotation process for multiple levels of annotation**	It is not possible to simultaneously annotate the text with items from different annotation schemes without explicitly switching the schemes, which is very ineffective.			1				1				3	Annotation process
WOR11	**Deletion of annotations is cumbersome**	Annotations can only be deleted if the respective scheme is active.			1				1				3	Annotation process
WOR12	**No phrase annotation without previously defined POS**	It is not possible to annotate "noun phrases" (annotation scheme: "Constituent") unless the parts of speech of the phrase have been annotated before.					1			1			2	Annotation process
WOR13	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone.			1	1			1				3	Annotation process
WOR14	**Visual overlapping of annotation values above anchors**	POS and phrase annotations are displayed above the respective anchors in the primary data. However, parallel annotation values (e.g. NN and NP) tend to overlap in the visualization. There is also no automatic adjustment of the line spacing, which means annotations can overlap with other primary data.	1										3	Annotation visualization
WOR15	**No direct visualization of annotations in the primary data**	Apart from the POS and phrase annotations, annotation values are only displayed in the footer of the main window.	1										2	Annotation visualization
15	**Usability problems**		2	2	3	3	6	0	5	2	0	3	2,7	

0	**Installation**
2	**General UI**
2	**Primary data**
3	**Annotation scheme**
6	**Annotation process**
2	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-WOR01	**Customizable display of primary data inside the tool**	The display of the primary data can be modified inside the tool. The parameters that can be set are *font size*, *font family* and *orientation* (left-to-right vs. right-to-left).			1					1				Primary data
S-WOR02	**Encoding can be adjusted within the tool and during the annotation process**	It is possible to adjust the encoding (e.g. UTF-8) within the tool via the "Font > encoding" menu. Changes are applied live during the annotation process; there is no need to update or reload the document.			1									Primary data
S-WOR03	**Alternative views for the display of primary data and annotation**	There are different views for displaying the primary data as well as the annotations. Among these views are tabular and tree-like visualizations.			1					1				Primary data
S-WOR04	**Automatic creation of an annotation file after a primary document is imported**	Once the primary data has been added to a project, the tool asks the user if an annotation file should be created automatically.					1		1					Primary data
S-WOR05	**Jump back and forward between anchors**	The user can navigate from one anchor to the preceding or succeeding anchor by using two buttons in the annotation scheme window, or by using the "right" and "left" keys from the keyboard.							1					Annotation process
S-WOR06	**Characterwise definition of anchor scopes**	The scope of an anchor can be adjusted by means of four buttons, or by using the "right" and "left" keys from the keyboard (in combination with SHIFT and CTRL).					1							Annotation process
S-WOR07	**Shortcuts for applying certain annotation values**	Certain annotation values can be applied by using predefined shortcuts (e.g. CTRL + 1).							1					Annotation process
S-WOR08	**Display of annotation values above the anchor**	POS and phrase annotations are displayed above the respective anchor.						1		1				Annotation visualization
8	**Specific strengths**		0	0	3	0	2	1	3	3	0	0		

0	**Installation**
0	**General UI**
4	**Primary data**
0	**Annotation scheme**
3	**Annotation process**
1	**Annotation visualization**

D.8. Analec

Source `http://www.lattice.cnrs.fr/Telecharger-Analec?lang=fr`

Date of download November 24, 2013

Date of evaluation November 24, 2013

Number of usability problems 15

Number of specific strengths 4

Usability problems

ID	Problem	Description	Violated heuristics										Severity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
ANA01	**Menu items without an implemented function**	There are many menu items that decribe functions which are not yet implemented ("Désolé, cette fonctionalité n'est pas encore implémentée").	1				1						4	General UI
ANA02	**No English version**	The tool is only available in French language.		1		1							3	General UI
ANA03	**No documentation**	There is no documentation / manual available. Some basic functions are described in an LREC paper.					1					1	4	General UI
ANA04	**Wording for creating an annotation scheme is not intuitive**	The tool makes use of an annotation model that relies on units (= anchors), relations (= relations), and schemas (= groups of anchors, relations and other schemas). An annotation tool should be theory-neutral and use intuitive concepts.		1		1							2	Annotation scheme
ANA05	**Hierarchy in annotation scheme tree is unclear**	It is not clear how the hierarchy of the tree (window for creating new schemes) translates into the actual annotation scheme.	1				1						3	Annotation scheme
ANA06	**Order of levels and items in annotation scheme cannot be adjusted**	The order of nodes in the annotation scheme tree cannot be modified; new nodes are automatically inserted on the top level.		1	1			1					2	Annotation scheme
ANA07	**No differentiation of items in the annotation scheme**	The different levels of annotation and the corresponding annotation items are displayed in horizontal order. They cannot be differentiated by means of different colors.						1					2	Annotation scheme
ANA08	**Scrolling in the annotation scheme drop-down lists required**	The items of an annotation level are displayed in a drop-down list. The list, however, only shows eight items at max, i.e. at times the list has to be scrolled to be able to access certain elements from the annotation scheme.							1				2	Annotation scheme
ANA09	**Annotation scheme only visible when an actual annotation is applied**	The annotation scheme is only visible when an actual annotation is applied, i.e. the user has to know what is in the scheme before he decides to apply an annotation.			1			1					2	Annotation scheme
ANA10	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone.			1	1			1				3	Annotation process
ANA11	**No feedback after the application of an annotation**	There is no feedback after the application of an annotation; the user cannot be sure if the annotations have been successfully applied and stored.	1										2	Annotation process

ANA12	**Switching to different annotation modes is impractical**	Whenever the user wants to annotate, he has to select the *annotation* mode. Whenever he wants to change an annotation, he has to switch to *edit* mode. This is impractical - annotation modes should rather be activated automatically whenever the user selects an item from the respective annotation scheme. The *change* / *delete* mode could be toggled in another way (e.g. by a right click).					1						4	Annotation process
ANA13	**Relation annotation is cumbersome**	The user has to select an anchor for "element 1" and another anchor for "element 2"; then he has to create a relation annotation.					1		1				3	Annotation process
ANA14	**No direct display of annotations during the annotation process**	Annotations are only displayed (in the annotation scheme window) when the respective anchor is clicked. If colors were defined for certain annotation items, these are only displayed when no annotation scheme is actively selected, i.e. the colored annotations cannot be displayed during the annotation process.	1										4	Annotation visualization
ANA15	**No efficient mechanism for the display of all annotations of one type**	It is not possible to show or hide specific annotation items or annotation levels in an efficient way, e.g. by means of a checkbox that is positioned in the annotation scheme window. There are several options in the "views" menu, but it is cumbersome to use those options to show or hide specific annotations.			3				3				3	Annotation visualization
15	**Usability problems**		4	3	6	3	5	3	6	0	0	1	2,9	

0	**Installation**
3	**General UI**
0	**Primary data**
6	**Annotation scheme**
4	**Annotation process**
2	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-ANA01	**Set encoding during the import process**	The encoding of the primary data can be set during the import process, i.e. before the actual document is displayed in the tool.					1							Primary data
S-ANA02	**Edit primary data mode**	The primary data can be edited in a special mode.					1							Primary data
S-ANA03	**Adjust display of primary data**	The display of the primary data can be adjusted (*font family*, *font size*, *line height*, *letter spacing*, *margins,* etc.)			1					1				Primary data
S-ANA04	**Tree metaphor in the annotation scheme creation GUI**	The metaphor of a hierarchical tree for the definition of an annotation scheme is intuitive and effective.		1					1					Annotation scheme
4	**Specific strengths**		0	1	1	0	2	0	1	1	0	0		

0	**Installation**
0	**General UI**
3	**Primary data**
1	**Annotation scheme**
0	**Annotation process**
0	**Annotation visualization**

D.9. WebAnno

Source `http://code.google.com/p/webanno/`
(all-in-one-version: "webanno-0.5.0-beta-13-standalone.jar")

Date of download December 3, 2013

Date of evaluation December 3, 2013

Number of usability problems 22

Number of specific strengths 15

Usability problems

ID	Problem	Description	Violated heuristics										Seve-	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	rity	
WEB01	**Installation requires technical skill (server / database)**	Although there is an all-in-one-version available for download, the developers make clear that this version is only for sandboxing and experimenting with the tool. In order to use WebAnno productively, it has to be set up properly. The technical requirements are a Tomcat server and a MySQL database.					1						2	Installation
WEB02	**No save button**	The project is automatically saved, but there is no save button that explicitly allows the user to save a document and its annotations.	1			1							3	General UI
WEB03	**Main navigation is not intuitive and accessible**	The main navigation is realized as a vertical list of items. The order of that list is not clear: "Project" should be the first option, as it is needed to use the tool. In order to access the main navigation, the user has to substitute the current window with the main navigation window. This is not intuitive / accessible. Typically, main navigation items are positioned as a horizontal menu structure at the top of the screen.				1			1				3	General UI
WEB04	**No explicit confirmation for setting the project parameter**	After the parameters for a project have been set (import document, create tagsets, etc.) there is no way to explicitly confirm the settings.	1	1									3	General UI
WEB05	**Resizing of "open documents" dialog is cumbersome**	The "open documents" dialog window can be resized by means of small, red arrows.				1	1						2	General UI
WEB06	**Import of primary data requires definition of a project**	In the main menu, the first item is "Annotation"; if the user clicks on that item for the first time, he switches to the annotation window, which asks him to open a project / document. It is not clear from here that primary data can only be imported if a project is created. It is also not clear where or how a project can be created.					1						3	Primary data
WEB07	**Import of primary data not intuitive**	In the "Project" window, the user can create a project. Only when an existing project is selected, several tabs that contain related options appear. The option for importing documents (which is most important) is in the third tab named "Documents". At first use, it is not clear how primary data can be imported into the tool.					1						3	Primary data
WEB08	**No import of multiple documents**	Although in the "Data" dialog is a function "New collection - Upload tar.gz", the function does not work.	1				1						4	Primary data

WEB09	**"Open" button for documents is not visible by default**	When the user enters the "Annotations" window, he is asked to open a document from a project. The "open" button (as well as the "cancel" button) are not fully visible in the small dialog window, the user has to manually resize the window to view the controls.					1						2	Primary data
WEB10	**Opening a document takes several seconds**	Although the test document only consists of 75 sentences, it took several seconds to open the document.							1				3	Primary data
WEB11	**Bug for reopening a previously annotated document**	If the user wants to reopen a previously annotated document, he has to click on the "Open" option, as the default "open document" dialog does not work.						1	1				4	Primary data
WEB12	**Restriction for annotation schemes through predefined "layer types"**	Annotation schemes are restricted to the six basic layer types that are available in the tool. Each layer type has implications for the anchor scope. These implications are not always clear. The layers do not cover all theoretical anchor scopes: e.g. it is not possible to define structural annotations that span several sentences. Another restriction implied by this model is, that only one annotation scheme can be defined per layer (i.e. it is not possible to define multiple annotation layers that use the anchor scope "token").		1			1						3	Annotation scheme
WEB13	**Colors for annotations cannot be defined / edited inside the tool**	Each annotation item has the same color by default. Colors need to be specified outside the tool, in a "visual.conf" file, as HTML or hex color code.						1	1				4	Annotation scheme
WEB14	**Annotation scheme is only visible after selecting an anchor**	The annotation scheme can only be displayed after an anchor has been selected.	1		1								1	Annotation scheme
WEB15	**Order of the annotation items cannot be modified**	All annotation items are ordered alphabetically - there is no way to sort them manually.			1				1				2	Annotation scheme
WEB16	**Scrolling in the annotation scheme drop-down lists required**	The items of an annotation level are displayed in a drop-down list. The list, however, only shows nine items at max, i.e. at times the lists has to be scrolled to be able to access certain elements from the annotation scheme.							1				2	Annotation scheme
WEB17	**Delay when applying / deleting an annotation**	There is a significant delay when applying or deleting an annotation.	1						1				3	Annotation process
WEB18	**Deletion of annotations is cumbersome**	If the user wants to delete an existing annotation, he has to double click on the label. The annotation pop up window opens and the user needs to click on the "delete" button to delete the annotation. There are no shortcuts (DEL key) or options to delete multiple annotations at once.							1				3	Annotation process

WEB19	**Interaction for creating relations is irritating**	Relations between two existing annotations are created by means of click (first annotation) and release (second annotation). While the first annotation is selected at its label, the second annotation has to be selected slightly underneath the label, which is not intuitive.					1		1				3	Annotation process
WEB20	**Tool jumps to the top of the document after each annotation**	If an annotation is applied to an anchor that is located further down the page (scrolling necessary), the tool jumps back to the top of the page after each successfully applied annotation. This is very unefficient, as the user has to scroll down to the actual annotation position every time; it is not clear what the "auto scroll" feature actually does, or if it works as it should.			1				1				4	Annotation process
WEB21	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone.			1	1			1				3	Annotation process
WEB22	**Confusing visualization of relations that span multiple lines**	Coreferences that span multiples lines are displayed in a confusing manner: an arrow (above the line of text) points until the end of the line and starts anew in the next line. For every line, there is a new arrow, until the anaphor is reached. This becomes even more confusing when several, parallel coreference annotations are used.								1			3	Annotation visualization
22	**Usability problems**		5	2	4	4	7	2	11	1	0	0	2,9	

1	**Installation**
4	**General UI**
6	**Primary data**
5	**Annotation scheme**
5	**Annotation process**
1	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-WEB01	**Numbered lines**	The lines of the primary text are numbered, for facilitated navigation.					1	1						Primary data
S-WEB02	**Row striping for primary data**	The lines of the primary data are alternating between white and light grey for improved readibility.					1			1				Primary data
S-WEB03	**Page metaphor and adjustable number of sentences per page**	The tool allows the user to display a document on several pages, which can be navigated forward and backward by means of arrow controls. It is also possible to define the number of sentences to be displayed per page. At the top of each page, the tool shows how many sentences in relation to the total number of sentences are currently displayed (example: "showing 1-20 of 75 sentences").		1					1					Primary data

S-WEB04	**Easy import and export of tagsets in JSON format**	Tagsets can be imported or exported in structured JSON format.							1					Annotation scheme
S-WEB05	**Easy navigation between different primary data documents**	The user may navigate between different primary data documents that have been previously imported by means of small arrow controls.							1					Primary data
S-WEB06	**Tag sets can be created inside the tool (GUI)**	As annotation schemes can only represent one hierarchical level, it is relatively easy to create tag sets and tags. First, the user creates a tagset via a form, second, he can add arbitrary tags via another form. Each added tag is displayed in a list, and can be modified or deleted.	1						1					Annotation scheme
S-WEB07	**Availability of example tagsets for different layers of annotation**	There are predefined example tagsets that can be used right away to achieve common annotation tasks such as POS annotation. The example tagsets also help to unserstand how the different layers work, and how they behave during the actual annotation process.						1						Annotation scheme
S-WEB08	**Layers can be set visible and hidden in the annotation scheme window**	This is very helpful, as it allows the user to hide annotation schemes that are not needed at a specific point in time, which in turn reduces the overhead given by many parallel schemes. The user has fewer options to choose from, which leads to reduced complexity and less cognitive overhead.			1				1	1				Annotation scheme
S-WEB09	**Last selected tag value becomes the new default value**	This can speed up the annotation process if the annotator decides to annotate, e.g., all "nouns" first. Other tools have the first tag in the scheme as default, or no default value at all.							1					Annotation process
S-WEB10	**Different users and roles can be created within the tool**	Users can be defined and assigned to projects inside the tool.			1									Annotation process
S-WEB11	**Monitoring of user progress**	The admin user can monitor the progress of different annotators for different projects. It is also possible to monitor the level of agreement for different annotators on different levels of annotation.	1											Annotation process
S-WEB12	**Display of anchor as text string during the annotation process**	After an anchor has been selected, the annotation scheme opens. It also contains the selected anchor as a text string. This feedback helps the annotator to make sure that he has selected the right anchor scope.	1					1						Annotation process
S-WEB13	**Curation mode for comparison of parallel annotations**	A special curation mode allows the user to display parallel annotations by different annotators, and to merge them.							1		1			Annotation process
S-WEB14	**Hover over annotated anchors displays the annotation values**	Annotation values are displayed when the user hovers over an annotated anchor.	1							1				Annotation visualization

S-WEB15	**Annotations as labels**	Annotations are not only visualized by means of color, but also have a label with its actual value. In case there is not enough space to display a longer label, it is possible to define a shorter, alternative version that will then be displayed instead.						1						Annotation visualization
15	**Specific strengths**		4	1	2	0	2	4	7	3	1	0		

0	**Installation**
0	**General UI**
4	**Primary data**
4	**Annotation scheme**
5	**Annotation process**
2	**Annotation visualization**

D.10. Knowtator

Source Protégé: `http://protege.cim3.net/download/old-releases/Protege%203.x/3.3.1/full/`;
Knowtator: `http://sourceforge.net/projects/knowtator/files/`

Date of download December 6, 2013

Date of evaluation December 6, 2013 (task 1), December 8, 2013 (tasks 2-6)

Number of usability problems 17

Number of specific strengths 16

Usability problems

ID	Problem	Description	Violated heuristics										Seve-	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	rity	
KNW01	**Activation of Knowtator plugin requires previous creation of a project**	Knowtator cannot be activated as a plugin unless the user creates (and saves) a project in Protégé before.					1						2	Installation
KNW02	**No explicit feedback in dialogs that change the settings of the project**	Whenever the user changes settings or properties of the current project, a dialog window opens. It is not possible to explicitly save / apply the changes via a button, but rather the changes are stored by closing the window via a red "X" icon. There is no feedback if the changes have been applied successfully.	1			1							3	General UI
KNW03	**Overhead through the complex knowledge model of the Protégé framework**	Protégé is a complex framework for the modelling of ontologies. Therefore it uses concepts such as classes and slots. For the standard linguistic annotator, these concepts may be hard to grasp and hard to differentiate.		1						1			2	General UI
KNW04	**Modifications of the font size are globally applied to menus and controls**	The font size of primary data can be increased or decreased. These modifications are however also applied globally to the font used in the tool's menus and controls.			1								1	General UI
KNW05	**Controls not fully visible because of logo**	The Protégé logo on the right covers some of the controls when the main window is displayed in full screen mode (1280 x 1024 pixels resolution). The arrows for navigating between annotations are hardly visible.					1						1	General UI
KNW06	**Arrow controls for different functions cannot be distinguished easily**	There are three different "arrow controls" in the bar beyond the main navigation bar. The first control is for switching primary data documents, the second for switching filters, and the third is for navigating between annotations. The first two arrow controls also have a small "document icon" in their center. It is, however, not easy to distinguish the function of the three arrow controls, as they look very similar.	1	1				1					3	General UI
KNW07	**Proprietary Protégé files are irritating / unclear**	When creating a new project in Protégé, the user may select project type "Protégé Files", which consists of *.pont and *.pins files. It is not clear what these two different files do, or whether the user has to create them manually.		1				1					2	Primary data
KNW08	**Icon for the import of primary data is not intuitive**	Primary data can only be imported by means of a folder icon that is positioned beyond the menu structure, toward the right side of the window. Usually, the user would expect to find an "open / import document" dialog in the "file" menu, or on another prominent interface position.				1							2	Primary data

KNW09	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone.			1	1			1				3	Annotation process
KNW10	**Some functions are unclear**	It is not clear how the advanced functions "Required mode" and "Consensus mode" work.				1		1					3	Annotation process
KNW11	**Bug: Anchor selection does not work**	During the annotation process, it happened a few times that the anchor selection mechanism did not work; solution: restart Protégé.					1				1		4	Annotation process
KNW12	**Difference between "clear" and "delete" annotation is not clear**	Despite the explanation on the tool tip, it is not clear what "clear annotation" actually does, and how it is different from "delete annotation".					1						2	Annotation process
KNW13	**Coreference annotation is cumbersome**	Once the antecedent and the referent anchors have been defined, it is possible to add a coreference relation. This is achieved by selecting the antecedent anchor and by adding referent anchors via the "slots" menu. The menu shows all available referent anchors, which can be rather confusing.					1		1				2	Annotation process
KNW14	**Bug: Not all annotations are displayed in the text**	Even though they should be displayed because of the active filter, some annotations are not displayed with their respective color highlighting; solution: restart Protégé.					1				1		4	Annotation visualization
KNW15	**Insufficient visualization of parallel annotations in the primary data**	It is hard to distinguish multiple, parallel levels of annotation by means of overlapping, colored highlights.								1			2	Annotation visualization
KNW16	**Bug: Filters do not work properly**	Sometimes a newly defined filter does show no effect when activated. Solution: restart Protégé. Note: In one test case (after a restart), the previously created filters prevented Knowtator from being available in Protégé at all - only after the deletion of those filters and another restart, Knowtator "came back".					1				1		4	Annotation visualization
KNW17	**Creation of filters is cumbersome**	In the "Configure" menu, there are two forms "active filters" and "currently selected filters". Via the "Add Instance" icon it is possible to add classes to a filter. It is not clear, what is the difference between "Knowtator filters" and "Knowtator consensus filters". It is also not clear what is the difference between "Direct Instances" and "All Instances". The function of the empty drop-down menus at the bottom of the dialog is unclear, too.					1	1					2	Annotation visualization
17	**Usability problems**		2	3	2	4	7	4	2	2	3	0	2,5	

1	**Installation**
5	**General UI**
2	**Primary data**
0	**Annotation scheme**
5	**Annotation process**
4	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-KNW01	**Advanced functions are explained in an introductory dialog**	If the user chooses one of the advanced functions such as "Remove annotations", "Merge annotation", "Reassign annotator value in batch", "Assign annotation set value in batch", etc., an introductory dialog explains how the functions work, and how they might have an impact on the annotation project.						1				1		General UI
S-KNW02	**Batch import of documents**	It is possible to import a whole directory with primary data documents at once.							1					Primary data
S-KNW03	**Easy navigation between different primary data documents**	The user may navigate between different primary data documents that have been previously imported by means of small arrow controls.							1					Primary data
S-KNW04	**Annotation scheme hierarchy can be modified by means of drag-and-drop**	The annotation scheme is displayed as a hierachical tree, in which nodes can be modified by means of drag-and-drop.				1			1					Annotation scheme
S-KNW05	**Possibility to use only selected annotation levels and / or tags during the annotation process**	In the "Configure" menu it is possible to define which layers of annotation or which single tags are available during the annotation process.			1				1					Annotation scheme
S-KNW06	**Navigate existing annotations with arrow controls**	It is possible to jump from one annotation (of a selected type) to the next or back, by using two small arrow icons, which are positioned in the top bar of the main window.							1					Annotation process
S-KNW07	**Feedback before applying an annotation**	Before an annotation is applied to a selected anchor, the tool gives feedback on the annotation value as well as on the selected anchor in the following form: "Create ANNOTATION VALUE annotation with ANCHOR VALUE**".**	1				1							Annotation process
S-KNW08	**Counts for different types of existing annotations**	There are counts behind each item in the annotation scheme, that indicate the number of existing annotations of this type.	1											Annotation process
S-KNW09	**Existing annotations are listed in a context menu**	When a new annotation is applied, the context menu shows a list of anchors that were already annotated with the same annotation value.	1				1	1						Annotation process

S-KNW10	**Fast annotation mode for efficient annotation**	When applying an annotation value, there is always the option to enable "fast annotation mode". When this mode is enabled for one specific item of the annotation scheme, the value is automatically applied to each anchor selection. The fast mode can be ended via the "Quit" button on top of the annotation window. All tags that have been selected for "fast annotation" previously are also displayed on top of the annotation window, thus allowing the user to quickly switch between different annotation values in "fast mode".							1					Annotation process
S-KNW11	**Selection assistant for anchors with multiple annotations**	If an anchor has multiple, parallel annotations, a context menu is opened that shows all the existing annotations. The user may select one of the annotation items, to modify or delete it.					1							Annotation visualization
S-KNW12	**Anchor scope can be edited by means of small arrows**	The scope of an existing anchor can be edited via small arrow icons.					1							Annotation process
S-KNW13	**Double click anchor selection can be controlled via a regular expression**	Usually, double clicking a word selects the word. The same is true for Knowtator (default setting). It is, however, possible, to modify the "token" that is selected by a double click via a regular expression in the "Configure" menu. Default RE for one word: `\W+`			1				1					Annotation process
S-KNW14	**Mass removal of annotations**	Multiple annotations can be removed by applying a previously defined filter. Any annotation that is not specified in the filter will be removed. This can be a very efficient function to get rid of unwanted levels of annotation, or to clean up the document.							1					Annotation process
S-KNW15	**Tailored visualization for levels of annotation or for single tags**	It is possible to define show / hide filters on different levels of granularity, e.g. for a whole scheme, for different levels of annotation, or for single tags from one level of annotation.			1					1				Annotation visualization
S-KNW16	**Colors for annotation levels or for single tags**	Colors can be specified inside the tool, either for levels of annotation or for single tags. Colors can either be specified by means of RGB values, or by choosing names from a predefined color palette.			1				1					Annotation visualization
16	**Specific strengths**		3	0	4	1	4	2	9	1	0	1		

0	**Installation**
1	**General UI**
2	**Primary data**
2	**Annotation scheme**
8	**Annotation process**
3	**Annotation visualization**

D.11. GATE

Source `http://gate.ac.uk/`

Date of download December 9, 2013

Date of evaluation December 9, 2013

Number of usability problems 21

Number of specific strengths 4

Usability problems

ID	Problem	Description	Violated heuristics										Seve-rity	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
GAT01	**Datastore option is irritating**	The datastore metaphor and its function is unclear. Also, the process of creating a datastore is unclear. The user typically would expect to be able to just save the data / project via the "File" menu.		1			1	1					2	General UI
GAT02	**Wording is too technical / too specific**	The wording for setting the parameters is quite technical and hard to understand for standard users. The metaphors for "application", "data store", "language resource" and "processing resource" are hard to differentiate. This problem is due to the fact that GATE is more than just a manual annotation tool.		1			1	1					2	General UI
GAT03	**Wording for opening existing annotation schemes is irritating**	The option "New" > "Annotation Schema" suggests that a new scheme can be created, where actually only an existing scheme in a certain format can be imported.		1			1						1	General UI
GAT04	**Accidental manipulation of primary data**	The original text can be edited and deleted at any time throughout the annotation process, no explicit confirmation is necessary. That results in accidental modifications and even deletions, which cannot be undone.					1						4	Primary data
GAT05	**Import of primary data is cumbersome**	It is not clear at first sight how a text can be imported. Wording ("New Language Resource > GATE Document") and use of icons (folder icon) make it difficult to import a text document.				1	1						2	Primary data
GAT06	**Dialog for importing primary data is cumbersome**	When importing a document, the user has to set many parameters (e.g. encoding) in free text forms. If the user does not know the right / valid value, he cannot set the necessary parameters correctly.		1			1	1	1				3	Primary data
GAT07	**Order of levels and items in annotation scheme cannot be adjusted**	The order of levels of annotation as well as the annotation items on each level cannot be modified; they are arranged in alphabetic order.			1			1					2	Annotation scheme
GAT08	**Default annotation schemes are not explained**	At the beginning, it is not clear which annotation scheme is used. The default annotation schemes are not explained and cannot be disabled inside the tool.	1		1		1						2	Annotation scheme
GAT09	**"Original markups" layer is not clear**	It is not clear where the default annotation level "original markups" with its single tag "paragraphs" comes from.	1		1								1	Annotation scheme
GAT10	**"Empty" layer is irritating**	There is an empty annotation level, which has no name at all. It can neither bei renamend, nor can it be deleted. If no other annotation level is created, the tags are assigned to this level, which function is unclear, as it has no name.					1						2	Annotation scheme

GAT11	**Creation of new annotation levels is inconsistent**	It is not clear how a new annotation level can be created. Although renaming and deleting can be achieved via right click and a context menu, a new layer is created by using a form in the bottom right corner, with a small button "New".				1	1	1					2	Annotation scheme
GAT12	**No renaming of annotation items**	Once a tag is created and applied, it cannot be renamed. If the user wants to change a tag after it has been created, he has to delete it and create a new one. All previous annotations with this tag will of lost and have to be created anew.			1				1				4	Annotation scheme
GAT13	**Application of annotations via click and hovering over selected anchor is not intuitive / practical**	The selection of an anchor via click-and-release is intuitive. It is, however, not clear how to apply an annotation. If the user hovers over the selection for some time, a pop-up menu appears, which allows the user to apply an annotation. It disappears if the user does not move the cursor into the menu quickly.					1		1				3	Annotation scheme
GAT14	**No feedback after application of annotation**	Even if the user manages to open the context menu, it is unclear how to apply the annotation. There is no immediate feedback, that the annotation has been applied correctly.	1				1						3	Annotation process
GAT15	**No modification of previously created item-level relations**	As new tags are created ad hoc, while annotating, it is important to be able to relate newly created tags to an existing level of annotation. It is not clear how to select a level in order to assign a tag to it. It is not possible to reassign falsely assigned tags to another level; the user has to delete the tag and create it anew.			1				1				4	Annotation scheme
GAT16	**No undo / redo of actions**	Actions (e.g. deletions) during the annotation process cannot be un- / redone.			1	1			1				3	Annotation process
GAT17	**Annotation pop-up window contains irritating parameters**	It is not clear what can be done with the additional fields in the annotation context menu.						1		1			3	Annotation scheme
GAT18	**Icon for the deletion of annotations is irritating**	The delete icon (green pen with a red cross) in the annotation pop-up window is not intuitive.				1		1					2	Annotation process
GAT19	**Coreference annotation is not intuitive**	The annotation of coreferences is realized with the "Co-reference Editor". The mechanism is not intuitive: First, a set with coreference annotation items has to be selected. Next, several annotated items can be added to a co-reference chain; the name of the chain is automatically named after the longest item in the chain.					1		1				2	Annotation process
GAT20	**Different views on annotations are not self-explanatory / irritating**	There seem to be multiple views on the annotated data. However, the *set, list* and *stack* view are not self-explanatory.					1	1					2	Annotation visualization

GAT21	**Insufficient visualization of parallel annotations in the primary data**	It is hard to distinguish multiple, parallel levels of annotation by means of overlapping, colored highlights. Note: The "stack" visualization works fine for the display of parallel annotations (cf. strengths section)								1			2	Annotation visualization
21	**Usability problems**		3	1	6	4	10	6	6	2	0	0	2,6	

0	**Installation**
3	**General UI**
3	**Primary data**
9	**Annotation scheme**
4	**Annotation process**
2	**Annotation visualization**

Specific strengths

ID	Strength	Description	Applied heuristics										–	Category
			H1	H2	H3	H4	H5	H6	H7	H8	H9	H10		
S-GAT01	**Auto-annotate of similar anchors**	There is a function for the annotation of similar anchors with the same annotation value *(*first*,* previous, next, all next). This can speed up the manual annotation process significantly. The expression can be formulated in the GATE search expression syntax, which is very sophisticated.							1					Annotation process
S-GAT02	**Annotation list view**	The *list* view shows annotations in a structured, tabular view.								1				Annotation visualization
S-GAT03	**Annotation stack view**	The *stack* view shows parallel annotations in a layer-like visualization.								1				Annotation visualization
S-GAT04	**Checkboxes for show / hide of annotation items**	All annotation levels and items are visible at all times, and can be shown or hidden via checkboxes.							1					Annotation visualization
4	**Specific strengths**		0	0	0	0	0	0	2	2	0	0		

0	**Installation**
0	**General UI**
0	**Primary data**
0	**Annotation scheme**
1	**Annotation process**
3	**Annotation visualization**

E. Overview of interactive prototypes

This appendix gives an overview of the interactive prototypes that were created in the course of this work. All prototypes were implemented by means of *HTML5*[208] and and *JavaScript*. Most of the JavaScript functionality is based on *jQuery*[209], a JavaScript library that is widely used in the web design community. jQuery works across most available web browsers and provides a large set of features for document traversal and *DOM* manipulation, event handling, animation, etc. Another helpful resource used for the creation of the prototypes is *jQuery UI*[210], a JavaScript library that builds on top of jQuery. jQuery UI focuses on different aspects that are concerned with the appearance and behavior of the *user interface* (UI). The library comprises the following main components (cf. http://jqueryui.com/):

Interactions, e.g. drag-and-drop, resizing, sorting, etc.

Widgets, e.g. buttons, dialogs, menus, sliders, progress bars, tabs, etc.

Effects, e.g. adding and removing of CSS classes, color animations, show / hide effects, etc.

In addition to the standard functionality of jQuery and jQuery UI, there is also a large collection of third party plugins[211] for more specific purposes, like e.g. a *color picker widget*.The following sections briefly describe which libraries and plugins were used for each of the prototypes.

E.1. Prototype for P3.1

Functionality Although one tool-specific strength has been related to this pattern, it only describes a partial aspect of the solution, and cannot be used

[208] "HTML5 - A vocabulary and associated APIs for HTML and XHTML" is currently a W3C candidate recommendation (cf. `http://www.w3.org/TR/html5/`).

[209] jQuery is freely available at `http://jquery.com/`. The jQuery version used to implement the prototypes is "jquery-1.10.2".

[210] jQuery UI is freely available at `http://jqueryui.com/`. The jQuery UI version used to implement the prototypes is "jquery-ui-1.10.4".

[211] The official jQuery plugin registry is available at `http://plugins.jquery.com/`. There are, however, more plugins available that are not listed in the official registry, but which may be found via a generic web search.

as a full example (cf. S-DEX02), i.e. the prototype is the only concrete example for this pattern. The prototype implements a basic wizard that guides the user through the import and pre-processing process of primary data documents.

Limitations Although it implements an import dialog, the prototype does not allow the user to import actual primary data documents, but rather works with a static text sample. Only two exemplary pre-processing parameters have been implemented (*tokenizer* and *sentence splitter*).

JavaScript libraries This prototype utilizes the *dialog* widget from the jQuery UI library: `https://jqueryui.com/dialog/`

Online demo `https://dl.dropboxusercontent.com/u/4194636/prototypes/P31-Guided-Preprocessing-Primary-Data.html`

E.2. Prototype for P3.2

Functionality Although two tool examples exist for this pattern, the prototype was created to illustrate the basic interaction steps mapping existing primary data documents to a corpus structure. The prototype allows the user to create and delete corpora (displayed as tabs), and to relate a set of five existing documents to the corpus when it is created.

Limitations The prototype does not allow the user to add and remove documents to a corpus after its creation, e.g. by means of drag-and-drop.

JavaScript libraries This prototype utilizes the *tabs* widget from the jQuery UI library, and more specifically adopts and enhances the following online example: `https://jqueryui.com/tabs/#manipulation`

Online demo `https://dl.dropboxusercontent.com/u/4194636/prototypes/P32-Import-Documents-Mapping-Corpora.html`

E.3. Prototype for P4.1

Functionality Although five tool examples exist for this pattern, the prototype was created to illustrate the basic interaction steps for creating an annotation scheme via a graphical user interface. The prototype allows the user to create and delete annotation items on two different hierarchical levels (*parent elements*, e.g. annotation level "parts of speech", and *child elements*, e.g. annotation items such as "noun, verb, etc.").

Limitations The prototype does not allow the user to modify the value or order of the annotation items.

JavaScript libraries This prototype utilizes the *dialog* widget from the jQuery UI library, and more specifically adopts and enhances the following online example: `https://jqueryui.com/dialog/#modal-form`

Online demo `https://dl.dropboxusercontent.com/u/4194636/prototypes/P41-Integrated-Annotation-Scheme-Editor.html`

E.4. Prototype for P4.2

Functionality Although one tool example exists for this pattern, the prototype was created to illustrate the basic interaction for manipulating the order of annotation items in an annotation scheme. The prototype allows the user to rearrange a set of predefined annotation items by means of drag-and-drop. Items can also be moved from one annotation level to the other.

Limitations The prototype only implements two exemplary levels of annotations, and provides a static set of predefined annotation items.

JavaScript libraries This prototype utilizes the *tabs* widget as well as the *sortable* interaction from the jQuery UI library, and more specifically adopts and enhances the following online example: `https://jqueryui.com/sortable/#connect-lists-through-tabs`

Online demo `https://dl.dropboxusercontent.com/u/4194636/prototypes/P42-Organization-Annotation-Scheme.html`

E.5. Prototype for P4.4

Functionality Although two tool examples exist for this pattern (without a specific relation to a documented strength), the prototype was created to illustrate the basic interaction behavior for setting individual colors for different annotation items by means of a color picker.

Limitations The prototype only allows the user to set the color for three exemplary annotation items. The color changes are also displayed in a fragment of a primary data document.

JavaScript libraries This prototype utilizes the third party jQuery plugin *ColorPicker*: `http://www.eyecon.ro/colorpicker/`

Online demo `https://dl.dropboxusercontent.com/u/4194636/prototypes/P44-Facilitated-Distinction-Colorpicker.html`

E.6. Prototype for P5.8

Functionality Although one tool example exists for this pattern, the prototype was created to illustrate the basic interaction behavior for a delete function that is integrated in the annotation process. The prototype allows the user to open a context menu for an annotated anchor. The context menu displays three exemplary annotation values, which may be deleted via a small trashcan icon. If all annotation values are deleted, the anchor is no longer highlighted in the primary data (highlight indicates one or more associated annotations).

Limitations The prototype only implements annotations for one predefined anchor.

JavaScript libraries This prototype utilizes the *dialog* widget as well as the *buttons / icons* widget from the jQuery UI library: `https://jqueryui.com/button/#icons`

Online demo `https://dl.dropboxusercontent.com/u/4194636/prototypes/P58-Integrated-Delete.html`

E.7. Prototype for P6.3

Functionality This pattern is the only pattern that was derived merely from literature review, and which has no concrete problems or strengths. Accordingly, a prototype had to be implemented to provide an example of the suggested solution. The prototype implements an alternative view for the visualization of coreference annotations. In the left pane, two static documents and corresponding antecedent annotations are listed in a tree view. The tree may be expanded or collapsed for both documents. In the right pane, an alternative graph view for the display of existing coreference annotations for the selected antecedent is displayed. Grey arrows indicate coreference annotation within the same document, blue arrows indicate annotations in another document. Clicking on a "referent" node reveals the full sentence in which the referent occurs.

Limitations The prototype only implements one exemplary coreference graph view for a generic antecedent.

JavaScript libraries This prototype utilizes the jQuery *TreeView* plugin to implement the tree view pane: `http://bassistance.de/jquery-plugins/jquery-plugin-treeview/`. In addition, the D3 (Data-Driven Documents) JavaScript library for the interactive visualization of data documents was used to implement the graph view.

More specifically, the following D3 online example for *directed force graphs* was adopted and enhanced: `http://bl.ocks.org/mbostock/1153292#index.html`

Online demo `https://dl.dropboxusercontent.com/u/4194636/prototypes/P63-Alternative-Coreference-Visualizations.html`