Linguistic editing support

Piotrowski, M ; Mahlow, C

DOI: https://doi.org/10.1145/1600193.1600240

Linguistic Editing Support

Michael Piotrowski Cerstin Mahlow
Institute of Computational Linguistics, University of Zurich
Binzmühlestrasse 14
8050 Zurich, Switzerland
{mxp,mahlow}@cl.uzh.ch

ABSTRACT

Unlike programmers, authors only get very little support from their writing tools, i.e., their word processors and editors. Current editors are unaware of the objects and structures of natural languages and only offer character-based operations for manipulating text. Writers thus have to execute complex sequences of low-level functions to achieve their rhetoric or stylistic goals while composing. Software requiring long and complex sequences of operations causes users to make slips. In the case of editing and revising, these slips result in typical revision errors, such as sentences without a verb, agreement errors, or incorrect word order. In the LingURed project, we are developing language-aware editing functions to *prevent* errors. These functions operate on linguistic elements, not characters, thus shortening the command sequences writers have to execute. This paper describes the motivation and background of the LingURed project and shows some prototypical language-aware functions.

Categories and Subject Descriptors: I.7.1 [Document and Text Processing]: Document and Text Editing; I.7.2 [Document and Text Processing]: Document Preparation; I.2.7 [Natural Language Processing]: Language parsing and understanding

General Terms: Design, Human Factors, Experimentation.

Keywords: Language-aware editing, computational linguistics, revising, authoring, action slips, cognitive load.

1. INTRODUCTION

Revising and editing are crucial for the production of high-quality texts—but they are also tedious and error-prone. Authors think about their texts in terms of high-level language units, such as phrases, sentences, or paragraphs, but word processors for the most part only provide low-level character-oriented functions. Thus, when editing or revising texts, authors must translate their high-level goals (e.g., changing a main clause into a subordinate clause) into a complex sequence of low-level word processor operations. This poses a high cognitive load on authors; since writing is already a complex task, this risks overloading cognitive resources [7, 15]. It also requires authors to concentrate on small areas of the text; ultimately, this results in typical revision errors, as illustrated by the following example from an actual text:[1]

(1) Zusätzlich ist zu berücksichtigen, dass das Werkzeug **hat** natürlich auch Einfluss auf den Prozess **hat**.[2]

The verb *hat* occurs twice, whereas it should only occur once, at the end of the sentence. It is very probable that this is a revision error: German uses verb-second (V2) word order in main clauses, but verb-final (SOV) word order in subclauses. Thus, the original version likely read:

(2) Das Werkzeug **hat** natürlich auch Einfluss auf den Prozess.

When changing sentence 2 into sentence 1, the author probably forgot to remove the verb in the original (first) position. Further examples for typical revision errors are sentences without finite verb, extraneous conjunctions, misplaced determiners, and agreement errors. While some of these errors can be detected by grammar checkers, it would be desirable to *prevent* them in the first place.

In the next section, we will briefly discuss causes for such errors. We will then outline previous and related work on preventing errors in writing. In the rest of the paper we will describe the LingURed project[3], in which we are developing language-aware editing functions to reduce the cognitive load on authors and to prevent editing and revision errors.

2. ERRORS IN WRITING

When we abstract from the specifics of writing, editing and revising texts using a word processor effectively means operating a complex tool (the word processor) with the intention of achieving specific goals. On this level, editing can be compared to the operation of other complex systems, such as when driving a car, controlling a power plant, or programming a video recorder. It is well known that people make errors with such systems.

Norman [10] introduced a widely accepted classification of errors; the editing errors shown above can be interpreted as *action slips*. A slip is "the error that occurs when a person does an action that is not intended" [10, p. 1]. Slips can be further classified into subcategories; in the context of editing, the most relevant classes are *misordering the components of an action sequence*, *capture errors*, and *forgetting an intention*. As Norman [11] shows, many slips

DocEng'09, September 16–18, 2009, Munich, Germany.

[1] As we are currently focusing on German, the following examples are in German.

[2] Free translation of the intended sentence: 'In addition, one should consider that the tool obviously has an influence on the process.'

[3] LingURed stands for "Linguistically Supported Revising and Editing"; see also `http://www.lingured.info/`.

result from bad design and are preventable by more appropriate design.

The editing commands of today's word processors operate on very small units, namely characters. To achieve a high-level goal, such as changing a main clause into a subordinate clause (see examples 2 and 1), the author has to combine these commands in the correct way and execute a long, complex sequence of low-level editing commands. Devising and executing these command sequences poses a high cognitive load on authors: There is thus the risk that authors execute the commands in the wrong order or skip steps, that they inadvertently execute the (often similar) sequence for a different goal, or that they forget what they had intended to do.

Many of these slips could be prevented by reducing the cognitive load on the author. One important way to reduce the cognitive load is by shortening the command sequences required to achieve a certain goal. Shorter command sequences can be achieved by operating on units larger than single characters. These larger units should also be more closely related to the object authors are working on, i.e., words, clauses, phrases, and other language objects. Editing functions which operate on language objects can thus reduce the translation effort for the author, simplify the operationalization of goals, and shorten the resulting command sequences.

3. LANGUAGE-AWARE EDITORS

As we have noted above, editing functions in current word processors are primarily character-based. However, natural-language text is not merely a string of characters, but it is marked by underlying linguistic (morphologic, syntactic, rhetoric) structures. While there are some important differences between natural languages and programming languages, there are many similarities in this respect. Unlike word processors, however, modern source code editors are aware of the structural properties of the language and use this awareness to support programmers in the creation and editing of programs. Well-known examples of *language-aware editors* [13] include Emacs and Eclipse.

The similarities between programming and writing have already been pointed out in the 1980s [6, 12]. Nevertheless, while software developers have various "power tools" at their disposal that make the writing of computer programs more efficient, authors of texts still do not enjoy the support of such tools.

The DocEng proceedings from recent years show there is much research on formal document structures (in particular XML) and metadata and on enforcing, transforming, and generally processing these structures. However, there is very little research on supporting the creation of the textual content *inside* these structures, neither in document engineering nor in computational linguistics. In fact, even though more people than ever before are writing and editing on computers, there seems to be very little newer research on tools for text editing in general, with the exception of text entry methods for mobile devices and users with disabilities. It actually seems that text editors are currently *losing* editing support: Mobile devices, Web text entry fields, and Web-based editors offer little more functionality than inserting and deleting characters and cutting and pasting spans of characters.

Composition research focuses on the analysis of high-level writing processes (e.g., phases) and on recommendations for writers. With respect to tools used for composing there is research on how to use *available* editors and writing aids (see, e.g., [3, 16]) to support the writing process, but there is no current research on how to design better editors or on functions that would help writers to work better.

There was some research on language-aware editing support in the 1980s and 1990s, which also examined the application of concepts from software engineering and development tools to the production of natural-language documents, e.g., [5, 6, 14, 17]. Two of the main problems at the time were that computers were not fast enough for natural language processing in interactive environments (as it is computationally much more demanding than processing programming languages), and that linguistic components were not readily available or not of sufficient quality.

Finally, it should be noted that language-aware editing functions differ conceptually from grammar checkers. The former aim to make it easier to manipulate the text (the process of writing), while the latter check the results (the product of writing). Ideally, both types of functionality would complement each other.

4. THE LINGURED PROJECT

In the LingURed project we are aiming to create a variety of language-aware editing functions and to explore their utility and usability; the first goal is to develop tools that simplify certain frequent, yet complex, editing operations by defining them on the level of linguistic units. Our work is motivated by the considerations outlined in sections 1 and 2, and by the work on language-aware editors described in section 3.

Our target language is German. As a highly inflectional language, it has a number of linguistically challenging phenomena which do not exist in English. Our target group are *experienced writers*, i.e., writers who are native speakers of German, who write on a professional level, who are used to write with word processors, and who have an extensive repertoire of stylistic devices for achieving their communicative goals. Thus, these writers are not looking for linguistic or stylistic guidance, but for efficient tools to help them realize their ideas quicker and with fewer errors.

To be able to assess the usefulness of language-aware editing functions, they need to be used by writers for actual tasks in their real working environment. We therefore do not create a new editor, but rather extend an existing text editor, viz. XEmacs[4]. XEmacs and other Emacs variants are used intensively by many experienced writers, who use it for *all* of their authoring and editing needs (papers, reports, e-mail, etc., as well as programs). The Emacs architecture is also especially well-suited as a test bed, as it allows to easily add new functionality, to seamlessly combine it with existing functions, and to quickly adapt it in response to user feedback (see [2] for an overview of the Emacs architecture).[5]

4.1 Language-Aware Editing Functions

Since there is no research on the actual use of editing functions by authors, we are using the functionality of source code editors as an inspiration for potentially useful functions. We distinguish three types of language-aware functions: Functions that give information about elements of the text, functions for navigating through the text, and functions for actually changing the text. These types of functions can be outlined as follows:

Information functions. Programmers are used to so-called *syntax highlighting* in source code editors: Certain elements and structures of the programming language are highlighted (by using different colors or fonts) or indented by the editor. Natural languages also have certain key elements and structures. Highlighting these structures may improve the overview when editing and revising. Language-aware information functions can also give information about certain aspects of a text, such as the frequency of certain

[4] `http://xemacs.org/`

[5] As a feasibility study, some functions have also been successfully implemented as Microsoft Word add-ins. However, add-in development is complex, so that Word is unsuitable for experimentation.

POS, variants of multi-word terms, etc. These functions do not change the text.

Movement functions. Texts consist of words, phrases, clauses, sentences, and paragraphs. These elements represent different levels; phrases consist of words, clauses consist of phrases, etc. Language-aware movement functions enable writers to move the cursor to the previous or next instance of an element on a certain level, e.g., to the beginning of the next clause, or to jump to a specific location. These functions also do not change the text.

Operations are functions that change the text. Language-aware operations are the type of functions that can actually shorten the sequence of commands needed to realize a certain intention. These functions operate on linguistic elements and can be used to manipulate (i.e., delete, copy, paste, or change) these elements. Operations allow the writer to select an element or structure and to determine the action to be performed, e.g., moving or modifying it. Modifying a linguistic element or structure means changing its grammatical features, e.g., pluralizing a complex noun phrase.[6]

All language-aware editing functions require some amount of linguistic knowledge. The exact requirements vary and depend on the task to be solved. For example, pluralizing an entire noun phrase and the verb forms governed by it requires syntactic analysis as well as morphologic analysis and generation, whereas swapping the elements of a conjunctive construction only requires a concept of orthographic words. The functions successively build on each other, i.e., movement functions rely on information functions, and operations utilize movement functions and information functions.

4.2 Implementation

In this section we will describe some of the functionality we have already implemented; the LingURed project is still ongoing, so this is just a small sample of what that we intend to eventually make available. We plan to implement functions for frequent editing operations as well as functions for less frequent, but complex editing operations; the overall goal is to prevent slips by reducing cognitive load on the writer.

The functions are implemented by combining functionality already available in XEmacs with external linguistic resources such as part-of-speech taggers or morphologic analyzers and generators. Since natural language processing is still relatively slow and unreliable, especially on large and unrestricted texts, language analysis is kept as simple and localized as possible, while trying to maximize its usefulness. Each area of functionality is implemented as a separate package, so authors can choose to load only the functions they want to use.

4.2.1 *Transposing Conjuncts*

One of the currently implemented functions provides support for swapping the elements of conjunctive constructions, e.g. to allow authors to easily change *teachers and students* into *students and teachers*, but the elements of the conjunction (the *conjuncts*) are not limited to single words. This task has some interesting properties: Even though it is conceptually simple, it requires a complex sequence of commands in today's word processors; for example, the optimal command sequence in Microsoft Word requires eight steps [9]. Furthermore, research shows that writers rarely use "optimal" command sequences, but typically more complex ones [1]. Authors

[6]In English, pluralizing a noun phrase may seem a simple task easily solvable with regular expressions, but in German—a highly inflectional language—agreement restrictions and stem alternations in the involved word forms have to be taken into account, which cannot be implemented using regular expressions.

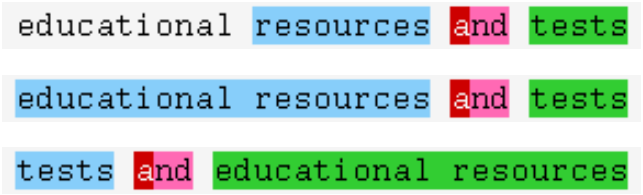

Figure 1: Transposing conjuncts using `conjunct-mode`

are thus likely to make errors during this operation, and, especially if it is only executed rarely, they are prone to forget steps. Finally, such changes are often stylistically motivated. Stylistic decisions are frequently taken after comparing different variants and considering their effect in context. This is therefore a situation in which authors could especially profit if they could easily "play" with their words without the risk of introducing errors.

Our implementation works as follows: To swap two elements around a conjunction, the author places the cursor on the conjunction and invokes `conjunct-mode`; the conjunction and the words immediately to the left and right of the conjunction are highlighted (see figure 1, top). The normal editing keys are disabled in `conjunct-mode`; instead special key bindings are available to extend, move, or shrink the selection for the left and right conjuncts word by word (see figure 1, middle). Once satisfied with the selection, the author presses t to transpose the conjuncts (see figure 1, bottom); pressing t again reverts the transposition. The extent of the conjuncts can always be readjusted. Once satisfied with the result, the author exits `conjunct-mode` with the customary C-c C-c. As usual in XEmacs, all changes can be reverted using `undo`.

`conjunct-mode` is an example of a language-aware function which can be built with standard XEmacs functionality. Note that language awareness does not necessarily require external linguistic resources: In this case, the function is based on basic structural properties of a language, which are encoded into its definition.

At first glance, it would seem desirable to automatically identify the conjuncts and to only allow operations resulting in grammatically well-formed structures. However, during editing it is frequently the case that the text is still incomplete and therefore not well-formed. NLP is also not yet able to recognize all well-formed constructions reliably. It is thus preferable to let the author determine the extents of the conjuncts. Our objective is to empower the author, not to create a syntax-directed editor[7].

4.2.2 *Integration of POS Tagging*

Many more advanced functions require part-of-speech (POS) information. We have therefore created an XEmacs interface to the MBT part-of-speech tagger [4]. This interface provides a function which annotates the word forms (using *text properties*) of a stretch of text (e.g., a sentence or a paragraph) with the POS tags returned by MBT. Text properties are a mechanism for attaching "invisible" information to characters in an XEmacs buffer. This interface provides basic functionality for accessing linguistic information on which higher-level functions can be built.

A simple example of a function that makes use of POS information is the highlighting of certain parts of speech, e.g., finite

[7]A *syntax-directed editor* [8] is a programmer's editor in which the programmer does not edit the textual representation (characters and lines) of a program, but directly manipulates the abstract syntax tree. It can thus ensure that the program is syntactically valid at all times. Syntax-directed editors were widely researched in the 1980s and 1990s, but programmers did not accept them, as they were considered too restrictive and too cumbersome to use.

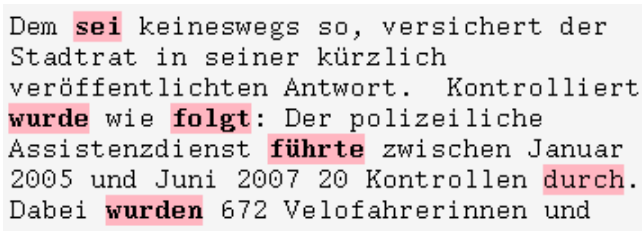

Figure 2: Highlighting of selected parts of speech

verbs. Figure 2 shows an excerpt of a text in which finite verbs are highlighted. While it is unlikely that authors have highlighting turned on at all times (as in source code editors), highlighting can be useful with complex constructions. For example, German has a large number of verbs with *separable prefixes*, i.e., prefixes which are in some cases separated from the verb; the prefix is then placed at the very end of the sentence, e.g., the verb *durchführen* occurs in the form *führte ... durch* in the extract shown in figure 2. It is thus easy to miss a separated prefix in long sentences during editing and revising; highlighting verbs can help to identify potential problems. Highlighting other parts of speech could be useful in other cases. For example, highlighting conjunctions could help to get a better overview of the argumentative structure of the text.

The availability of part-of-speech information to Emacs Lisp functions also makes it easy to implement language-aware movement functions, e.g., to position the cursor on the first finite verb of a sentence, to jump from a verb to its separated prefix, or to move to a subordinating conjunction, i.e., to the beginning of a subordinate clause, on which one might then apply some operation.

Similarly, part-of-speech information can be used for selection functions or operations, such as selecting or deleting the current phrase.

5. SUMMARY AND OUTLOOK

We have reported on the ongoing development of language-aware editing functions for German in the LingURed project. The motivation of the project is to support experienced writers during revising and editing and to prevent errors. Editing errors can be regarded as action slips. Slips are caused by inadequate design and cognitive load; by offering editing functions on the level of language units, instead of on the level of characters, we are aiming to reduce the cognitive load on writers caused by the need to translate high-level editing goals to low-level character operations. Another inspiration comes from the language-awareness available to programmers in source code editors.

We are currently working on implementing further language-aware editing functions for German. As the LingURed project is still ongoing, final results are not yet available. We are successively evaluating individual functions with writers from our target group. Once a larger set of functions is available, we are planning to conduct an overall evaluation of the usability and usefulness of the approach. At a later stage, we will explore the adaptability of the functions to other languages.

References

[1] R. B. Allen and M. W. Scerbo. Details of command-language keystrokes. *ACM Trans. Inf. Syst.*, 1(2):159–178, April 1983.

[2] J. Blandy. GNU Emacs: Creeping featurism is a strength. In D. Spinellis and G. Gousios, editors, *Beautiful Architecture*, chapter 11, pages 263–277. O'Reilly, Sebastopol, CA, USA, 2009.

[3] A. M. Buck. The invisible interface: MS Word in the writing center. *Computers and Composition*, 25(4):396–415, 2008.

[4] W. Daelemans and A. van den Bosch. *Memory-Based Language Processing*. Studies in Natural Language Processing. Cambridge University Press, Cambridge, UK, September 2005.

[5] R. Dale and S. Douglas. Two investigations into intelligent text processing. In M. Sharples and T. van der Geest, editors, *The New Writing Environment: Writers at Work in a World of Technology*, chapter 8, pages 123–145. Springer, 1996.

[6] R. Hamlet. A disciplined text environment. In J. C. van Vliet, editor, *Text Processing and Document Manipulation: Proceedings of the International Conference*, pages 78–89, Cambridge, UK, April 1986. Cambridge University Press.

[7] R. T. Kellogg. Attentional overload and writing performance: Effects of rough draft and outline strategies. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 14(2):355–356, April 1988.

[8] A. A. Khwaja and J. E. Urban. Syntax-directed editing environments: issues and features. In *SAC '93: Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing*, pages 230–237, New York, NY, USA, 1993. ACM.

[9] C. Mahlow and M. Piotrowski. Linguistic support for revising and editing. In A. Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing: 9th International Conference, CICLing 2008, Haifa, Israel, February 17–23, 2008. Proceedings*, pages 631–642, Heidelberg, 2008. Springer.

[10] D. A. Norman. Categorization of action slips. *Psychological Review*, 88:1–15, 1981.

[11] D. A. Norman. Design rules based on analyses of human error. *Commun. ACM*, 26(4):254–258, 1983.

[12] V. Quint, M. Nanard, and J. André. Towards document engineering. In R. Furuta, editor, *EP 90: Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, pages 17–29, Cambridge, UK, 1990. Cambridge University Press.

[13] S. P. Reiss. Software tools and environments. *ACM Comput. Surv.*, 28(1):281–284, 1996.

[14] M. Sharples, J. Goodlet, and L. Pemberton. Developing a writer's assistant. In N. Williams and P. O. Holt, editors, *Computers and Writing: models and tools*, chapter 3, pages 22–37. Intellect Limited, 1989.

[15] M. Torrance and G. Jeffery, editors. *The Cognitive Demands of Writing: Processing Capacity and Working Memory Effects in Text Production (Studies in Writing, Volume 3)*. Studies in Writing. Amsterdam University Press, 1999.

[16] M. Torrance, L. Van Waes, and D. W. Galbraith, editors. *Writing and Cognition: Research and Applications (Studies in Writing, Volume 20)*. Studies in Writing. Elsevier Science, 1 edition, January 2007.

[17] N. Williams. Computer assisted writing software: RUSKIN. In N. Williams and P. O. Holt, editors, *Computers and Writing: models and tools*, chapter 1, pages 1–16. Intellect Limited, 1989.