# Opportunities and Limits for Language Awareness in Text Editors

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

## Abstract

In this paper we argue that the concept of *language awareness*, as known from programmer's editors, can be transferred to writing natural language and word processors. We propose editing functions which use methods from computational linguistics and take the structures of natural languages into consideration. Such functions could reduce errors and better support writers in realizing their communicative goals. We briefly compare characteristics of programming languages and natural languages and their processing tools with respect to their suitability for being used in language-aware functions in editors. However, linguistic methods have limits, and there are various aspects software developers have to take into account to avoid creating a solution looking for a problem: Language-aware functions could be powerful tools for writers, but writers must not be forced to adapt to their tools.

## 1 Introduction

Writing is a daily task for a great number of people. However, today's word processors offer only limited support for writing and editing: Most functions are character-based and thus force writers to translate high-level goals into low-level functions of the editor. This causes typical errors, e.g., missing verbs, agreement errors, or wrong word order. Functions improving the "brain-to-hand-to-keyboard-to-screen-connection" (Taylor, 1987, p. 79) as proposed by Dale (1989; 1996) or Mahlow and Piotrowski (2008) could help avoid several types of errors. Additionally, as cognitive resources are limited (McCutchen, 1996; Allen and Scerbo, 1983), language-aware functions could reduce the effort needed to deal with word processors and help writers concentrate on their actual goals and remain in control of their text. Writers should get interactive support during writing, very similar to the support programmers get from their editors during programming.

We will first describe the principles of language awareness as they can be deduced from respective functions in programmer's editors and have a look at the current situation in word processors. Then we propose interactive editing functions operating on linguistic elements and making use of tools and methods of computational linguistics. From the comparison of characteristics of programming languages and natural languages and the hence resulting quality of the respective language processing tools we will deduce opportunities and limits language technologists have to be aware of when implementing language-aware functions for word processors.

## 2 Language Awareness in Editors

Both text written in natural languages (such as English, German, or French) and computer programs written in programming languages (such as Perl, Python, or C) have underlying syntactic structures and are not merely strings of characters. In the context of programming languages, text editors which are aware of this structure and use this awareness to support the creation and editing of programs are referred to as *syntax-directed*, *language-sensitive*, *language-based*, or *language-aware editors* (Khwaja and Urban, 1993).

The ultimate goal of language awareness in programmer's editors is to *prevent errors* in programs, as the prevention of errors helps producing higher-quality programs. Language awareness supports programmers by giving them a better overview of programs and by providing them with editing functions operating on structural elements instead of characters or lines.

In general, we can distinguish two types of language-aware functionality: (1) *Information functions* for highlighting individual language elements and larger structures, or for displaying statistical information regarding certain elements, which do not change the text, and (2) *operations* for inserting, reordering, modifying, or deleting elements, i.e., functions changing the text. Both types of functions operate on the elements defined by the lexicon and the morphological and syntactical rules of a concrete language.

Just as we can distinguish between formal languages (in this case: programming languages) and natural languages, we can distinguish between two major types of editors: Editors intended for writing computer programs (*programmer's editors*) and editors intended for writing natural-language text (usually referred to as *word processors*). These two types of editors can be seen as two instances of the general class of text editors, each adapted to handle writing, editing, and revising in specific languages.

We will now briefly analyze language awareness in these two types of editors.

### 2.1 Language Awareness in Programmer's Editors

Programmer's editors generally implement both information functions and operations. Many editors support different programming languages through language-specific editing *modes*, which are either activated automatically or can be selected by the user.

Syntax highlighting is the most prominent instantiation of an information function: Keywords, variable names, and specific constructs can be highlighted using different colors or fonts. Programmer's editors generally also help to ensure that parentheses are properly nested, e.g., by highlighting mismatches.

As instantiations of language-aware operations we can typically find functions for deleting elements, e.g., parenthesized expressions or comments, for inserting or completing syntactic structures, such as conditional expressions or looping constructs, and for selecting certain syntactic elements (e.g., the current function definition) for a subsequent operation. Some editors offer code completion, i.e., the editor can complete an initial string typed by the user, either automatically or upon request. The editor may take the context into account; for example, in an object-oriented programming language it may consider only the names of those methods available for the particular object.

Programmer's editors also indent lines automatically according to the syntax and may control the insertion of whitespace and newlines (e.g., around operators or after block-opening braces). In some languages, such as Python, indentation serves to indicate the block structure of the code. For languages like Perl, C, Java, or Lisp, indentation is not mandatory but conventionally reflects the syntactic structure, which is also marked by parentheses or braces.

### 2.2 Language Awareness in Word Processors

Since programmer's editors support developers with specific functions for the programming language being used, word processors could be expected to offer specific functions depending on the language the writer is using.

However, unlike programmer's editors, word processors offer very few language-aware functions: Almost all functions are based on characters and lines. Thus, even state-of-the-art word processors offer only a basic set of core operations (e.g., select, cut, copy, paste, insert) (Piolat, 1991, p. 262), (Sharples and Pemberton, 1990, p. 49), regardless of the language the writer is using.

Checkers for spelling, grammar, and style, which are nowadays available for various languages in many word processors, provide a certain level of "language awareness." However, regardless of their quality (Vernon, 2000; McGee and Ericsson, 2002), they are essentially tools for *post-writing*: After a draft is finished, they can detect errors and propose modifications, but they generally do not support writers *during* writing and editing, and thus do not help to *prevent* errors.

This situation clearly is disappointing. Considering the fact that there already exist sophisticated natural-language-processing methods and tools, we think the time has come to add language-aware functions to word processors as well.

## 3 Language-Aware Editing Functions in Word Processors

Writers should receive interactive support from their word processors, similar to the interactive support programmers get. Supporting writers during the writing and editing process reduces the cognitive load and therefore helps avoiding errors.

Writers should be in control of their text, relying on post-processing support only denies the fact that writing is a very active and creative process.

We propose two types of functions operating on linguistic elements, such as words, phrases, or clauses. These functions are intended to work analogously to the corresponding functions known for programmer's editors: (1) *Information functions* for highlighting elements, such as verbs or PP-attachments, or for providing writers with information about certain aspects of the text, such as prepositions used, sentences without verbs, or variants of multi-word expressions. Writers can interpret the results themselves and decide how to make use of them. (2) *Operations* for reordering, modifying, or deleting linguistic elements. In order to reduce the cognitive load, the number of actions necessary to reach a specific goal should be reduced drastically by combining sequences of core operations into higher-level functions closer to writers' goals and their mental model of the task. Examples would be the pluralization of an entire phrase (a complex task for morphologically rich languages as German), the reordering of conjunctions, or the replacing of words or phrases through the whole text (also a complex task for highly inflectional languages). See Mahlow et al. (2008) for more details.

Both types of functions require *linguistic knowledge* and *linguistic resources*. Linguistic knowledge will influence the ideal combination of existing core operations into higher-level functions a user can call with one keystroke: Reordering conjuncts is a highly complex task if a writer has to find the sequence of core operations on their own; using *one* operation reduces the risk of producing ungrammatical conjuncts. Linguistic resources will be needed for operations that modify certain linguistic elements: Pluralization of entire phrases will obviously require morphological analysis and generation.

## 4 Natural and Programming Languages

It is clear that there are significant differences between programming languages and natural languages. Two important differences are:

1. The lexicon: The lexicon of programming language is small and essentially closed. The lexicon for a natural language is much bigger and can be extended *ad infinitum* by morphological processes.
2. The syntactical rules: Syntactical rules for formal languages are made *a priori*, i.e., prior of creating a language. Users are not allowed to change the rules. Syntactical rules for a natural language, however, try to describe the phenomena of a certain language *a posteriori*. Natural languages "live," i.e., users change the rules as they are using the language – generally, native speakers are not even aware of the rules. Linguists can only discover and adapt the rules of a language *afterwards* by observing the language.

Thus, as the lexicon of programming languages is relatively small and – most importantly – closed, functions for highlighting keywords, can be implemented relatively easily. There are strict rules for extending the "lexicon" of a language with variable names (e.g., a name for a hash variable in Perl has to begin with a "%"), so that these can generally also be detected easily.

These properties of programming and natural languages explain the difference in performance of parsers for the respective types of languages: Processors of programming languages can be implemented easily, they are very sophisticated, work very fast and deliver satisfying and reliable results. Processors of natural language struggle with incomplete rules, ambiguities, big and always incomplete resources, their results in general are not very convincing, they need much time to deliver these results, thus making them not very attractive for interactive use – it is not acceptable for a writer to wait several seconds for a phrase to be pluralized.

However, there exist morphological and syntactical parsers for several natural languages which work quite satisfactorily for restricted phenomena or purposes. Additionally, nowadays computers have sufficient processing power to reduce the time needed to analyze word forms or generate phrases drastically compared to the situation ten years ago. We therefore propose to make use of those processing tools in word processors in a similar way programmer's editors make use of parsers for programming languages.

## 5 Opportunities and Limits

### 5.1 Opportunities

Language-aware operations using syntactical and morphological components could offer writers new ways of working creatively with their texts: With

one click they could apply changes to their texts, inspect the results, undo them, and try a different change. They could concentrate on their goal, play with words and phrases, and would not have to care about how to realize these changes, would not have to worry about forgetting one occurrence, and would not have to keep in mind that other locations may need changes because of the original change (e.g., pluralizing the subject of a sentence requires adjustment of the finite verb).

Like syntax highlighting and indentation in programmer's editors assists programmers, the highlighting of specific linguistic elements could help writers to get a better overview of the structure of their text written so far or to identify characteristics with respect to style, e.g., overuse of certain conjunctions or identical beginnings of sentences. When linguistic resources are carefully chosen and cleverly combined with the existing core functionality of word processors, and when the principles of the respective language are taken into account and the available computing power is utilized, various interesting scenarios for language-aware functionality emerge (see Mahlow and Piotrowski (2008) and Mahlow et al. (2008)).

### 5.2 Limits

While today's computers are capable of performing analyses and generation of linguistic structures fast enough to be suitable for interactive use, linguistic components usually fail to produce results that are 100% correct in terms of precision and recall. Furthermore, for most of these components it cannot be predicted whether the results will be correct. When using them as basis for language-aware functions in word processors, writers must be aware that they should not blindly trust the system to avoid frustrations similar to those often associated with checkers.

A second limit are cases where linguistic resources can deliver correct, but ambiguous results, e.g., it may not be possible to determine the exact category of a word form. The editing function then cannot be executed automatically but has to interact with the writer to resolve the ambiguity. For example, the plural of the German word *Mutter* 'mother; screw nut' may either be *Mütter* or *Muttern*, depending on which of the two meanings are intended.

A third limit is the danger of concentrating on aspects of (computational) linguistics rather than on aspects of the writing process and on writers' needs. For example, at first glance, it seems to be obvious that only operations resulting in grammatically well-formed structures should be allowed. But, on the one hand, the structure may not (yet) be completed and therefore not well-formed *before* executing an operation (e.g., when pluralizing a phrase consisting only of a determiner and an adjective, and the noun is added only after pluralizing). On the other hand, the relevant operation may be used only as one step in a complex sequence: After executing this operation more changes will be applied, and the result is not the end result (e.g., a list of word forms, clearly not a phrase, shall be pluralized, and some of these are then moved to other parts of the text).

This has also been realized during the development of programmer's editors: Especially in the 1980s and 1990s there have been many attempts at programmer's editors which are not based on characters and lines at all but where the programmer instead edits the abstract syntax tree of the program directly (Khwaja and Urban, 1993), thus ensuring that the program was syntactically valid at all times. However, programmers did not accept this type of syntax-directed editors; one important problem was that it also prohibits invalid intermediate states, making editing very cumbersome (Neal, 1987). Current programmer's editors are therefore based on the textual program representation and only provide assistance as described in section 2.1 above.

Syntactic variability may also be considered a problem for language awareness. For languages with free word order, such as German, we can have sentences like:

(1) Ich gab dem Kind gestern einen Apfel.

(2) Gestern gab ich dem Kind einen Apfel.

(3) Dem Kind gab ich gestern einen Apfel.

All syntactical variants express the same basic meaning, 'yesterday I gave an apple to the child.' Another example are passive and active versions of a sentence. Obviously syntactic variants slightly change the meaning of a sentence by changing the focus, but they still express the same main idea. Syntactic variants are one aspect of creativity in writing.

However, similar phenomena also exist in programming languages: Just as in natural languages, one meaning can be expressed in different ways.

In addition, there are also many ways to layout the code, e.g., by using more or less line breaks.

Programmer's editors provide valuable support for programmers without restricting their creativity by forcing them to use one specific syntactic structure for expressing something. This would in fact be impossible since the editor is not able to predict what the programmer has in mind. The same applies to natural-language editing.

## 6 Conclusion

We have presented the concept of language-aware functions in word processors using methods and systems from computational linguistics. They represent opportunities for supporting the writing process, but developers should avoid concentrating on technical aspects alone, expecting writers to adapt to their tools, which would cause dissatisfaction and ultimately rejection of the tools. The goal clearly must be to support writers by lowering the cognitive effort for complex operations and at the same time allowing them to define *their* goals and to be in control of their texts. This principle has to direct the implementation with respect to technology and usability. Today's state-of-the-art methods and tools for NLP and the available computing power can be used – *and should be used* – to develop language-aware functions for interactive support in word processors.

In the *LingURed* project (see `http://www.lingured.info`) we are developing prototypical implementations of various language-aware editing functions. Depending on licences for the used resources we will publish these functions under an open source licence, and we will evaluate them for usability and effectiveness together with experts in writing research.

## References


[Allen and Scerbo1983] Robert B. Allen and M. W. Scerbo. 1983. Details of command-language keystrokes. *ACM Trans. Inf. Syst.*, 1(2):159–178, April.

[Dale and Douglas1996] Robert Dale and Shona Douglas. 1996. Two investigations into intelligent text processing. In Mike Sharples and Thea van der Geest, editors, *The New Writing Environment: Writers at Work in a World of Technology*, chapter 8, pages 123–145. Springer.

[Dale1989] Robert Dale. 1989. Computer-based editorial aids. In Jeremy Peckham, editor, *Recent Developments and Applications of Natural Language Processing*, chapter 2, pages 8–22. Kogan Page Limited.

[Khwaja and Urban1993] Amir A. Khwaja and Joseph E. Urban. 1993. 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. ACM.

[Mahlow and Piotrowski2008] Cerstin Mahlow and Michael Piotrowski. 2008. Linguistic support for revising and editing. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing: 9th International Conference, CICLing 2008, Haifa, Israel, February 17–23, 2008. Proceedings*, pages 631–642, Heidelberg. Springer.

[Mahlow et al.2008] Cerstin Mahlow, Michael Piotrowski, and Michael Hess. 2008. Language-aware text editing. In Robert Dale, Aurélien Max, and Michael Zock, editors, *LREC 2008 Workshop on NLP Resources, Algorithms and Tools for Authoring Aids*, pages 9–13, Marrakech, Morrocco. ELRA.

[McCutchen1996] Deborah McCutchen. 1996. A capacity theory of writing: Working memory in composition. *Educational Psychology Review*, 8(3):299–325.

[McGee and Ericsson2002] Tim McGee and Patricia Ericsson. 2002. The politics of the program: MS Word as the invisible grammarian. *Computers and Composition*, 19(4):453–470, December.

[Neal1987] Lisa R. Neal. 1987. Cognition-sensitive design and user modeling for syntax-directed editors. In *CHI '87: Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface*, pages 99–102, New York, NY, USA. ACM.

[Piolat1991] Annie Piolat. 1991. Effects of word processing on text revision. *Language and Education*, 5(4):255–272.

[Sharples and Pemberton1990] Mike Sharples and Lyn Pemberton. 1990. Starting from the writer: Guidelines for the design of user-centred document processors. *Computer Assisted Language Learning*, 2(1):37–57.

[Taylor1987] Lee R. Taylor. 1987. Software views: A fistful of word-processing programs. *Computers and Composition*, 5(1):79–90.

[Vernon2000] Alex Vernon. 2000. Computerized grammar checkers 2000: capabilities, limitations, and pedagogical possibilities. *Computers and Composition*, 17(3):329–349, December.