User-centred design of language checking tools

Martí Quixal, Toni Badia, Francesc Benavent, Jose R. Boullosa, Judith Domingo, Bernat Grau, Guillem Massó, Oriol Valentín

GLiCom, Fundació Barcelona Media Universitat Pompeu Fabra
c/Ocata, 1, E-08003 Barcelona, Spain
{marti.quixal,toni.badia}@upf.edu

Abstract

This paper presents a methodology for the design and implementation of user-centred language checking applications. The methodology is based on the separation of three critical aspects in this kind of application: functional purpose (educational or corrective goal), types of warning messages, and linguistic resources and computational techniques used. We argue that to assure a user-centred design there must be a clear-cut division between the 'error' typology underlying the system and the software architecture. The methodology described has been used to implement two different user-driven spell, grammar and style checkers for Catalan. We discuss that this is an issue often neglected in commercial applications, and remark the benefits of such a methodology in the scalability of language checking applications. We evaluate our application in terms of recall, precision and noise, and compare it to the only other existing grammar checker for Catalan, to our knowledge.

1. Introduction

Traditionally language checking (including error checking) is divided in two big tasks: isolated-word checking and word-in-context checking. The former is useful for those language fragments resulting in non-words (like in spelling errors, *hte* vs. *the*), and the latter is necessary to handle language fragments resulting in real correct words (like the spelling error *form* vs. *from*, or the usage error in *I work in/*at the Marketing Department*). As for error correction, until the early 1990's most research was devoted to the first task (Kukich, 1992), but from then on research has been centred on the checking of grammar, style or content errors in context (Vosse, 1992; Schneider and McCoy, 1998; Gojenola and Oronoz, 2000; L'Haire and A., 2003, to mention but a few). Context-sensitive language checking techniques have also been reflected in the availability of end-user applications, specially of grammar and spell checkers –commercial or not. We will centre the presentation on error checking software, which is the kind of software developed following the methodology presented.

Industrial spell, grammar and style checkers tend to separate the detection of isolated-word errors and word-in-context errors as two different functionalities –a separation usually imposed by the software architecture. These functionalities are often described as 'spell checking' and 'grammar (and style) checking', which often misleads the user to think that errors detected with the technique needed for the detection of word-in-context errors are always grammar and style errors. For instance, the Spanish grammar checker included in MS Word 2003 handles the confusion between the words *a* (a preposition often translated by to) and *ha* (the third person singular form of the auxiliary verb *haber*) with its context-sensitive error correction module. When the system detects an instance of this type of error, it presents it to the user as a grammar error, when in fact it is an orthographical error (in the strict sense). Though this might seem an irrelevant issue –and certainly is so if the question is simply the colour used to highlight the error–, it is not so irrelevant if the user can activate and deactivate 'grammar checking'. She believes to be deactivating the detection of grammar errors, but she is deactivating the detection of word-in-context errors, including some orthographical errors.

We present a methodology for the design and implementation of user-centred error correction tools, which has been applied for the development of COTiG, described below. With this methodology we are able to grant real customization facilities to corporate or institutional users of error correction (or controlled-language) software. The application does not limit the graphical representation (the 'painting') of errors, and it allows for a functionality-based activation and deactivation of the types of warnings.

2. Designing and implementing a user-adaptive error correction application

A user-adaptive error correction application must provide users with the possibility to customize the behaviour of the software architecture and the graphical interface to the criteria posed by the writing tasks performed. To grant customization, a clear-cut division of the three following aspects is required: the application's functional purpose, the design of an appropriate error typology, and the separation of linguistic resources and computational algorithms used. While the latter is most frequently assured by the design of modular NLP architectures, the separation between error classification/description and the presentation of errors to end-users (based on the functional purpose) is not always taken into account.

Figure 1 reflects the components of a development methodology that would grant the separation of the above mentioned aspects. The functional design of the application will be basically based on the error specifications (what is to be handled) and user interaction requirements (how is the information provided by the application presented to the user and which modes of interaction are provided with). The error typology determines how will be errors grouped into classes; the more fine-grained and structured it is, the more flexible and detailed will be the way in which the information can be presented to the end-user. Finally the software architecture, which is further detailed in Section 3.,

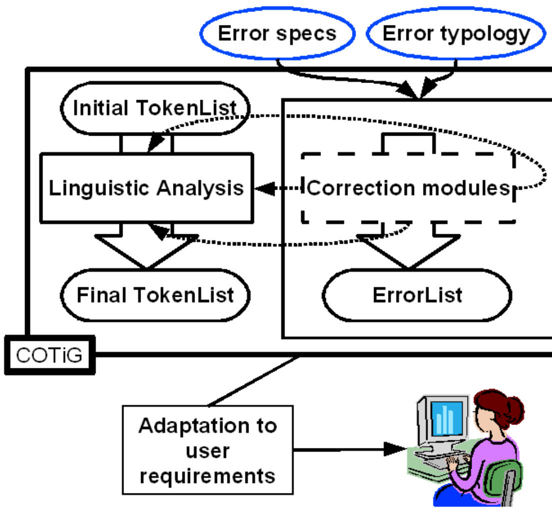

Figure 1: Specifications and software components for the design of user-centred language checkers.

which is modular, but in turn independent from the previous, too. Note that Figure 1 reflects that the list of errors (or linguistic structures) that will be later presented to the user is created during a progressive linguistic analysis process. At any stage, each correction module –represented by the three arrows grasping at the TokenList– has access to all the available linguistic information. All modules can assign any of the existing error codes, which makes error classification independent from the detection technique. For a given processing unit (paragraph, sentence, text, etc.), once all errors have been detected they are presented to the user according to functional criteria.

2.1. Designing a correction tool from the user's functional perspective

A user-centred design presupposes the definition of the functional goal of the application to be introduced. The use of an error correction application as a general purpose spell and grammar checker to be included as a support tool in standard word processors will significantly differ from the implementation of a style guide of a company in terms of functional design, and it will differ even more from an application to be used as a second language learning assistant.

The methodology we present has been used to implement a general purpose spell and grammar checker and is currently being used to implement a corporate style guide for a media company. The most critical aspects in the functional design of the underlying language checking architecture are the following:

Correction modes. A very usual division of correction modes is between typographical, orthographical, grammatical, and style errors. Imprecise as these labels may be, they are commonly used and most experienced word-processor users tend to know what they refer to. These are the general descriptions used in our proposal and they are reflected in the error typology, not in the computational techniques (see Section 3.). The latter is crucial to avoid the deactivation of errors that must be detected with context-sensitive strategies but do not 'belong' to correction modes typically or mostly detected with context-insensitve techniques. Customization of actions with respect to correction modes is granted by the definition of the error typology and the independence of this typology with respect to the error correction techniques (see Section 2.2.).

Types of actions to perform on linguistic constructions. A language checking application is usually expected to: a) validate apparently correct linguistic constructions (implicitly corrected with silence on the checker side); b) mark and give correction proposals to detected errors; and c) warn of the use of linguistic constructions that should be used cautiously. It is straightforward to solve a) and b), but not c). In our case the general purpose version of the application does not allow for customisation of warning messages related to language use. In contrast the version adapted to the media company requires flexibility in that some detections must be marked as 'errors' and others as 'warnings'. We achieve this by introducing further linguistic information in the dictionaries and by extending the error typology to differentiate must-correct errors from should-review words. Again the customization of the types of actions for each error typed is granted by the definition of the error typology together with a file that defines which are the actions to be performed for each error type or for given groups of error types (see Section 2.2.).

Regional, formal and technical linguistic varieties. Users might need to be able to write in different dialects, registers and domains both in the same text and in different texts. Since activation or deactivation of these language variants can be changed during the correction task, this is taken into account when the lexical resources are loaded. All the relevant linguistic information included in the machine-readable dictionary must be carefully included and managed during the linguistic processing. This will be specially relevant for the version adapted to the media company where different communication registers are used –where different registers require different types of actions (see previous paragraph).

These three main issues in the functional design of the error correction tool are defined externally either by a series of configuration files and the codification of words in the dictionary according to special actions related to them (see Section 3. for further details).

2.2. Defining an error typology

The definition of an error typology will be consistent with the fine-grainedness desired in the feedback. For error-specific feedback to be generated, and for error types to be grouped in general classes, a complex and detailed typology is required. We adapted and extended the error typology proposed in (Granger, 2003), conceived to describe second language learner errors, but also useful to describe native speaker errors.

Granger's typology, based on a three-level hierarchy, is used to characterise errors. The domain level, one of eight possible classes (form, morphology, lexis, grammar, register, etc.); the category level, whose classes diverge from one domain to the other (for instance, in the morphology domain we have classes such as derivation-prefixation,

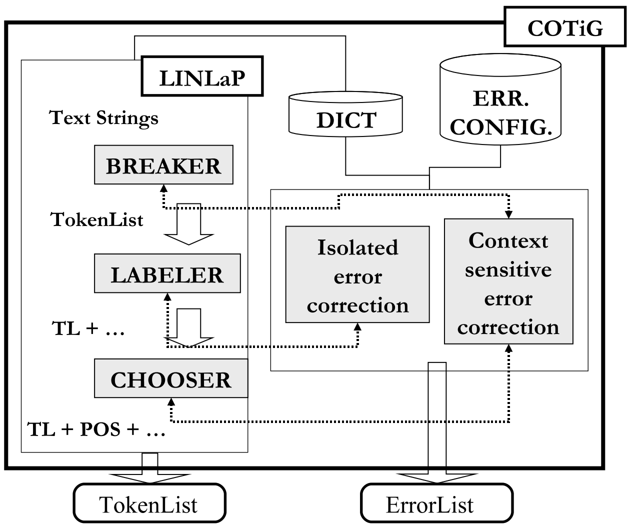

Figure 2: COTiG: a modular user-aware language checking architecture. Parsing modules and detection modules modify independent computational objects.

derivation-suffixation, inflection-confusion, etc); and finally the word category, based on a POS tagset.

We added two further description levels, which are in fact above the domain level. One of them is the error description level (or correction mode) with four possible classes, namely orthography, typography, grammar or style. The other one is the kind of transformation that determines the error: is there something missing, is there some extra word, or is there a word that should be substituted. The first of these two description levels helps us to relate error descriptions to correction modes –crucial for a general purpose error correction tool. The second one can be used to determine the fragment of text that will be underlined for the user when feedback is produced. This second feature is currently not exploited in the graphical representation of errors, though.

3. COTiG: a user-aware architecture for language checking

COTiG is based on LINLaP, a general purpose NLP architecture developed for Catalan (Benavent et al., In preparation), which in turn is inspired in previous work by (Badia et al., 2001). LINLaP is a morphosyntactic tagger consisting of a tokenizer, a dictionary lookup module, and a stochastic disambiguation module. COTiG introduces three correction modules that detect different error types based on the information available at each stage (also inspired in previous work (Badia et al., 2004)). The right hand-side part of Figure 2 shows that error detection depends on modules based in two different techniques that are invoked at several points in time during the processing. There are the modules based on isolated-word error detection, whose action is independent of the context of appearance of a word. And there are the modules whose rules are only applied if a given context is seen.

3.1. Extending the data structure of LINLaP to handle error lists

For COTiG to be a user-aware language checking architecture, it had to assure independence between the error description classes and the computational techniques used. COTiG ensures this by containing two independent data objects: the Token (TokenList), which has all the word-related pieces of (linguistic) information produced during the processing, and the Error (ErrorList) which is incrementally fed along with the linguistic processing. The architecture is inspired in so-called blackboard systems, which basically implies that all modules have access to all the available information at any point in time. Analytic modules are allowed to modify the information in the TokenList, while corrective modules are only allowed to read it and to feed the ErrorList.

The Error object includes information on the error span (starting and ending character/word), the error code assigned, the module responsible for the detection, the rule/strategy responsible for the detection and some other pieces of information useful for debugging. It also includes a list of correction proposals which is optionally filled. Once the correction process is finished a list of unordered errors is available. At this stage, errors are ordered according to the precedence, length and correction mode criteria.

3.1.1. Linguistic resources underlying the general-purpose correction engine

The general-purpose spell and grammar is checker based on prescriptive lexical and grammar specifications, in Catalan established by the Institut d'Estudis Catalans. If a word or a sequence of words is not accepted by the reference dictionaries or grammar, it will be marked and in most cases a correction proposal will be generated. It can deal with different dialects both at the dictionary and the error correction grammar level. Its linguistic resources are a general dictionary, which contains word, lemma, POS-tag and other morphosyntactic information, geolectal information and information related to case and word non-letter signs (hyphens, apostrophes, dots, etc.). It includes also a multi-word dictionary, including sequences constituting a lexical unit and that in that form have to be written according to the rules. The error dictionary has a table of erroneous words associated with its correction proposals commonly used by Catalan native speakers for which a successful correction proposal would not be generated with standard minimum edit distance (MED) algorithms. Finally, the grammar rules are context sensitive. Each rule has a description part and an action part. The description part defines in which contexts it has to be applied; the action part determines which word span will be marked to the user, which error code (hence error message, colour, action type, etc.) will be assigned to the error and a correction proposal. All these resources can be modified and updated externally without affecting the engine's modules.

3.2. Using the COTiG architecture to implement a style checker

Originally LINLaP included in its dictionary geolectal and register information. We currently designed an extension

of the register information to make the system capable of meeting the needs of the media company, which will use the tool as a corporate style checker.
In addition to the correction functionalities of the general-purpose correction engine, the style checker has to deal with five extra word groups: a) words which are commonly used and accepted in non-normative standard Catalan; b) confusable pairs of words, which are acceptable in one sense but not in others (confusion originates often in Spanish false friends or in homophony); c) words that are considered weird, obsolete or unnecessary by the style guide of the company; d) words with a restricted use (i.e. informal register as in soup operas, but not in news). The fifth group, f), is similar to the words in the error dictionary of the general purpose version of the tool –most of them coincide–, those for which a successful correction proposal would not be generated with standard MED algorithms. In all cases we mean words or word sequences, so all these changes affect both isolated-word error correction and context-sensitive error correction.
The additional functionalities described require flexibility in terms of type of action (error or warning message), how is the correction proposal generated (with a list, with MED algorithms, or hand-crafted in rules), which is the error code assigned and whether for that group of words the activation is optional or not. This flexibility was already granted for rule-based context sensitive error correction but not for isolated-word error correction. Therefore, the major change that these new functionalities imply is the separation of the MED-based spell checking module and the module for handling ad hoc corrections –group f). This new module exploits exhaustively information now coded in the error dictionary (each word group requiring different behaviour in any of the mentioned aspects is marked accordingly). In addition, we implemented the possibility to have two groups of correction proposals for a given word. This is used in the handling of confusable pairs, for which the correction proposals states something like “X is correct if you meant Y, but not if you meant Z”.
Special behaviours for any error type (linguistic characteristic) are currently defined in configuration files –reflected as Err. config. in Figure 2– that allow to determine (a) which error code/message is assigned to particular lists of words (specially coded in the dictionary or the context-dependent rules); (b) what kind of correction proposal is generated; and (c) whether for a particular correction action or mode the user is allowed to activate/deactivate it or not. Though it is not possible for the moment, behaviour in terms of regional variants could also be externally configured. The current default option is that all regional variants can be activated/deactivated in an optional non-exclusive manner, and that at least one must have been selected.

3.3. Adequacy to user needs

The evaluation of the adequacy of an error correction tool for the purposes of a corporate user is usually a time consuming task. To reduce time costs, we developed (and provide, since the resulting architecture is open source) an automatic evaluation non-graphical application (CotigEval). Following a predefined syntax, an annotated corpus can be used to estimate recall and precision measures of the overall detection capacity. In addition, total and partial accuracy and error rate measures can be produced as to the error localisation (word span marked), the error classification (provided the error typology has been adapted or introduced in the system), and the adequacy of the error proposals made. CotigEval allows to evaluate in seconds which is the performance of the system for a given evaluation corpus and would allow to assess how suitable is a given implementation for a specific purpose. In addition, it was used during the development of the current version of the general spell and grammar checker to compare improvements/effects on overall performance after significant changes in either the computational modules or the linguistic resources.

4. Evaluation and related software

To our knowledge our general-purpose correction engine and Maixgramar^{TM} are the only currently maintained existing spell and grammar checkers for Catalan.[1] We performed a manual evaluation of both engines on a small corpus (6 texts, 2300 words) from a variety of genres (newspaper articles, blog text, high school and college essays and informal e-mails). We corrected the texts using both applications and we evaluated precision and recall, as well as accuracy in terms of correction proposal.

Engine	Precision	Recall
COTiG	70%	83%
Maxigramar	56%	78%

Table 1: Precision and recall measures for both COTiG and Maxigrammar

Precision has been measured as correct hits divided by the total number of flags. Out of the incorrect flags for each of the applications, 57% of those flagged by COTiG were words missing in the dictionary. For Maxigramar these amounts to 50%, which can be accounted for by the fact that Maxigramar uses more lexical resources than COTiG, plus it includes non-prescriptive dictionaries.
As for recall has been measured as correct hits divided by the number of actual errors in the texts. As for undetected errors, we cannot say which of them are expected to be handled by Maxigramar and which not, but for COTiG we found that 52% of the undetected errors would require some kind of linguistic analysis (syntactic, semantic) which is not currently available for the NLP techinques exploited. For Maxigramar we calculated these figures in the same basis as we did for COTiG and this resulted in 47% of all undetected errors.
As for the evaluation of correction proposals, we distinguish between 4 categories: correct proposal, proposal generated but inadequate, proposal not generated, and proposal not relevant (this last category includes proposals generated for non-Catalan proper names and foreign words).
Table 2 shows that COTiG generates more correct proposals than Maxigramar, while proportionally Maxigramar is

[1]Both of them are available in the Internet in demo versions: http://www.maxigramar.com/ and http:/parles.upf.es/corrector.

Engine	CP	INAD	NG	NR
COTiG	55.8%	20%	7.4%	16.8%
Maxigramar	46.7%	24%	2.7 %	26.6%

Table 2: Accuracy of correction proposals.
CP stands for correct proposal, INAD for proposal generated but inadequate, NG for proposal not generated, and NR for proposal not relevant.

more often able to generate a proposal (failed only in 2.7% of flags vs. 7.4% for COTiG).
Generally speaking, COTiG performs slightly better than Maxigramar both in terms of error detection and proposal generation for the small corpus used. However, it must be noted that Maxigramar has a much larger coverage in terms of lexical resources and this might play a role both in the undetected real errors and in its relative noisiness in terms of correction proposals.

5. Concluding remarks

We described the design and implementation of a methodology for user-centred error correction applications. With this methodology we assure that the technique and the level of computational complexity required for the detection of an error does not affect the way this error is presented to the end-user in terms of error description or classification. A general purpose spell and grammar checker has been built and is freely distributed as open source (http://parles.upf.es/corrector) and an adaptation of the architecture to build the style checker for a media company is currently under way.
If a register error is detected with an isolated-word error correction technique, this error is shown whenever the user activates ‘mark register errors’. But this does not affect other errors detected with the same technique. Similarly, if an orthographic error is detected using a context sensitive technique, also used to detect grammar errors, orthographic errors will still be detected even if ‘detect grammar errors’ is deactivated.
We presented the results of a manual evaluation based on a small corpus comparing COTiG’s performance to another commercial Catalan spell and grammar checker and we found that COTiG outperforms the latter both in error correction and adequacy of proposal generation.

6. References

Toni Badia, Gemma Boleda, Eva Bofias, and Martí Quixal. 2001. A modular architecture for the processing of free text. In *Proceedings of the Workshop on ’Modular Programming applied to Natural Language Processing’ at EUROLAN 2001*.

Toni Badia, Àngel Gil, Martí Quixal, and Oriol Valentín. 2004. Nlp-enhanced error checking for catalan unrestricted text. In *Proceedings of Fourth International Conference on Language Resources and Evaluation*, volume VI, Lisbon, Portugal.

F. Benavent, S. Bott, B. Grau, M. Quixal, and T. Badia. In preparation. LINLaP: an open-source multiplatform language independent NLP architecture.

K. Gojenola and M. Oronoz. 2000. Corpus-based syntactic error detection using syntactic patterns. *Proceedings of the workshop on Student research workshop*, pages 24–29.

Sylviane Granger. 2003. Error-tagged Learner Corpora and CALL: A Promising Synergy. *CALICO*, 20:465–480.

Karen Kukich. 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24:377–439.

S. L’Haire and Vandeventer A. 2003. Error Diagnosis in the FreeText Project. *CALICO*, 20:481–495.

D. Schneider and K.F. McCoy. 1998. Recognizing syntactic errors in the writing of second language learners. *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and the Seventeenth International Conference on Computational Linguistics*, 2:1198–1204.

Theo Vosse. 1992. Detecting and correcting morpho-syntactic errors in real texts. In *Proceedings of the third conference on Applied natural language processing*, pages 111–118, Morristown, NJ, USA. Association for Computational Linguistics.

Acknowledgements

The research work presented here has been partially funded by the following projects: *ArquiText*, funded by the Ministerio de Educación y Cultura of the Spanish Government (HUM2004- 05321-C02-02); *El Corrector*, funded by the Generalitat de Catalunya (SE/CTTI/51/05); and *Revisor d’estil*, funded by the Corporació Catalana de Mitjans Audiovisuals.