Southern African Linguistics and Applied Language Studies 2003 21(4): 325–338
Printed in South Africa — All rights reserved

SOUTHERN AFRICAN LINGUISTICS AND APPLIED LANGUAGE STUDIES
ISSN 1607–3614

Various uses of a spelling checker project: practical experiences, teaching, and learning

Menno M van Zaanen[1] and Gerhard B van Huyssteen[2]*

[1] *Induction of Linguistic Knowledge, Computational Linguistics, University of Tilburg, Tilburg, The Netherlands*

[2] *School for Languages, Potchefstroom University for Christian Higher Education, Potchefstroom 2531, South Africa*

** Corresponding author, e-mail: afngbvh@puknet.puk.ac.za*

Abstract: In this article we give an overview of various aspects of a project developing a spelling checker for Afrikaans. We discuss two of the main aims of the project, viz. for researchers to obtain practical experience, and to further learning of both researchers and students. This article, therefore, consists of two relatively independent parts that each describes aspects related to these two aims. The first part describes the actual spelling checker. The existing spelling checker is evaluated and, based on the results, improvements are introduced and implemented. Remaining problems are discussed and possible solutions are proposed. It is illustrated how practical experience was obtained during the evolution of the project. The second part discusses the underlying teaching and learning benefits of the project. This includes improving our knowledge of computational linguistics on Afrikaans, improving programming skills for linguists, and communication between experts in different fields. We also give an overview of how the project is used for problem-oriented and project-organised educational purposes, and how it solves certain problems related to a shortage of computational linguistic tools and resources for Afrikaans.

1. Introduction

Even though South Africa has eleven official languages, very little computational linguistic research focussed on these languages in the past. Additionally, no complete computational linguistics qualification was available in South Africa.[1] It is only recently that directed research projects and coherent educational programmes in computational linguistics were established in South Africa.

In accordance with this general tendency, the *Potchefstroom University for Christian Higher Education* (PUCHE) established both a training programme and a research programme in computational linguistics. The graduate programme, BA Language Technology, was introduced in 2002, while the research sub-programme 'Language and Technology' was already instituted in 2000 within the focus area, 'Languages and Literature in the South African Context'. Within this research sub-programme, four divisions were identified, viz. Speech Technology, Text Technology, Corpora, and Subtitling. Different projects were consolidated or undertaken in each of these divisions, one of which is the Afrikaans spelling checker project, on which we report in this article.[2]

As a first major project, it was decided to update and redesign the existing spelling checker for Afrikaans (henceforth called *PUKspell*), that had been developed and marketed by the IT department of the PUCHE since 1996. Several possible benefits from this research were foreseen. First, it was considered to be a relatively easy project, by means of which researchers and students could get some useful practical and hands-on experience in the development of language technology applications. It does not only provide the opportunity to work with different natural language processing techniques and processes, but also gives insight into what problems arise when pure linguists have to consider what computers can and cannot do. Related are the (communication) problems that occur when linguists from their point of view need to interact with programmers (i.e. when working together in a

development team). Second, the fact that students are directly involved in the research project, was seen as an important aim, as this allows them to obtain knowledge about research in computational linguistics, but also in particular about developing a spelling checker. The third aim was that a practical product is developed that can be used and possibly commercially exploited. As a side-effect, other tools, such as a stemmer, a hyphenator, a lexicon, and other 'enabling' tools and resources are developed. These tools can then be reused in future projects. Lastly, this project would hopefully contribute to the status of the language, by providing the language users with a product that will make it easier for them to create high quality texts in Afrikaans.

This article is divided into two parts, based on the first two aims mentioned above. The first part gives an overview of the actual spelling checker. It describes the currently available spelling checker, which is compared to another available spelling checker for Afrikaans. Based on these results, the weaknesses and their possible improvements are evaluated. An overview of the new spelling checker, including the improvements, is also given.

In the second part of the article, different facets of the teaching and learning processes are described. The focus here is on how the development of the spelling checker is used in education and learning. It is shown that the project has two main advantages. On the one hand, it allows the students to get a good feeling of the problems that a computational linguist may encounter by helping directly with the actual development of the spelling checker. On the other hand, it gives the linguists in the team the opportunity to grow in their roles as computational linguists (while collaborating with experts in other fields, such as computer science), by learning about natural language processing techniques and methods, and even by learning how to do simple computer programming for text processing.

2. Practical experience

2.1 Introduction

There are two practical reasons why the development of a spelling checker is used in this project. First, the IT department of the university had already developed a working version of a so-called first generation spelling checker (cf. Prinsloo & De Schryver, 2003) since 1998, which has been commercially exploited successfully. This spelling checker is mainly a lookup module that compares each word to words in the lexicon.

The main problem with this approach is that the performance of the spelling checker is directly related to the comprehensiveness of the lexicon. If correct words cannot be found in the lexicon, they are considered incorrect. This problem led to the demand for a more sophisticated spelling checker.

Second, the 'old' spelling checker has been used commercially, which shows that there is an actual interest in an Afrikaans spelling checker. A new and improved version of the spelling checker will (hopefully) result in a financial profit. This profit can again be used to support further research.

In this section, we first investigate exactly what the problematic areas of the existing spelling checker are by comparing it against another commercially available spelling checker for Afrikaans. Next, we give an overview of the existing and the improved spelling checkers and discuss the implemented improvements. Finally, this section is concluded with a brief summary.

2.2 Comparison between two spelling checkers for Afrikaans

Only when we know how the existing *PUKspell* performs (also compared to other existing spelling checkers), we need to know what should be improved. We therefore compare two available spelling checkers for Afrikaans: *PUKspell* and another commercial spelling checker which we will call *Bspell*[3] (Van Huyssteen, 2002).

The approach taken here is to apply both spelling checkers to a text containing correct and incorrect words. The flags and corrections suggested by both spelling checkers are evaluated against the correct text. This evaluation is done using a set of metrics. The next section will describe which metrics are used and also discuss some of the problems that remain when evaluating spelling checkers in this way. It concludes with a discussion of the results of the evaluation phase.

2.2.1 Evaluation metrics

There are many aspects of spelling checkers

that can be evaluated. In this article, we divide the aspects into two parts: *user-friendliness* (e.g. completeness and readability of the manual, ease of use and installation, etc.), and *performance* (i.e. recall, precision, and suggestion adequacy).

User-friendliness is hard to measure. In the evaluation performed here, we found that both spelling checkers are comparable with regard to user-friendliness. Both spelling checkers perform reasonably. Based on this, no preference for one of the two spelling checkers could be made. The remaining part of the evaluation in this article will therefore focus on more directly measurable metrics.

Table 1 gives an overview of the more directly measurable metrics that will be used here. It is useful to define some terms here:

- *Valid words* are words that are part of the language, or which are sanctioned by the language system, in contrast to *invalid words*, which are *not* part of the lexicon or language system.
- *Flagging* a word is when the spelling checker claims a word is invalid, while *accepting* a word means treating it as valid. Accordingly, a *flag* is an indication that a word has been tagged as invalid (regardless if the word really was invalid or not).
- *Suggestions* are alternative valid words that are offered to the user to replace a flagged word with (Paggio & Music, 1998).

Lexical recall gives the percentage of valid words correctly accepted by the spelling checker, error recall indicates the percentage of invalid words correctly flagged and precision gives the percentage of correct flags (correctly found invalid words) over all flags by the spelling checker. With all metrics it holds that numbers are higher when performance is better.

There is a difference between a spelling checker and a spelling corrector. This difference is often recognised, but in practice hardly ever made. A spelling checker searches text to find spelling errors. A spelling corrector not only finds spelling errors, but also suggests possible corrections for the errors.

In practice, most spelling checkers are actually spelling correctors.[4] From this point of view, we should also evaluate the suggestion accuracy of the spelling checkers. However, the development of an improved suggestion module is currently still under construction and will not be treated in this article. Some more information on the results on the suggestion capabilities of the spelling checkers can be found in Van Zaanen and Van Huyssteen (2003).

2.2.2 Evaluation problems

When evaluating spelling checkers, it should be clear that the specific results obtained depend heavily on the used texts. Different texts will generate different results, since a number of words in the text are (or are not) part of the lexicon. This is not the only variable in the evaluation of spelling checkers. This section discusses some of the notions and problems of the

Table 1: Evaluation metrics for evaluation of spelling checkers

Metric	Method of measurement
Lexical recall	$\frac{\text{\# valid words accepted}}{\text{\# valid words}}$
Error recall	$\frac{\text{\# invalid words flagged}}{\text{\# invalid words}}$
Precision	$\frac{\text{\# correctly flagged invalid words}}{\text{\# words flagged}}$
Suggestion adequacy	$\frac{\text{\# correct suggestions for flagged, invalid words}}{\text{total \# of flagged, invalid words}}$

evaluation of spelling checkers.

The evaluation approach taken in this article is as follows. A text, called the *test text* is given to the two spelling checkers. This text contains text with valid and invalid words. The number of found valid and invalid words of both spelling checkers is counted and these counts are then used to determine the performance effectiveness of the spelling checkers, by using the formulas in Table 1.

The recall and precision results of the spelling checkers are only comparable when they are applied to the same text. This implies that comparing the results obtained in this article to results given in other publications (that are generated by applying spelling checkers on different texts) are not directly comparable. This also happens, only worse, when comparing spelling checkers of different languages.

Another problem is related to the composition of the test text. The test text should contain known valid and invalid words. If a large text is written for this, it may contain errors itself (because it has been written by hand or extracted from a corpus). Extreme care has to be taken to make sure that all spelling errors are known. A related problem is to decide what exactly is being evaluated. One could concentrate on specific features of a spelling checker (e.g. its capability to handle productive compounding, proper names, abbreviations, or other morphologically complex words). Examples of these features should then be contained in the test text. All the choices about the construction of the test texts should be taken into account when computing (and comparing) the recall and precision of the spelling checkers.

Currently, most (commercial) spelling checkers are unable to handle context-sensitive spelling correction. We consider the correction of multi-word units (such as fixed expressions or idioms) as context-sensitive, as well as the correction of words in context such as *wie* ('who') or *wat* ('which' – used as relative pronouns), or homonyms such as *vlei* ('flatter') and *vly* ('lay down'). We have not evaluated these types of errors, but in a more general evaluation, this could be taken into account.

In the current evaluation, we have applied the spelling checkers to a text containing 1 337 different words (i.e. types, not tokens), of which 1 255 are valid and 82 invalid. For this evaluation task, the text has been constructed based on a small Afrikaans corpus of e-mail messages, and contains simple words, morphologically complex words, compounds, proper names, abbreviations, and foreign words that are valid in Afrikaans. Invalid words from the corpus have also been included.

The evaluation in this article is not meant to show which spelling checker is better, but merely to investigate what improvements are wanted.

2.2.3 Results

This section will describe some results of the two evaluated spelling checkers, *PUKspell* and *Bspell*. The results of applying the spell checkers to the test text described above, can be found in Table 2. This table contains percentages as computed by the metrics of Table 1 and, additionally, the plain counts are given between brackets. Note that further results can be found in Van Zaanen and Van Huyssteen (2003).

The actual results indicate two interesting differences between the two spelling checkers. The numerical results indicate that *PUKspell* performs overall worse than *Bspell* on this text. This is mainly because it rejects too many valid words, hence the lower lexical recall and precision. This can possibly be explained by the fact that the lexicon of *Bspell* is 31% larger than that of *PUKspell* (respectively approximately 180 000 and 235 000 words, at the time of evaluation).

Table 2: Comparison of evaluation results

Measure	*PUKspell*	Bspell
Lexical recall	95% (1 196/1 255)	98% (1 225/1 255)
Error recall	84% (69/82)	83% (68/82)
Precision	54% (69/128)	68% (68/98)
Suggestion adequacy	59% (41/69)	85% (58/68)

The other difference can be found when looking at the rejected words. Both spelling checkers have difficulty handling compounds, as Afrikaans is a semi-agglutinative language with productive compound formation.

The results indicate two means of improving the existing spelling checker:

- Increase the number of valid words the spelling checker will accept. This can be done by increasing the lexicon or adding morphological information.
- Extend the spelling checker with compound analysis. Related to this improvement is the handling of morphologically complex (non-compound) words.

2.3 Improving the existing spelling checker

This section describes several improvements that are implemented in the existing spelling checker. The improvements are selected based on the results described in the previous section. First, an overview of the existing spelling checker is given, followed by a similar overview of the improved spelling checker. Next each improvement is discussed separately. Finally, some remaining problems are mentioned briefly.

2.3.1 Overview of the existing spelling checker

The architecture of the currently available version of *PUKspell* can be found in Figure 1. First, *Microsoft Word*[5] delivers an arbitrary piece of text to be checked. This piece of text is then tokenised by breaking the text up in words on spaces. The lexicon lookup module then tries to find each word in its lexicon. This lexicon consists of roughly 185 000 words. If it is not found, the token is handed to the suggestion module, which searches for words in the lexicon that are relatively close to the incorrect word. The similar (correct) words are given to the user.

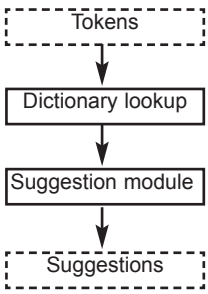

Figure 1: Architecture of the existing version of *PUKspell*

Since *PUKspell* is developed for *Microsoft Word*, the interface between the word processor and the spelling checker is fixed and outside our control. *Microsoft Word* fills an array with characters and then applies the spelling checker on this array. The text in the array does not have to be a single word, a sentence, or entire paragraph. It can contain one or more words, possibly including sentence boundaries, but this can be different each time the spelling checker is called. This is why the text needs to be tokenised first.

Since the spelling checker is mainly a lookup program that searches for words in a lexicon, the success of the spelling checker is directly related to the completeness of the lexicon. If not enough words are contained in the lexicon, too many valid words are flagged as invalid.

There are two opposing views on how large a spelling lexicon should be (Vosse, 1994: 49). One view claims that the lexicon size should be kept small, because increasing the lexicon size results in more misspelled words being accepted. This happens in particular when infrequent correct words that differ only in one letter with another valid word are added to the lexicon. The other view points out that the lexicon should be large, because this will reduce the number of valid words that are flagged as invalid. Based on the results, we concluded that the lexicon should be larger.

2.3.2 Overview of the improved spelling checker

This section will give an overview of the architecture of the improved spelling checker. Figure 2 shows the improved spelling checker in a graphical format.

In the improved spelling checker, the lookup module consists of several steps. First, the word is looked up in the standard lexicon. If it is not found, it does some error detection tests. This consists of two steps: lookup in the invalid word lexicon, and n-gram analysis. If the word is still found to be invalid, it is sent to the suggestion module (shown as a dashed arrow in the figure). The error detection lookup module

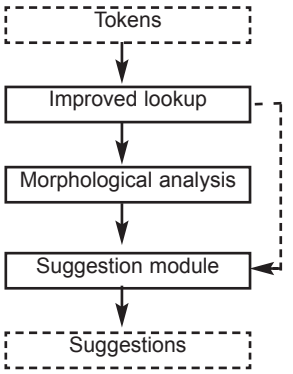

Figure 2: Architecture of the improved version of *PUKspell*

(invalid word lexicon and n-gram analysis) can only decide that a word is invalid. If the word is not recognised as an invalid word in this module, it is handed to the morphological analysis module.

In this module, inflected words, derivations, and compounds are analysed by stripping off parts of the original word. Inflected words and derivations are analysed by means of a Porter stemmer (Porter, 1980; Kraaij & Pohlmann, 1994) for Afrikaans (developed during this project), while compounds are analysed using a longest match string search algorithm. It then checks if the analysis leads to a valid word.

If the word is still not found, it is given to the suggestion module. Currently, the suggestion module is a simple extension of that of the existing spelling checker. However, as mentioned earlier in this article, improving the suggestion module is future work.

In the following sub-sections, we will sketchily discuss some of the improvements that are implemented in the new spelling checker. The first sub-section describes the improvements on the lexicon. We then give a concise depiction of the error detection module, after which the morphological analysis module is finally discussed.

2.3.2.1 Enlarged lexicon

Because too many valid words are not recognised (according to the evaluation), we decided to improve the spelling checker by extending the lexicon. This is done in two different ways, one direct and one indirect. The direct extension is treated in this section and the indirect in section 2.3.2.3.

The simplest way of making the spelling checker accept more valid words is to increase the number of words in the lexicon. This was done by adding words from the PUK/Protea corpus (especially compounds and other morphologically complex words), lists of technical terms (e.g. from an astronomy dictionary, chemistry dictionary, psychology dictionary, etc.), as well as terms from a corpus of legal texts. Proper names, business names and geographical names were also added to the lexicon. The next release of the spelling checker will have a larger lexicon, containing more than 250 000 words (37% larger than the existing version). In this respect, the improved *PUKspell* is then roughly similar to *Bspell*.

Although the process of increasing the size of the dictionary is expensive and time consuming, it is still considered a worthwhile effort; not only does it provide for higher lexical recall in the spelling checker, but such a lexicon can also be considered highly valuable for the development of other applications. While the extension (and quality control) of the lexicon is currently done semi-automatically (i.e. by automatically extracting new words from corpora and checking these words with other spelling checkers), some manual labour is still involved (e.g. going through the existing lexicon to find misspelled words added from corpora (e.g. **plavleisel*), as well as archaic words that can possibly lead to typing errors (e.g. *ens* 'be fond of', which can easily be mistaken for the abbreviation *ens.* 'etc.' when the full-stop is omitted)). In future, this process will continue and will be further automated wherever possible (e.g. by using a Named Entity Recogniser, currently under development in another project at the PUCHE).

2.3.2.2 Error detection

The handling of errors in the improved spelling checker is extended by using two new modules. These modules allow for a more efficient handling of invalid words in different ways. The new modules filter out obviously invalid words (like typing errors), as well as frequently occurring spelling mistakes (i.e. cognitive errors). The main advantage of this is that fewer invalid words are sent to the morphological analysis module, reducing the processing time.

The first module is actually another lookup module. Instead of looking up valid words, the word is matched against a list of frequently occurring invalid words, which was compiled over a period of almost two years from e-mail messages, the PUCHE's electronic bulletin board, students' essays and exam papers, as well as from colleagues making contributions to our error corpus. If the word is found in this list, the corresponding correction is directly made with *Microsoft Word*'s *Auto Correct* function. For example, **inteligent* is in the invalid word lexicon, with as obvious correction *intelligent*.

The second error correction module analyses the letters in the word itself. From a corpus (consisting of roughly 200 000 unique words), n-gram counts (of letters/graphemes) are extracted with *n* ranging from three to five. Invalid words often have n-grams that are not valid in Afrikaans. This module finds invalid 'words' such as *xqyz*, but also more subtle errors as in *maatxkappy* ('society'), where *x* is substituted for an *s*. Additionally, the n-gram analysis often gives a rough indication of where the error in the word can be found, and this information can then again be used in the suggestion module (especially for suggesting errors in compounds).

2.3.2.3 Morphological analysis

Even though increasing the lexicon allows the spelling checker to handle more words (cf. 2.3.2.1), Afrikaans, as a semi-agglutinative language, has a relatively strong generative lexical power, using a set of inflectional, derivational, and compounding 'rules' to form new words. This means that a spelling checker should be able to handle many morphologically complex words. One solution would be to add as many morphologically complex words as possible to the lexicon, so that the spelling checker will accept these words. However, this approach will not only greatly increase the size of the lexicon, but will also prove to be ineffective, as the lexicon will have to be updated on a regular basis to handle new words. Another solution will be to recognise morphologically complex words by adding morphological information to the lexicon separately, and by adding morphological analysis modules to the spelling checker.

As can be seen in Figure 3, removing all morphologically complex words from the lexicon (middle ellipse) results in a reduced lexicon (smallest ellipse). Adding morphological rules keeps the actual lexicon small, but the amount of words recognised by the lexicon and the morphological rules are much larger (largest ellipse).

The improved spelling checker implements morphological information in the form of morphological rules. Morphologically complex words that are not found in the lexicon during the lookup procedure are analysed using these morphological rules. This approach would actually allow for a smaller lexicon, as morphologically complex words (i.e. inflected words, derivations, and compounds) are analysed, rather than being listed in the lexicon. These rules are employed in two different modules, viz. a stemmer (to detect valid inflected words and derivations), and a simple compound analyser (to detect valid compounds that are not in the lexicon).

As mentioned earlier, a stemmer, based on the Porter stemmer for Dutch (Kraaij & Pohlmann, 1994), has been developed during the course of this project. More than 200 rules are employed in the stemmer to handle a set of frequently (as well as less frequently) occurring inflectional and derivational affixes (for a more detailed description of the stemmer, see Van Huyssteen & Van Zaanen, 2003). The remaining string is looked up in the lexicon.

Because stemmers are usually used in the context of information extraction and retrieval, words that are in the same semantic class should reduce to the same stem. However, in a spelling checking context, this is not the case, as valid words should simply remain valid (and

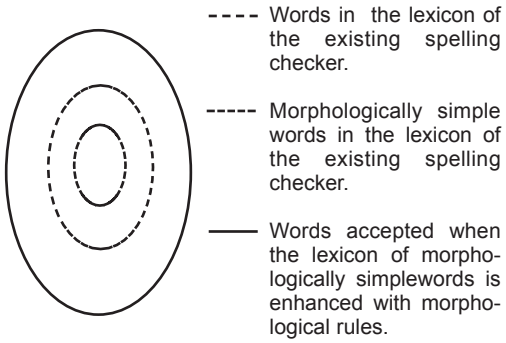

Figure 3: Morphological rules increase the number of accepted valid words

invalid words must also remain invalid). For example, it would be problematic in an information retrieval (IR) environment if the semantically unrelated forms *geboor* ('drilled'), *geboorte* ('birth') and *boorde* ('orchards') would all be reduced to the stem *boor* ('drill'); however, in a spelling checking environment it would be unproblematic, as long as the stem *boor* is a valid string. On the other hand, if the words *elektrisiteit* ('electricity'), *elektrifiseer* ('electrify') and *elektrisiën* ('electrician') are reduced to *elektr-* in an IR system it would probably be considered unproblematic, whereas it could be problematic in a spellchecker if the bound stem *elektr-* does not occur in the lexicon.

The main difficulty with using a stemmer is that it can easily over-generate. For example, Combrink (1990) identifies the *a-* as a possible prefix in Afrikaans, for example in words as *asosiaal* ('asocial') and *atonaal* ('atonal'). However, this prefix cannot be used in the spelling checker because the prefix cannot be removed in words like *Afrikaans* and *artistiek* ('artistic'), where the *a-* is clearly not a prefix. We therefore had to restrict the greediness of the stemmer by reducing the number of rules. As this resulted in higher precision (i.e. an invalid word like **venstere* is not analysed), but also in lower lexical recall (i.e. a valid word like *glasvensters* 'glass windows' is not recognised), we had to experiment with different combinations of rules in order to strike an effective balance.

A further problem is the handling of compounds. Compare the following characteristics of compounding in Afrikaans:

- A compound can consist of several words, e.g. *strand+hand+doek+stof* ('beach towel cloth'), so trying to combine only two words from a lexicon is not enough.
- Words can have metafixes, such as *-s-* in *bruilof-s-gas* ('wedding guest'), *-e-* in *student-e-nommer* ('student number'), or *-en-* in *wa-en-huis* ('garage'). But also *-ns-, -ens-, -tes-*, and *-er-* are possible metafixes in a compound. There are no straightforward (implementable) rules for handling this.
- Sometimes, doubling of consonants occurs, for example in *den-n-e-boom* ('pine tree'), where the *n* is doubled.
- A compound can also be constructed from morphologically complex words, for example *vergadering-s-prosedure* ('meeting procedure') or *verantwoordelikheid+sin* ('sense of responsibility').

Currently, a simple longest string matching approach is taken. Given a compound, the goal is to try to find in the compound the longest match from the left that is still a valid word in the lexicon. Similarly, this can be done from the right. Remaining parts of the compound can again be stripped in this way.

The problem with the longest string matching approach is that it is hard to handle morphologically complex compounds (meaning words that should be treated using the stemmer as well — e.g. *straatnaamgewing* 'street name giving'/'street naming'), or handling compounds that are formed by more than three constituents (e.g. *driepuntsterembleem* 'three point star emblem'). Another remaining problem with this is the correct treatment of metafixes (i.e. distinguishing between the forms *studente**n**ommer* and **studentnommer* 'student number').

Another approach that might be taken is to use machine learning techniques to find possible word boundaries (within compounds). A classifier can be trained using a lexicon of structured compounds (with delimiters between the elements within compounds). This classifier can then find word boundaries in unseen words.

The main problem with the machine learning approach is that in the context of the spelling checker, the compound splitter is not applied to valid compounds only, but also to compounds containing spelling errors. Especially when a spelling error occurs close to a word boundary, this might influence the accuracy of the compound splitter.[6]

Although some work on compound splitting has already been done (Koehn & Knight, to be published), most approaches make extensive use of structured training data. Unfortunately, this data is not (yet) available for Afrikaans, making these approaches less applicable.

2.4 Summary

This section described a comparison between two existing commercial spelling checkers to decide which improvements are desired in a new spelling checker. The results of the comparison showed that the coverage of the spelling checker (measured in precision and recall) should be improved. Additionally, the suggestion module should be improved.

The coverage of the spelling checker is improved in several ways. The lexicon is enlarged, and morphological information is added, which allows for the treatment of morphologically complex words. Finally, the problem of handling compounds is discussed although currently, it remains as future work. Additionally, n-gram analysis and an error lexicon are introduced that filter out some invalid words and help the suggestion module find the spelling errors in a more effective way.

When looking at the practical experiences from a wider perspective, we found that even building a spelling checker, which is often considered a simple problem, is extremely difficult if only limited resources are available. For example, to handle morphologically complex words, a sophisticated morphological analyser is needed. For this project, no such tools were available, so a stemmer (which does some simple morphological analysis) had to be developed. Similarly, to be able to obtain n-gram counts or to employ machine learning techniques (which is currently considered future work), large, clean and preferably annotated corpora are needed. Currently, these types of resources are not available for Afrikaans, which greatly hinders the building process of NLP applications, such as spelling checkers.

Fortunately, the lack of resources was recognised early in the project. As a result, certain design choices were made that lead to the development of resources, such as the stemmer, that can be re-used in future projects (e.g. in an information retrieval project). Additionally, this project gives an indication of the kinds of resources that will be needed in future to develop language technology applications for Afrikaans (and other South African languages).

More importantly, extensive hands-on experience in the field of computational linguistics, natural language processing, and software design was obtained by the inexperienced members of the development team. For instance, the evaluation of the spelling checkers introduced team members to concepts such as ‘recall’, ‘precision’ and ‘processing time’, as well as with other issues regarding the systematic evaluation of software — experience that will be valuable also for future projects. Also, with regard to programming, valuable experience was obtained by the linguists in the team. For instance, in order to develop the stemmer, the linguists had to implement the linguistic rules in Perl scripts. Not only did this force them to learn a scripting language, but also to gain experience with programming concepts such as sub-routines, iteration, data structures, and encoding. The non-linguists, on the other hand, gained enormous experience in understanding the flexibility and intricacies of language structure and learning to deal with it in a rigorous computer programming environment.

3. Teaching and learning

It was indicated in the previous section that a wealth of experience and knowledge was gained through this project. As little (or no) literature exists on Afrikaans computational linguistics (with specific reference to text technology), students and lecturers/researchers had to learn together, even by making mistakes and taking ‘wrong’ sidetracks. This hands-on approach has proved to yield excellent results in the acquiring of knowledge in the field of computational linguistics.

This section describes how the graduate and post-graduate teaching programmes at the PUCHE benefitted from this spelling checker project. First, it is indicated how work on the project was integrated in the graduate programme, with possibilities to incorporate future work in the programme. We then give an overview of the involvement of post-graduate students in the project, and how they gained knowledge by working on this project. In the last instance, it is illustrated how members of staff and students alike acquired knowledge (and experience) outside the boundaries of the curriculum, by getting trained in specialised skills.

3.1 Graduate programme

Since 2002 a new graduate programme was introduced at the PUCHE, called *BA Language Technology*. The introduction of this programme was motivated by two factors, viz.:

(a) the need to develop teaching programmes that are relevant, vocationally directed, and future-oriented; and
(b) a gap in the South African educational market, with no graduate programmes on offer in computational linguistics.

After wide consultation with international and local role-players and experts, a programme was designed that combines language

subjects, subjects from computer science, mathematics and statistics, as well as a core group of computational linguistic subjects (e.g. “Introduction to Computational Linguistics”, “Methodology and Formalisms”, “Corpus Linguistics”, “Statistical Natural Language Processing”, etc.). From the onset, the curricula of the latter group of subjects were envisaged to be problem-oriented and project-organised, based on the educational system developed at the Aalborg University, Denmark, since 1974 (Kjersdam & Enemark, 1994).

Problem-oriented education (or Problem Based Learning (PBL); Albanese & Mitchell, 1993; Schwartz *et al*., 2001; see also Macdonald, 2002) can be defined as learning “based on working with unsolved, relevant and current problems from society/real life… By analysing the problems in depth the students learn and use the disciplines and theories which are considered to be necessary to solve the problems posed, i.e. the problem defines the subjects and not the reverse” (Kjersdam & Enemark, 1994: 16). By means of this applied research and teaching approach a dynamic triangular equilibrium between research, education and practice is maintained, serving researchers (and research outputs), students, and the industry alike. Also, by enabling students to gain “comprehensive knowledge of the development of theoretical and methodological tools” (Kjersdam and Enemark, 1994: 17), they will, after completion of their formal studies, be able to contribute to research and the development of original paradigms to solve new and complex problems in the future.

PBL is incorporated with project-organised education in two ways. On the one hand various project-based modules are included in the curriculum. For instance in the third year of study, the modules “Speech Technology: Applications” and “Text Technology: Applications” are introduced, where students will develop various small modules of both speech and text technological applications (e.g. a simple rule-based sentenciser or tokeniser). In the final year of study, the largest part of the year is spent on independent project work, which is conducted either at the university, or while doing an internship elsewhere. The idea is that these projects will be on a much larger scale than the third-year projects, possibly building on the work of the previous year (e.g. to develop a more sophisticated sentenciser or tokeniser, using more advanced NLP techniques).

On the other hand, course work in some of the other modules (e.g. in “Corpus Linguistics” or “Speech Technology: Introduction”) is also organised around existing research projects. Students are mostly involved on a so-called “design-oriented” level, i.e. where they have to deal with “know-how problems which can be solved by theories and knowledge they have acquired in their lectures” (Kjersdam & Enemark, 1994: 7). After the project and the problems related to the project are explained to students, they get involved by collecting data, identifying possible/different solutions, formulating rules and algorithms, analysing data, evaluating different components, etc. In this way they therefore get know-how and experience on theoretical, methodological, and implementation level.

For instance, in the modules “Grammar: Introduction” and “Grammar: Intermediary”, students were introduced to this spelling checker project. It was explained to students why the project was started, and what the problems were with the existing spelling checker (i.e. that it was not able to deal with the morphological productivity of Afrikaans). After introducing some basic and general concepts of morphology (e.g. inflection, derivation, compounding, etc.), students got involved in the project by collecting data (e.g. compounds) and analysing this data, using the theoretical framework that was introduced during the lectures (i.e. Cognitive Grammar and Construction Grammar). The outcome of this course work was several descriptions of different morphological constructions in Afrikaans (e.g. the plural construction, the past tense construction, etc.), which was implemented in the stemmer that was developed. Also, new data could be added to the lexicon of the spelling checker.

The advantages of this approach proved to be numerous: not only did the project benefit from the data collection and descriptions, but students got the feeling that they were involved in “important” and relevant work. They got the opportunity to apply the theoretical knowledge they acquired in the classes in a practical, hands-on environment, and in that way improving their understanding of the theories and concepts of the study field. Also, of course, our general understanding of Afrikaans morpholog-

ical structure (especially within a computational context) was enriched by means of this approach.

3.2 Post-graduate programme

At post-graduate level, the Honours module "Language, text, and technology" in the Afrikaans honours programme, was, among other modules, introduced to provide students with a general BA or BSc background, an entry point into post-graduate studies in computational linguistics. From its very inception, this module was conceptualised to be problem-oriented and project-organised, offering lecturers an opportunity to align course material with their research projects.

In 2002, three students enrolled for the module, and the curriculum was organised around the spelling checker project. Together with these three students we first discussed the project, its aims, and its possible outcomes in immense detail (i.e. students were involved on the design-oriented level), thereby applying their theoretical knowledge of aspects such as text editing and Afrikaans grammar in a practical and useful way. After that, each of the students got involved in solving different problems of the projects, e.g. evaluating the spelling checkers, developing a stemmer and a word segmenter, extending and cleaning-up the lexicon, etc. Although each of them worked separately on different problems, they were forced to extend their experience by helping each other with their different tasks, thereby expanding their general knowledge and experience.

Once again, the advantages of this approach proved to be multitudinous. Students and lecturers/researchers with no background in computational linguistics got the opportunity to obtain knowledge and experience in the field, while learning to solve real-world problems. Additionally, members of staff were enabled to harmonise their research and teaching responsibilities, optimising the quality and quantity of their outputs. By involving students as co-workers we were able to get work on the spelling checker, as well as on the development of computational linguistic resources for Afrikaans, done in a shorter time. Moreover, all three students were motivated to continue with their studies in computational linguistics on MA level, where they are currently involved in the NRF-funded project on Afrikaans computational morphology.

3.3 Extracurricular teaching and learning

Learning in this project was, of course, not restricted to the formal educational programmes, as extensive learning took place outside the curriculum. For example, the linguists in the team (students and researchers) had no (or very little) experience in computer programming. However, during the course of this project, they were all forced to learn to formalise their ideas and knowledge, in order to communicate effectively with the other members of the team (i.e. the computational linguist and the programmers). They therefore had to learn more about formulating algorithms, drawing up flow charts, and, in general, thinking like computer programmers.

More importantly, during the project, the linguists were forced to learn how to do actual programming in order to develop prototypes of the stemmer and word segmenter. This was done by learning to use regular expressions, implementing it in the Perl programming language. Perl is a programming language that was initially developed specifically to do text processing, and is therefore ideally suited for the computational linguist working with text. Furthermore, it is a relatively easy programming language to learn, and is therefore considered to be a good first language for inexperienced programmers to learn. During the course of this project, the linguists in the team therefore acquired programming knowledge and experience that will be invaluable for future projects.

Related to the above, was the learning involved in working in a project team. The team consisted of several linguists, a computational linguist, and a computer scientist (programmer), with administrative, legal and financial support by people not working in either of the above mentioned fields. It was therefore of utmost importance to ensure that all parties involved understood exactly what was expected from him/her, by communicating ideas in an effective way. This project was thus an attractive way for linguists to get a good feeling of particular problems that arise when implementing linguistics in a computational environment, while learning to express their ideas in a structured, logical way. The computer scientist also

had to learn to deal with data and structures that are atypical of the data that he was used to working with. In order to facilitate these communication problems, the computational linguist played a vital role in training the other groups to understand the idiosyncrasies of the respective fields.

3.4 Summary

It was shown in this section that the spelling checker project offered us the opportunity to integrate research and educational activities, by taking a problem-oriented and project-organised approach to education. Students and researchers/lecturers worked together on the development of computational tools and resources for Afrikaans, while simultaneously learning about computational linguistics, natural language processing, and other related activities (i.e. programming, working in project teams, etc.). Not only did this approach prove to be an effective way of learning, but also to be efficient in financial terms and in terms of time.

4. Conclusion

Even though many computational linguists are not particularly interested in researching spelling checkers (because it is considered to be an easy task), implementing a spelling checker that performs well proves to be a difficult problem. In this article, we gave an overview of a spelling checking project for Afrikaans. This project has several aspects that are all discussed, focusing on how practical experiences were obtained during the evolution of the project. In the first part we described the evaluation of the existing spelling checker and, based on the results, discussed the improvements that were introduced and implemented. We then discussed some remaining problems and proposed some possible solutions. We also indicated that the lack of NLP resources and corpora for Afrikaans has an important influence on the development tempo and efficiency of applications; an urgent need therefore exists for the systematic acquisition of corpora, the standardisation of protocols, annotation schemes, etc., as well as the rapid development of annotated corpora and enabling technologies.

In the second part, the underlying teaching and learning benefits of the project were discussed. This includes improving our knowledge of computational linguistics on Afrikaans, improving programming skills for linguists, and communication between experts in different fields. We also gave an overview of how the project is used for problem-oriented and project-organised educational purposes, and how it solves certain problems related to the above-mentioned shortage of computational linguistic tools and resources for Afrikaans.

After the first phase of this project (i.e. to develop an improved, second generation spelling checker for release as a commercial product), much is still left to be done. It is for example absolutely necessary to give a systematic usage-based description of spelling (and other) errors that occur in Afrikaans texts. This will motivate further improvements on the current spelling checker, making it possible to design solutions that will benefit end-users directly. Although many 'typological' studies of spelling errors have already been done (i.e. categorising the different kinds of spelling errors; cf. Kukich, 1992 for an overview of several studies on error classification), it will be necessary to describe the kinds of spelling errors occurring in different genres of texts. For example, one can suspect that more typos will occur in e-mail correspondence than in other kinds of texts, because e-mail messages are often composed hastily and on-line. One might, on the other hand, expect more cognitive errors (especially with regard to compounds) in academic writing (e.g. scholarly articles, theses, etc.), since more complex concepts are expressed with increasingly complex compounds and other constructions.

Other future work will focus on context-sensitive spelling checking, which is the detection and correction of so-called 'real-word' errors (Verberne, 2002: 32). Although some work has already been done in this regard (cf. Mays *et al*., 1991; St-Onge, 1995; Golding & Schabes, 1996), much is still left to be done; with regard to Afrikaans, no research has been done in this field as far as our knowledge goes. This current project will therefore be extended in future to find effective ways to solve real-word error problems (i.e. to detect and correct the incorrect usage of homographs, homonyms, homophones, etc.), eventually leading to and opening up ways for the development of a fully fledged grammar checker for Afrikaans.

We therefore conclude that a spelling

checker project for a particular language can hold many benefits for a research team, despite the fact that it is often considered a less interesting computational linguistic project. Especially for researchers working with aggltinative languages (like Afrikaans or Setswana) a spelling checker project offers the opportunity to empower team members (linguists and non-linguists alike), while developing a product that has the potential to earn revenue to fund other, 'more interesting' research.

Notes

[1] Various universities in South Africa have single courses/modules on specific computational linguistics topics, but no complete programmes are given, with the exception of the programme discussed in this article.

[2] We thank and acknowledge the following members of the research team for their inputs in this research: Roald Eiselen, Christo Els, Petri Jooste, Christo Muller, Sulene Pilon, Martin Puttkammer, Werner Ravyse, and Daan Wissing. We would also like to express our gratitude towards Attie de Lange, Ulrike Janke, Elsa van Tonder, Annette Combrink, and Boeta Pretorius for technical, administrative, and legal support. The PUCHE also sponsors this project generously — our thanks to Frikkie van Niekerk for his support.

[3] Due to South African law, we are not allowed to use the name of the commercial spelling checker directly.

[4] In this article, we will not make a real distinction between the terms spelling checker and spelling corrector, and will use the term spelling checker to also imply the functionalities of a spelling corrector.

[5] *Microsoft* is a registered trademark and *Microsoft Word* is a trademark of *Microsoft Corporation*.

[6] At the time of writing this article, work along these lines has already been started. Although no conclusive results are available, all indications are that this approach will yield better results than the simple longest string matching approach.

References

Albanese MA & Mitchell S. 1993. Problem-based learning: A review of literature on its outcomes and implementation issues. *Academic Medicine* **68**(1): 52–81.

Combrink JGH. 1990. *Afrikaanse Morfologie*. [Afrikaans Morphology]. Pretoria: Academica.

Golding AR & Schabes Y. 1996. Combining trigram-based and feature-based methods for context-sensitive spelling correction. In: Joshi A & Palmer M (eds) *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*. San Francisco: Morgan Kaufmann. pp. 71–78.

Kjersdam F & Enemark S. 1994. *The Aalborg Experiment: Project Innovation in University Education*. Aalborg: Aalborg University Press.

Koehn P & Knight K. to be published. Empirical Methods for Compound Splitting. *Proceedings of the 11th Annual Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*. Dublin, Ireland.

Kraaij W & Pohlmann R. 1994. Porter's stemming algorithm for Dutch. In: Noordman LGM & De Vroomen WAM (eds) *Informatiewetenschap 1994: Wetenschapelijke bijdragen aan de derde STINFON Conferentie*. pp. 167–180.

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

Macdonald R. 2002. Problem-based learning: some references. Available at: [WWW:]www.ics.ltsn.ac.uk/pub/pbl [Accessed 5 May 2003].

Mays E, Damerau FJ & Mercer RL. 1991. Context based spelling correction. *Information Processing and Management*. **27**(5): 517–522.

Paggio P & Music B. 1998. Evaluation in the SCARRIE project. *Proceedings of the First International Conference on Language Resources and Evaluation*. pp. 277—282.

Porter MF. 1980. An algorithm for suffix stripping. *Program* **14**(3): 130–137.

Prinsloo DJ & De Schryver G-M. 2003. Towards second-generation spellcheckers for the South African languages. In: De Schryver G (ed) *6th International Terminology in Advanced Management Applications Conference: Conference Proceedings*. Pretora: (SF)^2 Press. pp. 135–141.

Schwartz P, Mennin S & Webb G. 2001. *Problem-based Learning: Case Studies, Experience and Practice*. London: Kogan Page.

St-Onge D. 1995. *Detecting and Correcting Malapropisms with Lexical Chains*. MA thesis. Published as Technical Report CSRI-319. Toronto: University of Toronto.

Van Huyssteen GB. 2002. Desiderata of spellchecking/spell checking/spell checking: towards an intelligent spellchecker for Afrikaans. Paper presented at a one-day symposium, *Developing Spelling Checkers for South African Languages*. 14 March. Potchefstroom University for Christian Higher Education, Potchefstroom, South Africa.

Van Huyssteen GB & Van Zaanen MM. 2003. A spellchecker for Afrikaans, based on morphological analysis. In: De Schryver G (ed) *6th International Terminology in Advanced Management Applications Conference: Conference Proceedings*. Pretora: $(SF)^2$ Press. pp. 189–194.

Van Zaanen, MM & Van Huyssteen GB. 2003. Improving a spelling checker for Afrikaans. In: Gaustad T (ed) *Computational Linguistics in the Netherlands 2002: Selected Papers from the Thirteenth CLIN Meeting*. Amsterdam: Rodopi. pp. 143-156.

Verberne S. 2002. *Context-sensitive Spell checking Based on Word Trigram Probabilities*. MA thesis, University of Nijmegen, Nijmegen, The Netherlands.

Vosse TG. 1994. *The Word Connection: Grammar-Based Spelling Error Correction in Dutch*. PhD thesis, Rijksuniversiteit Leiden, Leiden.