An OpenOffice Spelling and Grammar Checker Add-in Using an Open Source External Engine as Resource Manager and Parser

Editha D. Dimalen, Davis Muhajereen D. Dimalen
eddimalen@yahoo.com, d_dimalen@yahoo.com
Information Technology Department
School of Computer Studies
MSU- Iligan Institute of Technology
Iligan City, Philippines

ABSTRACT

In this paper we explored a different approach in developing a spelling and grammar checker add-in for OpenOffice Writer. The typical approach uses a file as lexicon with parser and stemmer embedded directly to the OpenOffice Writer. On the other hand, we used postgresSQL, an open source database management system (DBMS), as storage for our lexicon. The parser and stemmer was developed using postgreSQLs stored procedures. With our architecture, the spelling and grammar checker engine is independent of OpenOffice Writer and can be used in other NLP applications. PostgreSQL is capable of storing and processing large wordlist and can be installed in both Unix and Windows platform. Our lexicon contains 14,000.00 Tagalog root words and can still be updated up to the number of records of data a hard disk can handle. The inclusion of a Tagalog stemmer (TagSa) increases the number of words that can be corrected by our spell checker. TagSa is used to minimize the inclusion of inflicted words in the lexicon. Before a lexicon look-up is done the TagSa module is executed first to check if the word is a root word or not. If it is not a root word, the word will be stemmed into its root word and finally be checked if it is found in the lexicon. Words that are found in the lexicon are flagged as correct. We were also able to develop an on-the-fly grammar checker, a feature which is still an on-going research by OpenOffice.org. Both the spelling and grammar checker add-in are for Tagalog, a widely spoken language in the Philippines. The system has been evaluated using a separate program to calculate the execution time in spell checking and grammar checking. The evaluation showed faster return of results that effectively checked the text and gave adequate suggestions.

1. INTRODUCTION

The effort in obtaining an error-free spell checking of words and automatically suggests possible match is a great research challenge [3]. Several issues are being addressed to give an appropriate resolution to a spell checker in varied natural languages. According to O'Neill, et. al., 2003 [11], "spelling checkers have looked for four possible errors: a wrong letter ("wird"), an inserted letter ("woprd"), an omitted letter ("wrd"), or a pair of adjacent transposed letters ("wrod")". This process can be resolve by means of a simple dictionary lookup. However, the notion of having languages with high degree of inflection (like Tagalog) requires additional computational work such as morphological analysis and stemming.

Current developments of open source spell checkers are the following: Bahasa Melayu (BM) Spell Checker for Malay language [8], Fijian Spell Checker developed for Fijian language [16], Divvun a spell checker research for Sami language [9]. Other variations of existing open source spell checkers family are the ISpell, Myspell and HunSpell. ISpell is a unix-based system, MySpell and Hunspell support spell checking in OpenOffice.org [6]. The dictionary and the affix files are stored in a file. The parser and stemmer engine are embedded in OpenOffice.org.

Developing a spell checker and grammar checker as add-in for OpenOffice.org from scratch is complex especially for a language with words rich in affixations like Tagalog. A word processor add-in is a supplemental program that extends the capabilities and functionalities of a word processing application [7]. Aside from the formulation of an effective algorithm that can process a bunch of text and lexicon to produce good results, storage optimization and management should also be considered to compliment the algorithm. Development will be focused on how these elements would compliment each other and can be time consuming. Issues in building hash tables, memory optimization is still an open problem in developing spell checker and grammar checker for OpenOffice [5]. Thus, we employed postgreSQL wherein hashing, storage management and memory optimization is not an issue [13]. There are no current researches that use postgreSQL as a resource and spell checker and grammar checker engine for OpenOffice.

2. CONCEPTUAL FRAMEWORK

The following sections describe the underlying theories used in the development of the spell checker and grammar checker for OpenOffice Writer.

2.1. Tagalog Stemming

Affixation in Tagalog language is complex especially on verbs and nouns. TagSA [1], a dictionary-based stemming algorithm for Tagalog considered a procedure in reducing all words (inflected) with the same root to a common form. This is basically done by stripping each word with appropriate affixes (derivational and inflectional affixes).

The Tagalog morphological combination includes prefixation, infixation, suffixation, circumfixation and reduplication. Prefixation involves the process of attaching a bound morpheme before the root word. An example is ma*G* + *Aral* → mag-aral, in which a consonant *G is* attached to vowel *A* with a hyphen. Infixation is attaching a bound morpheme within the root word. Example the word "kinuha" has the infix /*-in-/* wherein the root word is "kuha". In suffixation, bound morpheme is attached at the end of the root word. For example, harap + in → harapin. In circumfixation, bound morpheme may occur as in any order (prefix, infix and suffix). The example pa + in + punta + han → pinapuntahan, morphemes appear anywhere within the word. Tagalog reduplication can either be partial or full. Partial includes certain syllables that are duplicated to project the form of the stem. Full reduplication includes the entire stem to be repeated.

TagSA consists of several routines in handling different affixation. The main routines are the following [1]:
1.0 Hyphen-Search Routine
2.0 Dictionary-Search Routine
3.0 /-in-/ Removal Routine
4.0 Prefix Removal Routine
5.0 /-um-/ Removal Routine
6.0 Partial Reduplication Routine
7.0 Suffix Removal Routine
8.0 Full Reduplication/Compounding Routine

2.2. N-gram Theory

The suggestion strategy employed in the system is based on n-gram. N-gram is a result of removing spaces from a given string. In a given string, *n* items can be generated from a given sequence. The sub-sequence of these items can be compared to other sequences [17].

An n-gram can also be seen as an n-character slice of a longer string in which a string is sliced into sets of overlapping n-grams. However, blanks are appended at the beginning and end of the string before the string is sliced [2]. Example,

String = "text" bi-grams (N=2) = _t, te, ex, xt, t_
Token = "_text_" tri-grams (N=3) = _te, tex, ext, xt_, t_
quad-grams (N=4) = _tex, text, ext_,

(knowledge about individual words) is essential. Lexical knowledge is encoded through a lexicon in strictly formal structure. A lexicon has long been recognized as a critical system resource [4]. A basic lexicon typically includes explicit and specific linguistic information about the word. It includes the morphology either by enabling the generation of all potential word-forms or by simply listing all associated pertinent morphosyntactic features. Lexicons are traditionally been built by hand specifically for the purpose of language analysis and generation. It is typically encoded in a text file for lookup.

2.3. Grammar Checking Issues

Grammar components include grammar rules, lexical entries, principles and parts-of-speech specifications of each lexical entry. The input text is passed through a series of filter: preprocessing, segmentation, tokenization, lookup, chunking, disambiguation, rules and recourse.
Preprocessing stage converts the text into the native character if the default text is in different encoding. The segmentation step involves breaking text into sentences and split the sentence into words. The next step is to looked up each word in the lexicon in which each word is tagged with its part-of-speech (POS). Words that are not found in the lexicon will be processed by the morphology engine to be able to recognize the known root word. In this stage, phrases will be grouped together to form a single units by the grammar checker. The text that has been analyzed will be matched against the built-in rules [14].

It turns out that there are basically three ways to implement a grammar checker: syntax-based checking, statistics-based checking and rule-based checking. Rule-based checking is the most common method used. It comprises a set of rules that is matched against a text which has been at least tagged with POS. In this approach, all the rules are developed manually [10].

3. METHODOLOGY

3.1. Data Gathering, Review and Analysis

A thorough research, review and analysis of existing works on spell checking and grammar checking was conducted. Computational issues on lexicon development, stemming (which includes morphological and syntactic information) and complexities in add-in development were considered. Existing add-ins are examined to be able to determine the most effective implementation process.

3.2. Design of the System

An architectural design of the system which includes the internal process of spell checking and grammar checking process was developed and is discussed in detail in section 4.2. Another architectural design which describes the components of the add-in is also discussed in section 4.2.

3.3 Implementation of the System

The implementation of the system is based on the final architectural designs discussed in the later sections. StarOffice Basic programming language and UNO (Universal Network Object) [12] was used to develop the add-in. SDBC (StarOffice Database Connectivity) and ODBC (Open Database Connectivity) was used to bridge the postgreSQL engine with the OpenOffice document.

4. RESULTS AND DISCUSSIONS

Research results and its corresponding discussions are discussed in the succeeding sections. The discussion follows from the design up to the evaluation of the system.

4.1. Architectural Design

This section describes the architectural designs of the Spell Checker, Grammar Checker, and the overall architecture of the entire add-in system.

4.1.1. Spell Checker

Figure 1illustrates the spell checker architecture of the add-in. It describes the processes in spell checking a document how suggestions are listed.

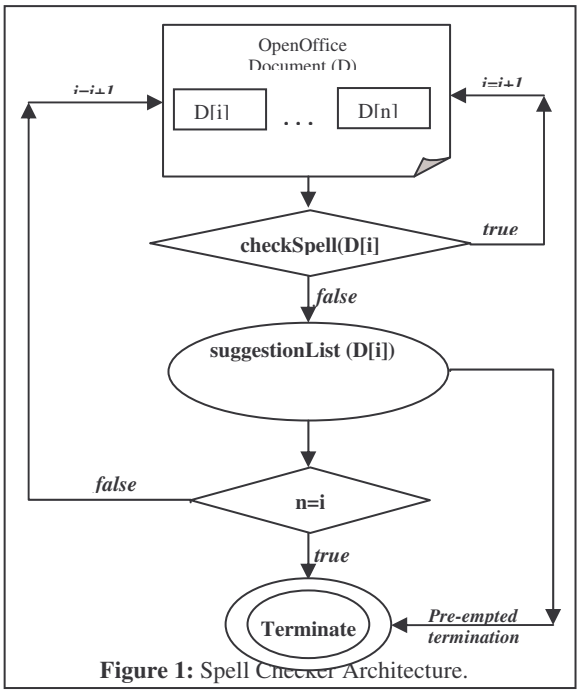


Figure 1: Spell Checker Architecture.

Consider a document *D* with number of words equal to *n* where *n-1* is equal to the index of the last word in document *D*. Let *i=0* be the index of the first word found in document *D* and *D[i]* be the word pointed to by the index *i.* Let *i+1* be the index of the next word. Let **checkSpell**(*w*) be the function that will accept a parameter *w* wherein *w* can be the word *D[i]*. The function will return true if the word *w* is spelled correctly. Correctly spelled word means that the root word of the input word *w* is found in the lexicon after stemming is done. Let **suggestionList**(*m*) be the function that will return a list of suggested word as replacement for the miss spelled word *m*.

The following steps describe the algorithm shown in the architectural design of the spell checker in Figure 1.

Step 1: At *i=0*, get the word *D[i]*

Step 2: if **checkSpell**(*D[i]*) returns true, consider the next word i wherein *i=i+1* and repeat Step 2 if *i* is equal to *n* goto Step 5. If **checkSpell**(*D[i]*) returns false then continue to Step 3.

Step 3: Display a list of word suggestions returned by the function **suggestionList**(*D[i]*). Select a word from the list returned by **suggestionList**(*D[i]*) (the process can be pre-empted or manually terminated by jumping to Step 5 or continue to Step 4).

Step 4: if $i < n$ then consider next word *D[i]* wherein *i=i+1* and repeat Step 2 else goto step 5.

Step 5: terminate algorithm.

The ***checkSpell(D[i])*** is lexicon based. It uses TagSa as an initial subroutine that will check if a root word can be extracted from an input word before a final lexicon look-up is done. Words are tagged as miss-spelled if after it has been stemmed to its root word, it is still not found in the lexicon.

The ***suggestionList(D[i])*** uses an n-gram approach and at the same time uses a lexicon based approach to look-up generated n-grams of input words to the lexicon. What is compared to the input word n-gram is not the entire word from the lexicon but the substring of the words found in the lexicon that matches the n-gram of the word. In this case there is no need to maintain an n-gram profile since the algorithm is more of a direct string pattern matcher. No statistical analysis involved in the algorithm unlike an n-gram based algorithm that makes use of a n-gram profile table.

4.1.3. Grammar Checker

In Figure 2, the architectural design of the grammar checker is shown.

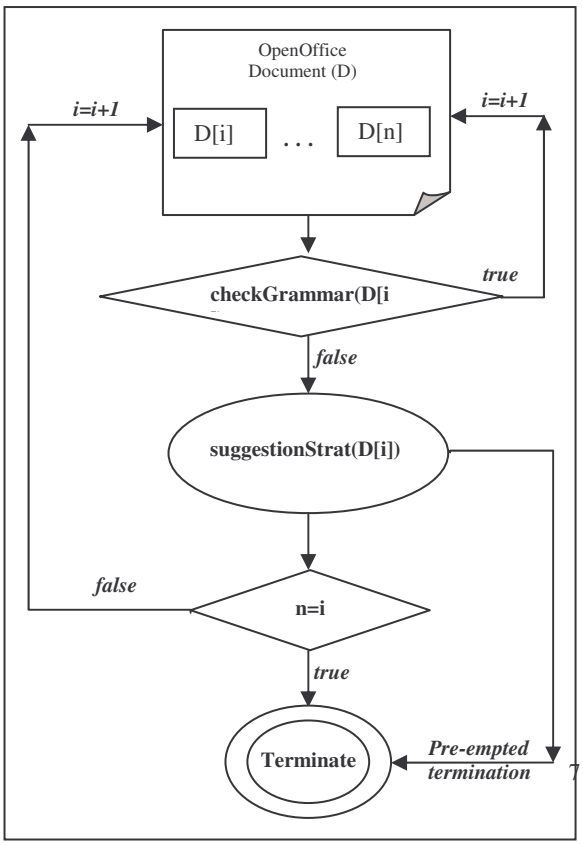


Figure 2: Grammar Checker Architecture.

Consider a document *D* with number of sentences equal to *n* where *n-1* is equal to the index of the last sentence in document *D*. Let *i=0* be the index of the first sentence found in document *D* and *D[i]* be the sentence pointed to by the index *i*. Let *i+1* be the index of the next sentence. Let **checkGrammar**(*s*) be the function that will accept a parameter *s* wherein *s* can be the sentence *D[i]*. The function will return true if the sentence *s* is grammatically correct. Let **suggestionStrat**(*m*) be the function that will return a sentence with appended POS of missing word or words in the sentence that would make the sentence correct.

Step 1: At *i=0*, get the sentence *D[i]*

Step 2: if **checkGrammar**(*D[i]*) returns true, consider the next sentence i wherein *i=i+1* and repeat Step 2 if *i* is equal to *n* goto Step 5. If **checkGrammar**(*D[i]*) returns false then continue to Step 3.

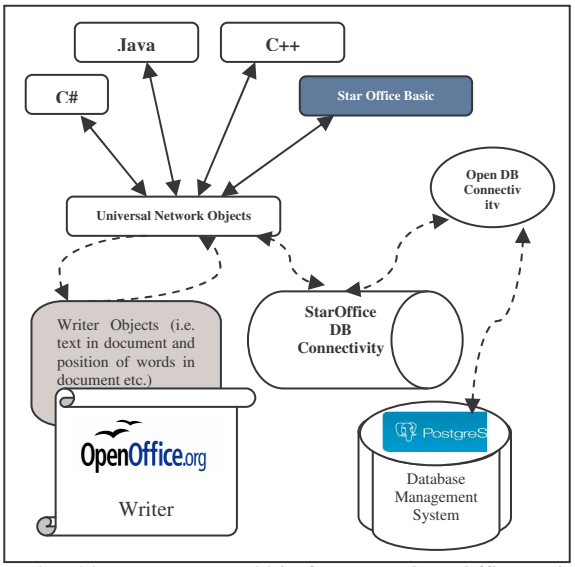
Step 3: The function **suggestionList**(*D[i]*) will display a corrected sentence with appended POS of missing words or display recommendation to rephrase sentence if needed. Apply the suggestion to sentence and do the necessary word replacement. (the process can be pre-empted or manually terminated by jumping to Step 5 or continue to Step 4)

Step 4: if *i < n* then consider next word *D[i]* wherein *i=i+1* and repeat Step 2 else goto step 5.

Step 5: terminate algorithm.

4.1.4. Add-in System Architecture

The different components used in the implementation of the add-in are depicted in Figure 3.



To be able to create an add-in feature to OpenOffice Writer, a programming language that supports UNO must be used to access and manipulate the elements of the OpenOffice writer document. There are four programming languages to choose from but Star Office Basic is the easiest to use compared to the other three. To be able to communicate with postgreSQL, Open Database Connectivity (ODBC) must be used. ODBC is a multi-platform driver that connects applications to supported DBMS and applications. Unfortunately, Open Office does not support ODBC because it has its own DBMS connectivity driver exclusive to Open Office applications. However, the Star Office Database Connectivity (SDBC) driver can connect to a registered ODBC definition making it possible for Open Office applications to communicate with postgreSQL via ODBC thru SDBC.

4.3. Evaluation Metrics

In the evaluation process, the input text is categorized having two types of words: correct and incorrect. Correct words are words that are accepted by Tagalog (excluding proper nouns not unless they are added to the lexicon). The system identifies a word as correctly spelled, if after stemming is applied the resulting root word is found in the lexicon.

The system finds misspelled words and flagged it with a pink wavy line. The evaluation is done using a separate program that automatically computes the total number of words found as *correct* and the words found as *misspelled*. It also computes the total execution time. Table 1 depicts the automated evaluation results in spell checking Tagalog documents having large number of words (example, books in the Bible).

Table 1. Automated Evaluation Results

Test Data	TIME (in seconds)			Total Number of Words	Correct	Error (Misspelled)
	Start	End	End - Start			
Book of Genesis	03:23:16	03:37:43	14 min and 27 sec	35,739	31,398	4,341 words or 12.14 %
Book of Obadiah	08:37:48	08:38:28	40 sec	671	625	46 words or 7.36%

The book of Genesis consists of 35,739: the system found 31,398 correct words and the 4,341 mispelled words or 12.14%. In Obadiah, the system found 671 words correct, and 46 mispelled words or 7.36%. The errors (misspelled) are caused by the lack of conformity with the lexical entries (that is, proper noun or absence of the root words in the lexicon). Misspelled words also include words that are over-stemmed and under-stemmed by TagSa. The only solution is to recognize words that cannot be handled by TagSa is to add the over-stemmed and under-stemmed words to the lexicon.

5. SUMMARY AND CONCLUSION

A Tagalog spell Checker and grammar checker was developed for OpenOffice Writer to aid in writing documents in Tagalog. The system's capability in handling large wordlist in the lexicon, powerful parsing and stemming power is due to the third party engine employed and enhancement made in TagSA, respectively.

The grammar checking that was incorporated in the system is capable of handling basic sentence structures of Tagalog. There is no program re-compilation needed since the program, as stored procedures, can be edited on the fly on the third party software's end without restarting Open Office or even the operating system. Currently, no grammar checker has been incorporated in OpenOffice Writer. It is still a research proposal for the up coming season by Sun Microsystems which was presented in Summer of Code Project 2006 [15].

The advantage of having postgreSQL as parsing engine for NL applications is its ability to store, manage and manipulate very large data. It is independent to applications like Open Office, thus avoiding interference to the functionality of Open Office applications. The disadvantage on the other hand is that you need to install postgreSQL along with Open Office and setup database connectivity to bridge the two.

While running on a corpus of 14,000 root words (plus the root words extracted from words with affixes processed by TagSa), we found that our system works with high accuracy. The misspelled words are all correctly detected. They are mainly due to the presence of proper nouns and non-existent of the root words in the lexicon. We are planning to take care of euphony and assimilation in near future.

6. IMPLICATIONS AND RECOMMENDATIONS

The wordlist in the lexicon can be further incorporated with more Tagalog root words. To include more grammar rules and enhanced suggestion strategy is also a necessary improvement for the grammar checker.

Other Philippine-type languages can be incorporated in the system which could be used for a web web-based document processing applications. An example of these applications is the google docs.

7. LITERATURE CITED

[1] Bonus, Don Erick J. (2003). A Stemming Algorithm for Tagalog Words. MS Thesis. De La Salle University, Manila.

[2] Dimalen, Davis (2004). AutoCor: Automatic Acquisition of Corpora of Closely-Related Languages from a Closed Corpus (MS Thesis). De la Salle University - Manila.

[3] Chaudhuri, Bidyut Baran (2004). Reversed Word Dictionary and Phonetically Similar Word Grouping Based Spell-checker to Bangla Text. Proc. 2nd International Conference on Information Technology for Applications (ICITA), China.
Available at:
http://www.emille.lancs.ac.uk/lesal/bangla.pdf

[4] Grishman, R., & Calzolari, N. New York University, New York, USA. Istituto di Linguistica Computazionale del CNR, Pisa, Italy.
Available at: http://cslu.cse.ogi.edu/HLTsurvey/ch12node6.html

[5] Hendricks, Kevin B. The Mail Archive.
Available at: http://www.mail-archive.com/dev@lingucomponent.openoffice.org/msg01312.html

[6] Lingucomponent Project (2001). OpenOffice.Org
Available at: http://lingucomponent.openoffice.org/

[7] Microsoft Corp., 2005.
Available at:
http://msdn.microsoft.com/office/technologyinfo/developing/overview/default.aspx.

[8] MIMOS Open Source R&D Group (2004).
Available at: http://opensource.mimos.my/?main=mimos/openoffice_spellchecker

[9] Moshagen, S., Pieski, T. & Trosterud, T. (2005). OpenSource Speller Technical Documentation.
Available at: http://www.divvun.no/doc/proof/Spelling/X-spell/index.html#MySpell

[10] Naber, Daniel (2003). A Rule-Based Style and Grammar Checker. Technische Fakultät, Universität Bielefeld.
Available at: www.danielnaber.de/languagetool/download/style_and_grammar_checker.pdf

[11] O'Neill, M.E. & Connelly, C.M. (2003). Spell Checking Using Hash Tables.
Available at: http://www.cs.hmc.edu/courses/mostRecent/cs70/homework/cs70ass9.pdf

[12] OSTG (Open Source Technology Group), (2006).
Available at: http://sourceforge.net/docman/display_doc.php?docid=29374&group_id=143754

[13] PostgreSQL
Available at: http://www.postgresql.org/

[14] Scannel, Kevin. (2005). An Gramadóir.
Available at: http://borel.slu.edu/gramadoir/

[15] SummerOfCode2006.
Available at: http://wiki.services.openoffice.org/wiki/SummerOfCode2006

[16] UNDP APDIP (2007). Fijian Spell Checker for OpenOffice.org.
Available at: http://www.apdip.net/news/fijianspellchecker/view

[17] Wikipedia (2006).
Available at: http://en.wikipedia.org/wiki/N-gram

9. ACKNOWLEDGEMENT

This research is being funded by the Philippine Council for Advanced Science and Technology Research and Development (or PCASTRD) under the Department of Science and Technology (DOST), Philippines.