An Open Source Solution to Spell Checking and Grammar Checking Add-in for Open Office Writer

EDITHA D. DIMALEN
DAVIS MUHAJEREEN D. DIMALEN

Abstract

Writing manuscripts, communications and other forms of documents that include Tagalog can be very difficult, and daunting task if there are no available tools that can check its spelling and grammar. Hence, a Tagalog spell checker and grammar checker add-in for OpenOffice Writer has been developed. Since Tagalog is a free-word order language, processing of text is very complex. This research addressed several issues in the development of the add-in. These word processing tools are being developed for OpenOffice Writer word processing application. OpenOffice Writer is a non-proprietary software (a free ware) that is readily available for all users. The spell checker and grammar checker checks the spelling of the word, incomplete sentences, awkward phrases, wordiness and poor grammar construction.

EDITHA D. DIMALEN, Associate Professor, College of Computer Studies, MSU-Iligan Institute of Technology, Iligan City; DAVIS MUHAJEREEN D. DIMALEN, Associate Professor, College of Computer Studies, MSU-Iligan Institute of Technology, Iligan City

The research has been tested and evaluated using a separate program that automatically computes the execution time in checking each document written in Tagalog. The result showed better results in spell check and grammar checking, respectively. It can also successfully spell checked words (inflected or not) as long as root words are found in the lexicon. It can also spell check words with hyphen or apostrophe. In addition, it can checked words that are not present in the lexicon due to the stemmer employed in the system. New words can be generated out of the given root words which is exponential. This is due to the stemmer employed in the system which means the lexicon is very powerful in handling a very large wordlist.

1.0 Introduction

The importance of written communication using word processing is a very important tool for better writing. It even let the worst speller to appear flawless; and is the first defense against typographical errors. As a result, the aim of electronic word processing has expanded. 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.

Currently, the developments of open source spell checkers are only available for foreign languages such as the 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].

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 complement the algorithm.

Development will be focused on how these elements would complement 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.

2.0 Spell Checking

Spell checking proceeds

2.1 Spell Checker Architectural Design

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

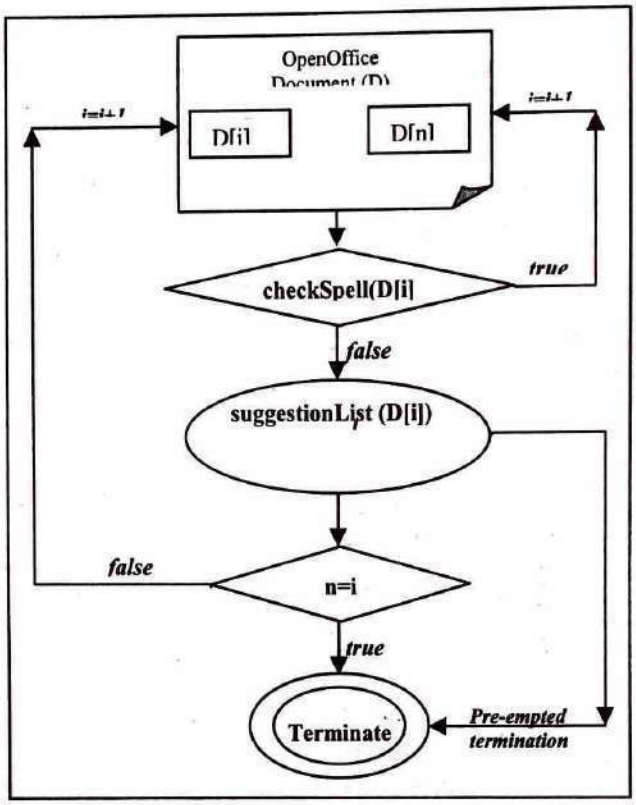

Figure 1: Tagalog Spell Checker Architecture.

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 words as replacement for the missspelled 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 or else go to 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.

2.2 Tagalog Stemmer

According to Surian ng Wikang Pambansa (2003) as mentioned by Bonus (2003) [1], some of the morphological features of the language are the complex system of affixes, the reduplication of a syllable in a word or the whole word itself, compounding, and its combination. Tagalog affixation has several types: prefixation, infixation, suffixation and circumfixation (Surian ng Wikang Pambansa, 1940; Gana & Matute, 1964; Schachter & Otanes, 1972; Komisyon ng Wikang Filipino, 1998, as cited by Bonus (2003) [1]).

Bonus (2003) [1], cited that prefixation is involves attaching a bound morpheme before the root word. Prefixes in Tagalog can be just 1 syllable (e.g. ma-) and as many as 7 syllables (e.g. ikinapagpapaka-). Suffixation involves attaching the bound morpheme at the end of the root word. There are four suffixes defined in Tagalog, namely: */-in/*, */-an/*, */-hin/*, and */-han/*. Normally, */-in/* or */-an/* is attached to words ending with a vowel, while */–hin/* or */–han/*, to words ending with a consonant. Infixation is a process where the bound morpheme is attached within the root word. There are two infixes defined in Tagalog, */-in-/* and */–um-/*. The base form of the word may undergo infixation of either /-in-/ or /-um-/ wherein the infix occurs after the first consonant of the stem. However, in circumfixation bound morphemes maybe present (as prefix, infix and suffix). Table 1 below shows the different affixation and its corresponding examples; and phoneme change and its examples also.

Table 1. List of Tagalog Affixations [1].

Affixation	Basic Example	Changes	
		Morphophonemic Change	Phoneme Change
Prefixation	maG + Sama → magsama (consonant / consonant) maG + Away → mag-away (consonant / vowel)	2 types: 1. Partial Assimilation Ex: /-ng/ - /m/ sing + puti →simputi 2. Full Assimilation Ex: mang + kuha →manguha	/d/ → /r/ ma + dami → marami
Suffixation	sikap + in → sikapin puna + hin → punahin		/d/ → /r/ + /-in/ or /-an/ lipad + in → liparin tawid + an → tawiran
Infixation	in + bili → binili um + sayaw → sumayaw	infix /-in-/ to be attached within a prefix and must appear immediately after the first consonant of the prefix Ex: in + pag + sama →pinagsama in + ipag + bili → ipinagbili	
Circumfixation	pa + in + punta + han → pinapuntahan ipag + um + hiyaw + an → ipaghumiyawan		

There are two types of reduplication in Tagalog: partial or full. In partial reduplication, certain syllables are duplicated to project the form of the stem. However, in full reduplication the entire stem is repeated to express continuative aspect, and happens in the derivational process. A single word also may have reduplicated syllables (prefixation or suffixation and infixation) all at the same time. Table 2 depicts examples of partial and full reduplications, respectively. The following rules are used in partial reduplication (Komisyon ng Wikang Filipino, 1998; Schachter & Otanes, 1972 as cited by Bonus (2003) [1]).

Table 2. List of Reduplication Rules [1].

	Rules	Examples
Partial Redup.	1. If the root of a two-syllable word begins with a vowel, the initial letter is repeated.	alis - a + alis → aalis iwan - i + iwan → iiwan
	2. In a two-syllable root, if the first syllable of the stem starts with a consonant vowel, the consonant and the succeeding vowel is reduplicated.	takbo - ta + takbo → tatakbo
	3. If the first syllable of the root has a cluster of consonants, two approaches can be used. This is based on the speaker's habit.	
	a. Reduplicates the first consonant and the first vowel of the stem.	plantsa + hin →pa + plantsa + hin - paplantsahin
	b. Reduplicates the cluster of consonants including the succeeding vowel of the stem.	plantsa + hin → pla + plantsa + hin - plaplantsahin
	4. In a three-syllable root, the first two syllables are reduplicated and hyphenated from the stem.	bahagya → baha + bahagya - baha-bahagya
Full Redup.	1. Reduplication and hyphenation of a two-syllable root without any affix.	araw - araw + araw → araw-araw
	Exceptions to this rule are words that consist of two segments that are alike, but are not hyphenated and treated as a whole.	alaala gamugamo
	2. Reduplication of an adjective prefixed by /ma-/.	ma + taas - ma + taas + taas → mataas-taas
	3. Reduplication of adjective in the superlative degree.	ka + liit - ka + liit + liit + an →kaliit-liitan
	4. Reduplication of nouns wherein the root is suffixed by /-an/, /-han/, /-ahan/, or /-anan/, to mean reduction or smaller than normal.	tao - tao + tao + han →tau-tauhan
	5. Inflectional reduplication of verbs suffixed by /-an/, /-hin/ or /-nin/.	bati - bati + bati + in →bati-batiin
	6. Reduplication of verbs affixed by /um/, /ika-/, /maki-/, /mapa-/, /magka-/, /makipag-/, /magpaka-/, /ipaka- /, etc.	um + alis - um + alis + alis →umalis-alis
	7. Reduplication of verbs prefixed by /pagka-/, and suffixed by the linker /-ng/ in the first part.	pagka + ani + ng - pagka + ani + ng + pagka + ani →pagkaaning-pagkaani
	8. Reduplication of an adverbial root. It is also the case with a root normally prefixed by /ka-/ or /pa-/.	ka + hapon + ng - ka + hapon + ng + ka + hapon →kahapung-kahapon

Compound word also exists in Tagalog in which two or more words are united to form a new meaning. It could be either with or without a hyphen. Table 3 presents certain rules that govern the forming of hyphenated compound words in the Tagalog Language (Surian ng Wikang Pambansa, 1940):

Table 3. Rules in Forming Compound Words [1].

Rules	Example
1. Missing words (one or more) →missing words will be replaced by a hyphen. Meanings will be retained but have another definition.	a. Missing *sa* or *ng*. For example: ningas-kugon (ningas ng kugon) b. Missing *ni*, *katulad ng kay*, or *katulad ng sa*. For example: kapit-tuko (kapit ng tuko or kapit katulad ng sa tuko)
2. Compounded two different words (with or without a linker in between). These compounded words can be of the following types: noun-noun, adjective-adjective, derived noun-noun, derived verb-noun, and adverb-noun.	a. Noun-Noun For example: matang-tubig bungang-araw b. Adjective-Adjective For example: magandang-pangit matabang-payat c. Derived Noun-Noun For example: kadaupang-palad kabigayang-loob d. Derived Verb-Noun For example: kamagandahang-loob kamasamaang-salita e. Adverb-Noun For example: biglang-yaman walang-hiya
Exemption: For a two different compounded word that establishes its own meaning different from that of the two words being combined, the compound word must be written as a whole.	For example: bahaghari hampaslupa hanapbuhay pataygutom

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]: Hyphen-Search Routine, Dictionary-Search Routine, /-in-/ Removal Routine, Prefix Removal Routine, /-um-/ Removal Routine, Partial Reduplication Routine, Suffix Removal Routine, and Full Reduplication/Compounding Routine.

2.3. Spell Checker Suggestion Strategy

The suggestion strategy employed in the system is based on n-gram. If a word in a document is found to be misspelled (that is, no match/root word found in the lexicon), corresponding suggestions are readily available. The nearest n-grams (sub-sequence of n items from a given word) will be displayed on the screen and one can choose among these the suggestions.

Each word found in the document will be processed by first checking if there are words in the dictionary containing a substring equal to the word being tested. If no match is found in the dictionary, the next substring of the word from left to right with length (n-1) will be looked up in the lexicon for a substring match. The process of taking, looking up the substring (n-i) continues until (n-i) is equal to 2 or if the maximum number of matches is achieved.

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"
Token = "_text_"

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

3.0 GRAMMAR CHECKER

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 look 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 unit 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].

3.1 Grammar Checker Architectural Design

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

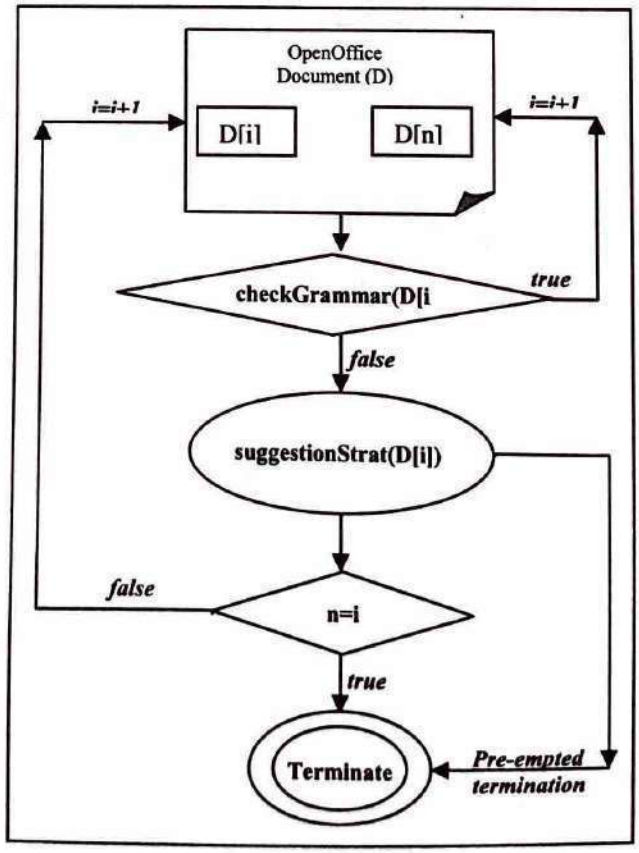

Figure 2: Tagalog Grammar Checker Architecture.

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.

3.2 Grammar Checker Suggestion Strategy

The system recognizes ungrammatical Tagalog sentences by evaluating the grammatical construction of each sentence. The evaluation of the grammaticality of the sentence proceeds by parsing each token in a sentence. The approach employed in the grammar suggestion strategy is simply based on the closest CFG rules stored in the database that matches that of the input sentence.

If the requirements specification is not met, the system will flag (through a green wavy line) that the sentence is ungrammatical and suggestions are displayed. The system display which tokens are missing as part of its suggestion strategy.

4.0. LEXICON AND GRAMMAR RESOURCE DEVELOPMENT

In Natural Language Processing, a lexical knowledge (that is, the knowledge about individual words) is essential. Lexical knowledge in encoded

though a lexicon in strictly formal structures. The lexicon has been long recognized as a critical system resource [4]. A lexicon is typically developed and encoded in a textfile.

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, or as a combination of the two. Lexicons are traditionally been built by hand specifically for the purpose of language analysis and generation. A more complex lexicon may include semantic information, such as a classification hierarchy and selectional patterns or case frames stated in terms of this hierarchy. This includes the typical subjects and objects for verbs, semantic features for nouns such as inanimate, human, etc. [13].

In this research, the lexicon is created using a third party engine, the Postgres SQL Database Management System (DBMS). It is being populated with Tagalog root words only with corresponding attribute (linguistic information). The necessary attribute being identifies is the Parts-of-Speech (POS) which is necessary for grammar checking.

Similarly, the grammar resource of the system is built in the Postgres SQL DBMS. It comprises a set of grammar rules for Tagalog.

5.0 SYSTEM'S IMPLEMENTATION

The implementation of the system is based on the final architectural designs discussed in the later sections. The different components used in the implementation of the add-in are depicted in Figure 3.

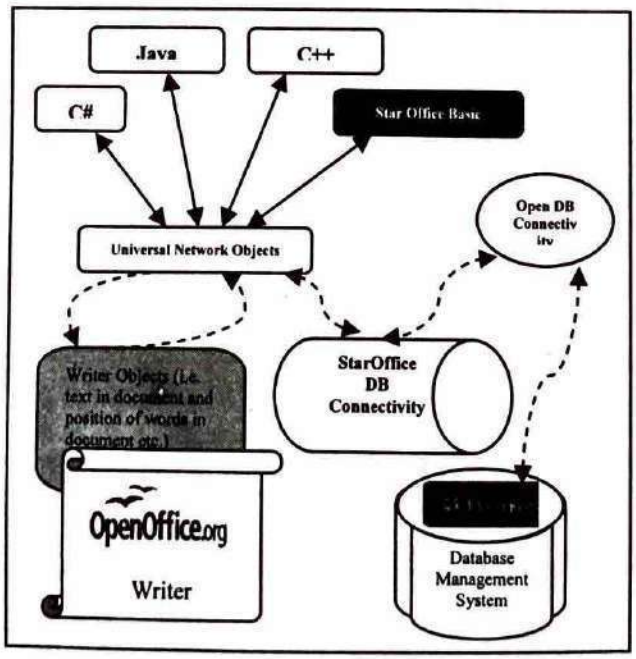

Figure 3. Architectural Design of Add-in System Components.

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 namely, C#, Java, C++ and Star Office Basic. In this research, 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.

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.

The mechanism employed is a new implementation strategy in developing an add-in in OpenOffice .org, which gives more comfort and flexibility in handling a very large resource.

6.0 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 4 depicts the automated evaluation results in spell checking Tagalog documents having large number of words (example, books in the Bible).

Table 4. 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.

7.0 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.

8.0 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 web, web-based document processing applications. An example of these applications is the google docs.

9.0 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

6.0 ACKNOWLEDGEMENT

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