A Two-level Engine for Tagalog Morphology and a Structured XML Output for PC-Kimmo

Hans J. Nelson

Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Linguistics Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/133

This Selected Project is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
A TWO-LEVEL ENGINE FOR TAGALOG MORPHOLOGY AND
A STRUCTURED XML OUTPUT FOR PC-KIMMO

by

Hans J. Nelson

A project submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Arts

Department of Linguistics and English Language
Brigham Young University
August 2004
BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a project submitted by

Hans J. Nelson

This project has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date ______________________________

Deryle W. Lonsdale, Chair

Date ______________________________

Alan K. Melby

Date ______________________________

Mark Edward Davies
As chair of the candidate’s graduate committee, I have read the project of Hans J. Nelson in its final form and have found that (1) its formats, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

______________________________  ______________________________
Date       Deryle W. Lonsdale
Chair, Graduate Committee

Accepted for the Department

______________________________
Lynn E. Henrichsen
Chair, Linguistics Department

Accepted for the College

______________________________
Van C. Gessel
Dean, College of Humanities
ABSTRACT

A TWO-LEVEL ENGINE FOR TAGALOG MORPHOLOGY AND
A STRUCTURED XML OUTPUT FOR PC-KIMMO

Hans J. Nelson
Department of Linguistics and English Language
Master of Arts

This paper presents a two-level morphological description of Tagalog for use in PC-Kimmo and a mechanism created for updating the results output from PC-Kimmo in order to meet the standards for current database and natural language processing applications.

There are two main research tasks presented in this paper which constituted this project. First, a complete morphological engine for Tagalog is presented. Next, a tool is introduced that takes the morphological engine output and stores it in XML format.
# TABLE OF CONTENTS

Chapter I: INTRODUCTION ................................................................. 1

Chapter II: BACKGROUND AND MOTIVATION ..................................... 2
  2.1. THE TAGALOG LANGUAGE ...................................................... 2
  2.2. COMPUTATIONAL MORPHOLOGY .............................................. 4
  2.3. TWO-LEVEL MORPHOLOGY AND PC-KIMMO ............................... 6
  2.4. FINITE STATE MODELING ...................................................... 8

Chapter III: A TWO-LEVEL MORPHOLOGY ENGINE FOR TAGALOG .......... 13
  3.1. TWO-LEVEL MORPHOLOGY ENGINE ......................................... 13
  3.2. THE TAGALOG RULES .......................................................... 13
  3.3. THE TAGALOG LEXICONS ...................................................... 18
  3.4. EVALUATION .................................................................. 23
  3.5. RUNNING THE TWO-LEVEL ENGINE ....................................... 27
  3.6. PROCESS OVERVIEW .......................................................... 29

Chapter IV: STRUCTURING THE OUTPUT ........................................... 31
  4.1. EXTENSIBLE MARKUP LANGUAGE .......................................... 31
  4.2. A DESCRIPTION OF KIMXM .................................................. 34
  4.3. RUNNING KIMXM ............................................................... 39

Chapter V: CONCLUSIONS ............................................................... 41
  5.1. FUTURE WORK ............................................................... 41

Chapter VI: REFERENCES ............................................................... 42

Appendix I: CODE DOCUMENTATION .............................................. 45
  1.1. CODE INTRODUCTION ......................................................... 45
  1.2. R4HANS.C CODE ............................................................. 45
  1.3. KIMXM VISUAL BASIC.NET CODE ...................................... 46

Appendix II: TAGALOG RULES FILE ................................................. 49
CHAPTER I: INTRODUCTION

1.1. INTRODUCTION. This paper presents a two-level morphological description of Tagalog for use in PC-Kimmo and a mechanism created for updating the results output from PC-Kimmo in order to meet the standards for current database and natural language processing applications.

There are two main research tasks presented in this paper which constituted this project. First, a complete morphological engine for Tagalog is presented. Next, a tool is introduced that takes the morphological engine output and stores it in XML format.
CHAPTER II: BACKGROUND AND MOTIVATION

2.1. THE TAGALOG LANGUAGE. There are roughly 101 languages spoken within the 7,200 islands of the Philippines according to the nationwide 1995 census conducted by the Philippine Government (National Statistics Office, 1997). Tagalog ranks at the top of these 101 languages as having the highest number of speakers of any language within the Philippines. Tagalog is a Meso-Philippine language which belongs to the Western Malayo-Polynesian branch of the Austronesian family. Filipino, based on Tagalog, became one of the official languages of the Republic of the Philippines in 1937. The majority of Tagalog speakers live on the central island of Luzon, though Tagalog is still spoken in many other areas of the Philippines.

Tagalog has been phonologically and lexically influenced by languages such as Spanish, English and Indonesian, however its morphosyntactic properties have remained wholly Tagalog. This fact is shown by a present-day language spoken in Manila with the adopting of English lexical items into a slang dialect language dubbed Taglish (Tagalog – English). English words like print are affixed with a verb morpheme and become na-print, along with English nouns like internet becoming nag-iinternet, meaning ‘you use the internet’ or ‘you are surfing the internet’. This same phenomenon has also taken place for borrowed Spanish lexical items introduced by colonization in 1565.

Morphology is the study of the meaning of individual units or morphemes of language and is concerned with the structure of words. Tagalog exhibits such morphological phenomena as affixation, stress shifting, consonant alternation, and reduplication for determining parts of speech, aspect, and voice. This includes the use of
various particles, prefixes, infixes, suffixes, and circumfixes (Ramos, 1971). This makes Tagalog much more morphosyntactically complex than a language like English which makes less use of markers and morphemes for determining parts of speech and focus as it does syntactic arrangements.

In Tagalog, verbal inflection to indicate aspect differs according to the affix class of the verb. *Ka-* marks a recently completed action of the verb. It is often followed by the adverbial particle *lang*. The recently completed aspect is formed by the affix *ka-* followed by the reduplicated first consonant-vowel of the verb root. This is seen with the prefix *ka-* added to the root *galing*, as in *kagagaling ko lang*, meaning ‘I just recently arrived’. The reduplication signals action started. Reduplication is the repetition of parts or all of the affix or of the root. For instance the root, *gawa*, could be infixed with *-um-* to produce the form *gumawa*, after which reduplication could form the word *gagawa* or *gumagawa*, thus reduplicating the initial consonant and vowel combination of the root. However, if the root begins in a vowel, like the word *abot*, loosely meaning ‘to pass’, then the infix with *-um-* is attached before the reduplicated initial vowel to form *umaabot*.

Because of the morphological complexities and the regularities of morpheme combinations, Tagalog seems well suited for computational analysis. Since the majority of the lexical items in the language are not words themselves, but composed of roots, affixes, and markers, computational morphological analysis provides a means by which only essential morphemes are stored and retrieved in the lexicon without unnecessary redundancy.
2.2. COMPUTATIONAL MORPHOLOGY. Computational morphology is the study of the computational analysis, synthesis, and treatment of word forms for eventual use in natural language processing (NLP) applications (Bosch et al., 2003). The purpose of such work is to aid in the effective means of the storage of words and lexicons, and provide time-efficient lookup capabilities. Computational morphological methods also give linguists the ability to create grammars and specify how word forms should be stored in lexicons. This analysis also helps answer fundamental questions of traditional morphology like which words are stored forms and which are based on derivational procedures in word formation.

Much work for natural languages has been done in the field of computational morphology. A number of systems have been developed, with a wide variety of approaches to processing, for use in NLP systems including analysis/generation of text (Sproat, 1992; Beesley, 2003), lexical analyzers (Silberztein, 1993), and speech processing (Mohri, Pereira, and Riley, 1996).

Various computational methods have been used to build comprehensive morphological descriptions for several languages. While no comprehensive list of such engines exists, some examples of various languages that morphological analyzers have been created for include English, Spanish, Russian, Swedish, Arabic (Beesley, 1997), Swahili, Zulu (Bosch et al., 2003), Aymara (Beesley, 2003), Finnish (Koskenniemi and Church, 1998), Hebrew, Japanese, Kasem, French, and Turkish (Oflazer, 1994). Indo-European languages have been the main focus such of descriptions while less common language families have been neglected (Roxas, 2000). Principled systems have proven to provide adequate morphological coverage for natural languages and are superior to
earlier “cut-and-paste” approaches to morphology (Karttunen et al, 1992). These approaches, which attempt to arrive at base forms through repeated modifications of the word, is still used for simple English morphology to remove the more common inflectional and morphological endings (Porter, 1980). This process does work for relatively uncomplicated English morphology where the affixes could easily just be ‘cut’ or removed without affecting the base form or requiring any additional knowledge of how morphemes form words. For instance, the word connect could have various suffixes which could be easily removed through such a method:

connect ed
connect ing
connect ion
connect ions

But this method is insufficient for processing morphologically complex languages that involve complicated morphophonemic variation across morpheme boundaries. A more adequate approach must be utilized for a morphologically complex language like Tagalog.

Though it could be argued that linguistic information is extensive on the languages spoken within the Philippines and much has been done theoretically, there is still a lack of important computational and empirical endeavors. While much work has been done in understanding Philippine languages, as Roxas states, “little (work) has been done on the computational aspect.” Roxas (2000) goes on to state that even the machine translation applications such as IsaWika! (Roxas 1999) and an LFG parser (Borra, 1999) application currently available may do a word-for-word translation or even translate a few declarative sentences of Tagalog, but “exclude morphological and syntactic aspects
of the language”. This is where a two-level approach, along with a morphosyntactic recognition engine which implements such an approach, becomes increasingly important for developing a method for Tagalog language processing.

2.3. TWO-LEVEL MORPHOLOGY AND PC-KIMMO. A two-level model is designed to generate and/or recognize words using finite-state techniques to process word structure correspondences between two representation levels: the lexical (L) and the surface (S), which are processed simultaneously (Koskenniemi, 1983). The lexical representation of a word is the underlying concatenation of its base morphemes, whereas the surface representation of a word is its normal orthographic form. Two-level processing thus synthesizes words from morphemes, and generates words from phonemes. These correspondences are thus bidirectional and in fact are executed in parallel. Two-level morphology involves relating these correspondences through the use of formal rules, providing a means of reconciling differences between L:S pairs.

Finite-state tables which correspond to these two-level rules are then processed to produce the L:S bidirectional word forms. To illustrate these principles let us look at a two-level lexical-surface correspondence shown below for the word happily:

<table>
<thead>
<tr>
<th>Lexical Representation:</th>
<th>happy + ly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface Representation:</td>
<td>happy 0 ly</td>
</tr>
</tbody>
</table>

The y:i spelling alternation in English can be described by a rule that reflects this correspondence. The two representations are formed with a symbol-to-symbol constraint. The word is made up of the lexical base happy and the suffix ly. Note that the ‘y’ is paired with the ‘i’ and the ‘+’ (morpheme boundary) with ‘0’ (nothing or zero on the
surface level). With rules that describe correspondences such as this one, complex morphological phenomena can be described.

Since the creation of two-level morphology, (Karttunen, 1983; Antworth, 1990) a number of natural languages have been described in these terms. A program called PC-Kimmo\(^1\) is one such implementation of two-level morphology named after its inventor Kimmo Koskenniemi (Koskenniemi, 1983). This program is designed to produce and/or parse words in which a word is represented as a L:S pair.

To discuss briefly just one of many two-level descriptions, a comprehensive two-level morphological description of Turkish (Oflazer, 1993) was created of word structures where he used 22 two-level rules and a lexicon of about 23,000 root words. This Turkish example is important in showing the practical computational usefulness of two-level modeling in such a highly morphosyntactic language. Another Philippine language of note other than Tagalog which is in progress at Brigham Young University is Cebuano\(^2\). The Cebuano language also ranks within the top nine of total number speakers in Filipino households. Farsi\(^3\) and certain Salish languages (Lonsdale, 2003) are also under development at BYU for use in two-level engines.

PC-Kimmo was formally introduced in the book by the same name, PC-KIMMO: *A Two-level Processor for Morphological Analysis*. Though this two-level modeling program developed by Koskeniemmi (Antworth, 1990) has been available since the early 90’s, it is still one of the only useful finite-state modeling programs available for field linguists. The book is intended for computational, theoretical, and descriptive linguists.

\(^1\) Evan Antworth, 1990, and later updated in 1995 to version 2 with added material in on-line format at the following website: [http://www.sil.org/pckimmo/v2/pc-kimmo_v2.html](http://www.sil.org/pckimmo/v2/pc-kimmo_v2.html). Maintained by SIL.

\(^2\) Currently under development by Jarren Bodily, Graduate Student in Linguistics, 2002.

\(^3\) Also under development by Jon Dehdari can be found at: [home.byu.net/~jmd56/download.html](http://home.byu.net/~jmd56/download.html)
It contains a description and explanation of two-level morphology, finite-state automata (FSA), and a how-to guide for developing engines using the software. It provides methods and strategies for hand-compiling rules into finite-state tables. The book also discusses Chomsky’s generative phonology vs. two-level finite-state phonology. One section especially relevant and useful for Tagalog provides rule examples for infixation and reduplication phenomenon in the language.

This two-level technology makes use of finite state modeling in order to perform the recognition functions used in PC-Kimmo. In order to understand more about PC-Kimmo and two-level morphology it is important to understand how finite state automata work. This next section will explain in further detail about FSA.

2.4. FINITE STATE MODELING. The PC-Kimmo program’s procedure for applying two-level morphology rules is based on a formal approach called finite state modeling. The use of finite-state machines in computing for language is very attractive; as Beesley (2003) states, “…the mathematical properties of finite-state networks are will understood, allowing us to manipulate and combine finite-state networks in ways that would be impossible using traditional algorithmic programs…” He continues by stating that they “…are computationally efficient for tasks like natural-language morphological analysis” and “can store a great deal of information in relatively little memory”.

It was C. Douglas Johnson who was the first to realize that finite-state transducers could be used to model phonological rewrite rules (Johnson, 1972). Using finite-state technology for morphological analysis has also been shown to be very effective.
A finite state automaton (FSA) recognizes (or generates) the well-formed strings of a *regular language* (a certain type of formal language) (Chomsky, 1965). Though most natural languages are not regular, regular languages are often taken as approximations for computationally-based processing of natural languages. An FSA is made up of states and arcs which represent the path between states. FSAs consist of at least an initial state and a final state with transition arcs directing movement from one state to the next. An FSA basically is thus a network where some set of paths traverse a finite number of states. Finite-state transducers (FSTs) are a special kind of FSAs that use these arcs to specify the relation between lexical and surface form correspondences and involve L:S pairs between states. Thus, transducers provide a means by which a single lexical symbol representation may be realized as another surface symbol in transitioning from one state to the next.

In order to more easily understand what is happening within a finite-state machine as it successfully transitions from one state to another an example should be given. For example, let us consider an artificial language whose alphabet is composed of only the letters or symbols *a* and *k* and the grammar being \{ak^n a\} where \( n \geq 0 \) (example based on Antworth 1990). This grammar could be written formally as:

\[
S \rightarrow a \ (X) \ a \\
X \rightarrow k \ (X)
\]

Words in this language are well-formed if they follow the pattern ‘aa’, ‘aka’, ‘akka’ and so forth as specified by the grammar. We shall call this language L1.
Figure 1 (left) shows an FSA for this language. The non-final states are represented by the single circles and the terminal states are represented by the double circles. This is the model or finite path through which words in L1 may be recognized. Let us step through the process of recognizing the word ‘akka’. The FSA will initialize when the first symbol or character of the string is ‘a’. Let us begin: 1. the first symbol ‘a’ is recognized and the automaton traverses the arc from state 1 to state 2; 2. the second symbol ‘k’ is recognized and the arc from state 2 back to state 2 is traversed; 3. the third symbol ‘k’ is recognized and the automaton traverses yet again the arc back to state 2; 4. the last symbol ‘a’ is recognized and the automaton traverses from state 2 to the final state 3. The initial input string ‘akka’ has now finished being processed. One can imagine the complexity with which a FST or finite state transducer can operate on a set of input strings. Extending the case above, a related transducer could also have L:S pairs on its arcs such as ‘a:a’ and ‘k:k’ which would simply map input letters to output directly. One could also extend L:S mappings to represent non-identical pairs.

This finite state machine can also be represented in a table called a finite-state table or state transition table. The above FSA is shown in table form in Figure 1 (right).
(with each column representing the alphabet of L1 and each row showing the state and ‘:’ showing final states). Language L1 is defined by this FSA table (where state 1 is the initial state and state 3 is the final state). The finite-state table is a product or representation of the finite state automaton to its left. This FSA was initially derived from the grammar rules for L1 used to recognize a valid symbol string or word.

The columns represent the arcs from one state to another state and the non-final states are marked with just a single period as opposed to the final states marked with a colon. The table is written to show or correspond to its FSA. To determine which state is the current state one must find the intersecting cell of the row and column. For instance, if you are in state one or the cell containing “1.” and the input is a you must find the intersecting cell of the column state and the row input to determine to which transition state you are to proceed. The number zero in a cell means that it is a trap state (where input is rejected) or that there is no transition from that state.

Finite state automata are the foundation upon which two-level morphological rules in PC-Kimmo are implemented. Through this input mechanism, a word may be broken down and analyzed at the level of single letter input.

In order for PC-Kimmo to recognize and/or generate word from a particular language there must be a description written for that language. Two types of files make up this description: the rules file and the lexicon file. The rule file contains two-level morphological rules for the language. Each word has two simultaneous representation levels: lexical (concatenation of morphemes), and surface (actual orthographic form). As mentioned earlier, the purpose for the rules file is to describe and resolve the differences between these levels (lexical and surface) through morphological rules.
Lexicon files on the other hand are used for listing the particular language’s lexical items such as various affixes and roots. Using these root and affix entries one may recognize and generate words in the language.

In the next chapter we will see how PC-Kimmo, two-level morphology, finite-state automata, and FS tables work together to produce a complete morphological engine for Tagalog.
CHAPTER III: A TWO-LEVEL MORPHOLOGY ENGINE FOR TAGALOG

3.1. TWO-LEVEL MORPHOLOGY ENGINE. PC-Kimmo Version 2\(^4\) is the system used for the Tagalog morphology engine. The engine together with the knowledge sources (rule files and lexicons) are the components to this system. This project focuses on mostly the rules and lexicon portion of the system, yet modifies slightly the recognition portion of the engine\(^5\). We will now examine each of the files required by PC-Kimmo (the rule file and the lexicon file) in more depth\(^6\).

It is important to note before continuing, that the purpose and design of such engines is to recognized and/or generate all possible parses of a given word. The engine has no method for disambiguating words or rejecting improper forms. In generating a word from its lexical forms, many more possibilities will likely result than from recognizing a word from its surface form. Since improper surface forms do not typically occur in language, the recognition of lexical forms from a single surface form will produce fewer possibilities.

3.2. THE TAGALOG RULES. Through two-level rules one may parse such extreme Tagalog words as *pinakamakapangyarihan*; which is made up of the morphemes *pinaka+maka+pang+yari+an*. Notice the insertion of the ‘h’ in the last morpheme. This discrepancy between lexical and surface forms is resolved using two-level rules as contained in a rules file.

\(^4\) http://www.sil.org/pckimmo/v2/pc-kimmo_v2.html  
\(^5\) See Appendix I for more information on modifications to the recognition code in PC-Kimmo  
\(^6\) A word grammar file also drives a third analytical component that creates and displays parse trees and feature structures for each word. This process was not a focal point of this project and therefore was not implemented.
One of the main user input components for the PC-Kimmo program is the \textit{rule file}\footnote{Grammar References for the creation of these rules were taken mostly from Teresita V. Ramos’ two books: Modern Tagalog and Tagalog Structures.}. This rule file contains two-level morphological rules for the language. Each word has two simultaneous representation levels; a lexical: concatenation of morphemes, and a surface: actual orthographic form. The rule file typically consists of an alphabet and phonological (or spelling) rules for the particular language being implemented\footnote{Summer Institute of Linguistics On-line at: http://www.sil.org/pckimmo/about_pc-kimmo.html}.

For example, to account for sound changes in affixation of the \textit{mang} prefix in Tagalog, its lexical form must be related to some surface form (where ‘+’ indicates a morpheme boundary, ‘0’ indicates a null element, and N symbolizes the consonant ‘ng’):

\begin{center}
\begin{tabular}{l}
\textbf{Lexical Representation:} & m a N + b i l i \\
\textbf{Surface Representation:} & m a 0 0 m i l i \\
\end{tabular}
\end{center}

This relation or \textit{special correspondences} of surface and lexical representation levels is accounted for through two level-rules written in the rule file. The rule in this case would account for the correspondences between N:0, +:0, and b:m. The actual rule as written in the rule file would look something like the following, if simplified from the actual Tagalog rule file:

\[ b:m \Rightarrow N:0 +:0 \]

This rule can be read as follows: when the lexical character \textit{b} is encountered by the parser it is realized as the surface character \textit{m} only in the following environment where ‘ng’ is realized as 0 followed by +:0 and then b:m. PC-Kimmo allows bi-directional input. Using this rule, PC-Kimmo may generate a surface form given a lexical form or recognize a lexical form given a surface form. In this project greater
importance has been given to the recognition of surface forms into their corresponding lexical forms (Antworth, 1991:31).

The Tagalog rule file created for this project contains roughly 22 rules. The majority of these rules described such language phenomena as insertion, deletion, reduplication, and infixation. The rules written were intended to account for as many of the regularities within the language as possible; exceptions were relegated to the lexicon which will be discussed later. For example, the rule below was written to show the insertion of an ‘h’ when the suffix -an or -in is added to a root, if that root ends in a vowel:

```
RULE " 0:h <=> V __ +:@ [a|i]  n" 7 7
 0 V + a n i @
h V @ a n i @
1: 0 2 1 2 1 2 1
2: 5 2 3 2 1 2 1
3: 0 2 1 4 1 4 1
4: 5 2 3 2 0 2 1
5: 0 0 6 0 0 0 0
6: 0 0 0 7 0 7 0
7: 0 0 0 0 1 0 0
```

Figure 2. A two-level rule for h insertion and its FST
Rules must be compiled into finite state tables for use in PC-Kimmo. Each rule within the rule file is composed of a RULE line (containing the special correspondence and the environment in which this correspondence occurs) and a finite state table created using KGen\(^9\). KGen (Miles, 1991) is a rule compiler written by Nathan Miles of Ohio State University and runs under MS-DOS, UNIX, and Macintosh. Its purpose is to compile two-level rules in order to create finite state tables for use in two-level morphology engines, such as PC-Kimmo. These tables can be created by hand only with considerable difficulty, but it is much more efficient to use a program like KGen to automate the process. Given a two level rule of the form \(x:y, a:b, c:d\) [i.e. one item in the left context and one item in the right context] an FST of 3 states (plus a ‘trap’ or ‘0’ state) is required, with transitions for \(x:y, a:b, c:d, \) and \(+\). An algorithm for this conversion is given in Antworth (1990:53-56) along with its proper implementation\(^{10}\).

KGen accepts as input a file of two-level rules and produces as output a file of state tables that is identical in format to PC-Kimmo's rules file. Any input that KGen does not correctly handle can be fixed by hand in the corresponding output file. KGen will take, for example, a rule like the one above for the insertion of an \(h\) and produce a 7 \(\times\) 7 table as also shown above (Nathan, 1991). KGen however does have its limitations. Rules created that consist of a transformation of more than a 31 \(\times\) 17 finite state table cannot be produced. In such cases as this, the rules must be broken down into small rules or transformations. Another limitation when compiling tables from rules in KGen is that each letter within a rule must be represented as single letter symbols. However in


Tagalog this is not always practical. For example, the orthographic letter 'ng' is represented by two symbols ‘n’ and ‘g’ (digraph) (i.e. ‘itisa’). In order for KGen to correctly read the rule, the single capital letter 'N' was used in place of the letter 'ng'. Once a table was produced, the digraph 'ng' was then replaced in the rule and table before being placed in the PC-Kimmo Tagalog rules file.

The following is a list of some of the many rules describing Tagalog created for this project. This rule shows assimilation to a nasal:

```
\{P,T,K\}:\{m,n,ng\} => ng: (+:0) (R:C E:V +:0) ___
```

It handles such concatenations in Tagalog as the prefix *mang* (indicates an occupational or habitual kind of work) with *bili* (buy), *takot* (frighten), and *kuha* (get):

- LF: maN+bili  maN+takot  maN+kuha
- SF: ma00mili ma00nakot ma00Nuha

This next rule deals with Tagalog infixation of either *-um-* (for actor focus verbs used to express simple acts) or *-in-* (for object focus verbs) when in combination with reduplication (determines aspect: completed, incompleted, or contemplated) for the consonants *b,d,g,h* in the conjugation of verbs:

"R:\{b,d,g,h\} => ___ [(0:i 0:n)|(0:u 0:m)] E:V +:0 \{b,d,g,h\}"

This rule handles such combinations of infixation and reduplications as in the words *pinipili* (to be choosing something), *pumipili* (is choosing), and *pipili* (will choose):

\[
\text{LF: R00E+pili \ R00E+pili \ RE+pili} \\
\text{SF: pini0pili \ pumi0pili \ pi0pili}
\]

This next rule also deals with Tagalog infixation of either *-um-* or *-in-* when in combination with reduplication for the initial letter of a word vowel *a*:

\[
\text{“E:a \Rightarrow (R:C) [(0:i \ 0:n) | (0:u \ 0:m)] (R:0) \_ +:0 (C:C) a”}
\]

This rule handles the following words in Tagalog, *aabot* (will pass) and *inaabot* (passing something):

\[
\text{LF: RE+abot \ R00E+abot} \\
\text{SF: 0a0abot \ 0ina0abot}
\]

A full listing of the Tagalog rules created in this project, without their finite state tables, can be found in Appendix II.

### 3.3. THE TAGALOG LEXICONS.

As mentioned earlier, the other main user input component for the PC-Kimmo program is the *lexicon file*. The lexicon file contains a list of words and morphemes along with their glosses. This file also contains data which encodes the morphotactic constraints for the language. PC-Kimmo’s lexical file works together with its rules file to recognize surface forms of words. The recognize function of PC-Kimmo is highly dependent upon the lexicon file. As Antworth states, “the
recognizer accepts as input a surface form, applies the phonological rules, consults the lexicon, and returns the corresponding lexical form with its gloss”\textsuperscript{11}.

PC-Kimmo performs the following functions: it cites the lexical representation of each lexical item, specifies the allowed order of morphemes that can make up a word (morphotactic constraints), and returns a gloss for each lexical item, word, or morpheme (Antworth, 1990:104).

Morphotactic constraints are implemented in PC-Kimmo using finite state methods. Morphotactic analysis implements alternation declarations in the main lexicon file (tlexA.lex). These alternations enforce the relatively simple morpheme orders of prefixes, reduplication, roots, and suffixes. Below is shown a portion of the main lexicon file with alternation definitions:

\begin{verbatim}
ALTERNATION Begin    PREFIX
ALTERNATION GetPrefix   PREFIX
ALTERNATION GetRedup    REDUP
ALTERNATION GetRoot     ROOT
ALTERNATION GetSuffix   SUFFIX
ALTERNATION End      End
\end{verbatim}

The alternation name reflects its position in a given word (i.e. ‘GetPrefix’). The sublexicon names stand for the types of lexical categories that may fill that slot (i.e. ‘PREFIX’). So, the alternation ‘GetPrefix’ contains the class ‘PREFIX’. According to the alternations listed above, a word is composed of the following finite state paths:

\textsuperscript{11} http://www.sil.org/pckimmo/about_pc-kimmo.html
The state labeled ‘Begin’ is the initial state. It represents the beginning of a word. There are two arcs leading from this state, labeled PREFIX and ‘0’. The arcs leading from the ‘Begin’ state show what can initiate a word. The ‘PREFIX’ arc can stand for any prefix in the lexicon. The zero arc (0) indicates that the control can traverse to the next state without consuming any input.

For instance, let us follow the recognition process in order to determine the morphemes and glosses of the word maglalaba. Below is an example of a sample entry within a lexicon file:

```
\lf mag+
\lx PREFIX
\alt GetRedup
\gl ActorFocus
```

Each entry specifies at least the lexical form ‘\lf’, the sublexicon ‘\lx’, the alternation ‘\alt’, and the gloss ‘\gl’. In this case, the mag+ lexical form is a prefix which can be followed by the alternation ‘GetRedup’ which includes the sublexical class ‘REDUP’. The next step is to go to the lexical subclass entries for ‘REDUP’ or reduplication as listed below:

![Figure 3. An FSA diagram of a Tagalog word](image-url)
The entry above shows the lexical reduplication that can take place in Tagalog and that such reduplication can be followed by another prefix or a root as shown by the alternations ‘GetPrefix’ and ‘GetRoot’.

The final entry shown above demonstrates the next path PC-Kimmo takes in its attempt to recognize the word maglalaba. Once the reduplication stage is passed, then its root is cited and its gloss is attached.

The Tagalog lexicon files created in this project are composed of three parts: the header file called tlexA.lex (contains alternations, field codes, and included lexicon files), the morpheme lexicon file called tlex1.lex (contains all defined affixes, reduplications, and some root entries – composed of roughly 65 entries), and the main lexicon which contains the bulk of lexical entries called tlex2.lex – composed of more than 5,500 entries. Much of this lexicon was produced from existing on-line sources such as HTML marked-up web pages. Building a large lexicon is highly important in this PC-Kimmo project. This lexicon was converted out of existing on-line dictionaries using regular expressions, into the format required by PC-Kimmo. Below is a sample from the HTML version which was converted using regular expressions:
Here are some of the expressions that were used along with the converted output lexicon file. This was mostly done using a search and replace method in the text editing program called TextPad\textsuperscript{12}. This program utilizes POSIX\textsuperscript{13} regular expressions.

Regular Expressions Sample:

```
(\lf([^$]*))\n\n(\lf([^$]*))\lx\l2([^$]*)(^\gl([^$]*))\n([a-z](^\gl([^$]*)))\n([0-9](^\gl([^$]*)))\n(^[^$]*)\n(^[^$]*)\n(^[^$]*)\n(^[^$]*[^$]*)\n(^[^$]*[^$]*)\n(^[^$]*[^$]*)\n([A-Za-z][^$]*)\n([0-9][^$]*)
```

Sample lexicon file format to which it was converted:

```
\lf a
\lx Root
\alt Suffix
\gl (intj.) an exclamation of sudden recollection
```

\textsuperscript{12} Located at: www.textpad.com
\textsuperscript{13} For more information visit: http://www.posix.com/posix.html or http://www.opengroup.org/onlinepubs/009695399/toc.htm
3.4. Evaluation. Evaluations of the coverage of the rules and lexicons files as a two-level description of Tagalog for PC-Kimmo were done using two methods: analysis using real text and an analysis of the coverage of particular morphological phenomena.

One such test of real text was done with a word list of approximately 895 words (tokens) from a Manila newspaper article. First, the words were extracted from the article as a word list of one lexical item per line with English words, numbers, and proper nouns being removed. Of the 895 tokens in the list, there were 323 types. The list of types was then run through the PC-Kimmo recognition engine. Of the 323 types, 86 forms were not recognized (26.6%). This means that the surface form to be recognized produced the output: “form not recognized: surface form”. In most of these cases the unrecognized word was simply not in the lexicon. For instance, the word “asuncion” meaning ‘the feast of the assumption of the blessed virgin’ was not a listing in the Tagalog lexicon. This is not surprising since the lexicon used for this project only contains roughly 5,500 lexical entries. Other forms not recognized were grammatical.

---

14 ABS-CBN News Articles; http://www.abs-cbnnews.com/images/news/microsites/TheCorrespondents/overseas911.htm -- English words, names, punctuation were redacted.
errors in word formation and certain morphological phenomena already known not to be covered in the rules.

In this project, it was more important that the various features and phenomena of Tagalog morphology were comprehensively described through their rules rather than for a complete lexicon in the language to be compiled. Because of this, the unrecognized words had less to do with the two-level description rules for Tagalog and more to do with the absence of a comprehensive lexicon file which was to be expected. The rules coverage therefore is much higher than that of the two files combined. This brings us to the second method of evaluating the two-level Tagalog engine’s coverage. Table 1 below shows a nearly complete list of sound changes in affixation in Tagalog with a description of each sound change and whether it is covered by the two-level engine. (X) stands for an exception to the general or typical sound change phenomenon. One other issue not dealt with or listed in Table 1 is stress shift due to suffixation. No stress features are used in the surface forms.

Table 1. A list of sound changes in affixation in Tagalog and their current function status in the system.

<table>
<thead>
<tr>
<th>Affix + (Base)</th>
<th>Symbol</th>
<th>Description</th>
<th>Lexical Form</th>
<th>Surface Form</th>
<th>Functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>mang + Labial</td>
<td>p</td>
<td>drops 'ng' and 'p' changes to 'm'</td>
<td>mang+pili</td>
<td>mamili</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>drops 'ng' and 'b' changes to 'm'</td>
<td>mang+bili</td>
<td>mamili</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>reduplicates vowel sequence of root</td>
<td>mang+RE+bili</td>
<td>mamimili</td>
<td>Yes</td>
</tr>
<tr>
<td>mang + Dental</td>
<td>t</td>
<td>drops 'g' and 't' changes to 'n'</td>
<td>mang+takot</td>
<td>manakot</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>drops 'ng' changes to 'n' and 'd' drops</td>
<td>mang+dalangin</td>
<td>manalangin</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>ng' changes to 'n' and retains 'd' (X)</td>
<td>mang+damo</td>
<td>mandamo</td>
<td>Yes (a few)</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>reduplicates vowel</td>
<td>mang+RE+takot</td>
<td>mananakot</td>
<td>Yes</td>
</tr>
<tr>
<td>Prefix</td>
<td>Vowel Change</td>
<td>Root Affix</td>
<td>New Word</td>
<td>Sound Change</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------------</td>
<td>------------</td>
<td>----------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>mang+</td>
<td>k</td>
<td>drops 'k'</td>
<td>mang+kuha</td>
<td>manguha</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>g</td>
<td>no changes</td>
<td>mang+gulo</td>
<td>manggulo</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>reduplicates second syllable</td>
<td>mang+RE+kuha</td>
<td>mangunguha</td>
<td>Yes</td>
</tr>
<tr>
<td>mang+</td>
<td>i</td>
<td>no change in vowels</td>
<td>mang+isda</td>
<td>mangisda</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>reduplicates vowel sequence of root</td>
<td>mang+RE+isda</td>
<td>mangisda</td>
<td>Yes</td>
</tr>
<tr>
<td>mang+</td>
<td>h</td>
<td>no changes</td>
<td>mang+huli</td>
<td>manghuli</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>reduplicates vowel sequence of root</td>
<td>mang+RE+huli</td>
<td>manghuhuli</td>
<td>Yes</td>
</tr>
<tr>
<td>mang+</td>
<td>l</td>
<td>ng' changes to 'n'</td>
<td>mang+likum</td>
<td>manlikum</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>reduplicates vowel sequence of root</td>
<td>mang+likum</td>
<td>manlikum</td>
<td>Yes</td>
</tr>
<tr>
<td>mang+</td>
<td>s</td>
<td>ng' changes to 'n' and 's' drops</td>
<td>mang+sayaw</td>
<td>manayaw</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>reduplicates vowel sequence of root</td>
<td>mang+RE+sayaw</td>
<td>mananayaw</td>
<td>No</td>
</tr>
<tr>
<td>pang+</td>
<td>p,b,m</td>
<td>ng' becomes 'm'</td>
<td>pang+butas</td>
<td>pambutas</td>
<td>Yes</td>
</tr>
<tr>
<td>pang+</td>
<td>t,d,n</td>
<td>ng' becomes 'n'</td>
<td>pang+damo</td>
<td>pandamo</td>
<td>Yes (most)</td>
</tr>
<tr>
<td>pang+</td>
<td>g,ng,h</td>
<td>no changes</td>
<td>pang+hiwa</td>
<td>panghiwa</td>
<td>Yes</td>
</tr>
<tr>
<td>ipang+</td>
<td>p,b,m</td>
<td>ng' becomes 'm'</td>
<td>ipang+butas</td>
<td>ipambutas</td>
<td>Yes</td>
</tr>
<tr>
<td>+an</td>
<td>V+an</td>
<td>when ends in vowel 'h' is inserted</td>
<td>kanta+an</td>
<td>kantahan</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>V+an</td>
<td>(X) some drop final unstressed vowels</td>
<td>laba+an</td>
<td>labhan</td>
<td>No</td>
</tr>
<tr>
<td>+in</td>
<td>V+in</td>
<td>when ends in vowel 'h' is inserted</td>
<td>kanta+in</td>
<td>kantahin</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>V+an</td>
<td>(X) some drop final unstressed vowels</td>
<td>bili+in</td>
<td>bilhin</td>
<td>No</td>
</tr>
<tr>
<td>ni+</td>
<td>ni+l</td>
<td>when base begins with 'l', infix -in- becomes ni-</td>
<td>in+linis</td>
<td>nilinis</td>
<td>Yes</td>
</tr>
</tbody>
</table>

There are also many morpheme concatenations in Tagalog which do not produce sound changes to the base or affix. Table 2 shows a nearly complete list of affixes that
occur in Tagalog that do not produce sound changes. The table also shows which are covered by the two-level engine.

Table 2. A list of affixation in Tagalog (no sound change) and their current function status in the system

<table>
<thead>
<tr>
<th>Affix</th>
<th>Lexical Form</th>
<th>Surface Form</th>
<th>Functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>-in-</td>
<td>Y+baha</td>
<td>bumaha</td>
<td>Yes</td>
</tr>
<tr>
<td>-um-</td>
<td>X+bili</td>
<td>bumili</td>
<td>Yes</td>
</tr>
<tr>
<td>mag-</td>
<td>mag+lakad</td>
<td>maglakad</td>
<td>Yes</td>
</tr>
<tr>
<td>ma-</td>
<td>ma+bait</td>
<td>mabait</td>
<td>Yes</td>
</tr>
<tr>
<td>ka-</td>
<td>ka+RE+alis</td>
<td>kaalis</td>
<td>Yes</td>
</tr>
<tr>
<td>napaka-</td>
<td>napaka+husay</td>
<td>napakahusay</td>
<td>Yes</td>
</tr>
<tr>
<td>i-</td>
<td>i+turo</td>
<td>ituro</td>
<td>Yes</td>
</tr>
<tr>
<td>ipag-</td>
<td>ipag+bili</td>
<td>ipagbili</td>
<td>Yes</td>
</tr>
<tr>
<td>pala-</td>
<td>pala+biro</td>
<td>palabiro</td>
<td>Yes</td>
</tr>
<tr>
<td>pag-</td>
<td>pag+kain</td>
<td>pagkain</td>
<td>Yes</td>
</tr>
<tr>
<td>maka-</td>
<td>maka+tao</td>
<td>makatao</td>
<td>Yes</td>
</tr>
<tr>
<td>nakaka-</td>
<td>nakaka+inis</td>
<td>nakakainis</td>
<td>No</td>
</tr>
<tr>
<td>taga-</td>
<td>taga+bundok</td>
<td>tagabundok</td>
<td>Yes</td>
</tr>
<tr>
<td>pinaka-</td>
<td>pinaka+maganda</td>
<td>pinakamaganda</td>
<td>Yes</td>
</tr>
<tr>
<td>kasing-</td>
<td>kasing+laki</td>
<td>kasinglaki</td>
<td>Yes</td>
</tr>
<tr>
<td>maki-</td>
<td>maki+abot</td>
<td>makiabot</td>
<td>Yes</td>
</tr>
<tr>
<td>p(m)aki-</td>
<td>paki+Y+bili</td>
<td>pinakibili</td>
<td>No</td>
</tr>
<tr>
<td>paki-</td>
<td>paki+abot</td>
<td>pakiabot</td>
<td>Yes</td>
</tr>
<tr>
<td>makapag-</td>
<td>makapag+basa</td>
<td>makapagbasa</td>
<td>Yes</td>
</tr>
<tr>
<td>-ng</td>
<td>wala+ng</td>
<td>walang</td>
<td>Yes</td>
</tr>
<tr>
<td>i- -in-</td>
<td>i+Y+tapon</td>
<td>itinapon</td>
<td>No</td>
</tr>
<tr>
<td>lka-</td>
<td>lka+apat</td>
<td>lkaapat</td>
<td>Yes</td>
</tr>
<tr>
<td>RE</td>
<td>RE+</td>
<td>reduplication</td>
<td>Yes (for all functioning forms above)</td>
</tr>
</tbody>
</table>
3.5. **RUNNING THE TWO-LEVEL ENGINE.** This section explains how to run PC-Kimmo from the command line with a recognition list as input in order to produce a recognition results list file. This project was developed and tested on only Windows XP Pro and is not guaranteed to work on other OS systems, therefore for this project a Microsoft Windows environment is assumed. It is recommended that the user have the following configuration:

1. Windows XP Professional SP1 with .NET Framework (Windows Update)
2. Internet Explorer 6 SP1 (Windows Update)
3. MSXML 4 SP2 (follow the link below)

Once your system complies with these specifications you may begin using the PC-Kimmo recognition engine (*r4hans.exe*) located on the *Project CD* in the following directory: “\VB7Kimmo2XML\KimXM\Kimmo Files”. To run the executable follow the instructions listed below (NOTE: before following these instructions, copy the entire contents of the *Project CD* onto your local hard drive):

**Running PC-Kimmo:**

1. Prepare your files for use as input to the program *r4hans.exe*. This means you must have your language *rule file* and *lexicon file* along with a recognition list or *rec list file* (each word to be recognized must be listed line-by-line with only a single word per line – See PC-Kimmo Documentation for more information). For this project, the Tagalog language files to be used are located in the following directory: “\VB7Kimmo2XML\KimXM\Kimmo Files”. There you will find the *trule.rul* and *tlexA.lex* files (*tlex1A.lex* and *tlex2.lex* are included (referenced) in
The final file, called *trec1.lst* is the recognition list file of Tagalog words to be used as input. This is a sample list made up mostly of the preamble to the Filipino Constitution.

2. Once you have located all three files needed (*trule.rul, tlexA.lex, and trec1.lst*), you are ready to run r4hans.exe from the command-line. To do this, go to: ‘start menu > run’. The ‘Run’ dialog box will appear; type ‘cmd’ in the ‘Open’ textbox field and click ‘OK’. This will open up a DOS-Prompt. You may also open a command prompt window by going to: start menu > all programs > accessories > command prompt.

3. At the command prompt change the directory to the local directory where the PC-Kimmo executable is located. For instance, if you placed the files directly on your C drive then the command would be: cd C:\VB7Kimmo2XML\KimXM\Kimmo Files. Now make sure you are in the correct directory by checking to see if *r4hans.exe* is located within your current directory. Type: r4hans at the command prompt. This should have returned a ‘usage:’ statement.

4. You are now ready to input your files as directed by the ‘usage’ line. The default is as follows (assuming all files are located within the same folder), type: r4hans trule.rul tlexA.lex <trec1.lst >RecOutput.lst. This will produce an output file called *RecOutput.lst* within your current directory. This output file will be used as the input file for KimXM.
3.6. PROCESS OVERVIEW. Now that the two-level engine portion of the project has been introduced and explained, we present a brief explanation of how each of these pieces we have discussed, along with the portions to be discussed, will work together to form the whole. The process is displayed in Figure 3 below. This figure shows each portion of the entire project working together to accomplish their final intended purpose.

The initial step in this process was the creation of a Recognition List as input to the PC-Kimmo recognition engine. In this case it is a list consisting of the Tagalog words *humihingi*, *kinakain*, *tao*, and *ng*. This list simply contains one word per line without accompanying white-space.

The second step in the process is the running of the PC-Kimmo recognition engine. This component utilizes the rules and lexicons described for a given natural language to ‘recognize’ the list of Tagalog words through a morphosyntactic parse which then associates each morpheme with its gloss as listed in the lexicons.

The third step in the process as displayed in Figure 3 shows the output PC-Kimmo generates as each word is run through the recognition function. An output list is produced by the recognition engine which will then be used as input for the next programming component of the process, called KimXM (Kimmo to XML), which is designed to structure the PC-Kimmo output. The next chapter will discuss this portion of the process and give further technical detail on KimXM and its purpose.
Figure 4. Project Overview Diagram

PC-Kimmo Recognition Engine

Parses and Glosses Morphemes (r4hans)

Rules
Lexicon

transforms PC-Kimmo output into valid XML

PC-Kimmo Output

XML Document and DTD

Recognition List

humihingi
kinakain
to
ng

Y+RE+hingi
um+RE+to
ask for

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE koml SYSTEM "koml v1.dtd">
<koml>
  <word id='1' lang='tagalog'>
    <surfaceType sf='humihingi'>
      <lexicalType id='1' lf='Y+RE+hingi'>
        <morpheme sf='Y'>
          <gloss sf='Y' id='1'>-um- verbal infix</gloss>
        </morpheme>
        <morpheme sf='RE'>
          <gloss sf='RE' id='2'>Reduplication incompled aspect for verbs</gloss>
        </morpheme>
        <morpheme sf='hingi'>
          <gloss sf='hingi' id='3'>to ask for</gloss>
        </morpheme>
      </lexicalType>
    </surfaceType>
  </word>
</koml>
4.1. EXTENSIBLE MARKUP LANGUAGE. The morphological results provided by these Kimmo-based systems have been very useful in helping linguists analyzing natural languages to a certain point. Yet PC-Kimmo fails to provide an output structure equal to the natural language processing application and data exchange requirements of today. Today’s applications require an open standard for data exchange not tied to any particular operating system or proprietary entity. One of the main objectives of the first Natural Language Processing Pacific Rim Symposium was “to provide NLP with deeper semantic structure clues and thus realize much more robust, higher precision NLP applications.”

Extensible Markup Language (XML)\textsuperscript{16} is a text format derived from SGML\textsuperscript{17} (ISO 8879). One of the main purposes of XML is for the exchanging of data; as the World Wide Web Consortium states, “…XML is playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.” XML is currently becoming increasingly used as the standard data exchange and storage format for almost all computer-related businesses. In the NLP world it is also taking root as a useful standard of information exchange (Grover et al., 2001).

XML is called extensible because it is not a fixed format language like HTML (a single, predefined markup language). XML is actually a metalanguage or a language for describing other languages. Because XML is a metalanguage, this allows an individual

\footnotesize\begin{itemize}
\item[16] XML specifications are maintained by W3C found on-line at: http://www.w3.org/XML/
\end{itemize}
to design their own customized markup languages for a limitless number of varying types of documents.

There are a variety of XML markup specification standards created for linguistic data and texts. The majority of these methods have focused on high-level text storage (i.e. paragraphs, phrases, sentences) for on-line publishing. The TEI (Text Encoding Initiative) Consortium\(^\text{18}\) is one such organization which provides standards for XML markup to linguistic texts. However, it does not appear to be helpful in providing detailed morphological markup standards. The EMELD\(^\text{19}\) organization (Electronic Metastructure for Endangered Languages Data), while not providing a single specific markup schema for morphological data, is devoted to the preservation and storage of language data in XML and does welcome and suggest different strategies for accomplishing this end. Other standards that do address low-level markup at the word level do so from a lexical or termbase theory approach. One major termbase XML markup approach is TBX. TBX (TermBase eXchange)\(^\text{20}\) is an open XML-based standard format for terminological data. This markup scheme is concept-oriented (terms are grouped and arranged by concept vs. headword) and provide no standards for morpheme-level linguistic data storage. These approaches do not define or propose a sufficiently detailed strategy for morphological data markup.

In order to overcome such markup schema limitations, this project introduces an XML markup standard. This markup language was designed specifically for use with PC-Kimmo called Kimmo Output Markup Language (KOML). This language conforms to a DTD which is specific to KOML.


\(^{19}\) [http://www.emeld.org/](http://www.emeld.org/)

\(^{20}\) [http://www.lisa.org/tbx/](http://www.lisa.org/tbx/)
A DTD or document type declaration “contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or DTD.”  A DTD is a formal description in XML Declaration Syntax of a particular type of document. It sets out what names are to be used for the different types of element, where they may occur, and how they all fit together. A DTD allows the user to constrain their XML document for the particular markup language being used, in this case KOML. The following is a segment of the *koml v1.dtd* file which validates the Kimmo XML output (also shown in figure 3):

```xml
<!ELEMENT koml (word)+>
<!ELEMENT word (surfaceType)?>
<!ATTLIST word
id   CDATA #REQUIRED
lang CDATA #IMPLIED>
<!ELEMENT surfaceType (lexicalType)+>
<!ATTLIST surfaceType
sf   CDATA #IMPLIED>
<!ELEMENT lexicalType (morpheme)+>
<!ATTLIST lexicalType
id   CDATA #IMPLIED
lf   CDATA #IMPLIED>
<!ELEMENT morpheme (gloss)?>
<!ATTLIST morpheme sf CDATA #IMPLIED>
<!ELEMENT gloss (#PCDATA)>
<!ATTLIST gloss
sf   CDATA #IMPLIED
id   CDATA #IMPLIED>
```

This DTD sets forth the elements and attributes that are allowed in the XML document it constrains. One can see from this DTD that there are six elements: koml (the

---

root element), word, surfaceType, lexicalType, morpheme, and gloss. Within each element is a set of attributes. For instance, within the ‘surfaceType’ element is the attribute ‘sf’. This is an optional attribute value within every ‘surfaceType’ element as denoted by the ‘#IMPLIED’ statement. A complete explanation of the DTD and its specifications is beyond the scope of this document. However the structure of the XML document, as it is constrained by the DTD, will be discussed shortly.

4.2. A DESCRIPTION OF KIMXM. Though the default output of PC-Kimmo is informative to users of the program at present, it is not structured such to be utilized by the growing number of computational linguistic applications requiring such data and is, in its present output, limited to non-NLP applications. The output of the PC-Kimmo engine does not adhere to or conform to any useful computational format; for example, it currently provides no means of tagging its output. This is a sample output format, taken from a Spanish recognition list, that PC-Kimmo returns:

```
coger
coJ+er   catch_seize_grab+(inf)

cogo
coJ+o    catch_seize_grab+(1SingPres)

coges
coJ+es   catch_seize_grab+(2SingPres)
```

The recognized word or surface form is first listed, followed by its corresponding lexical form with a gloss then directly to its right. PC-Kimmo does not identify by means of tagging or special marking in any way the displayed forms. The glosses for each lexical form are also not matched with their corresponding morphemes or stems. The
parentheses separating the gloss entries are not a product of PC-Kimmo, but of the authored lexicon description.

In order for morphological data output to be utilized by computational linguists today and more importantly in the future, it must be set up in regular relationships, be marked or tagged in some organized manner, and conform to a standard database or markup format. This necessitates the need for some module to produce usable output while at the same time utilizing the power of PC-Kimmo as a morpheme parser and glosser.

Because of these shortcomings, this project has included development of KimXM, a program designed and built in Visual Basic .NET. This particular piece of the project involves taking the structured output of PC-Kimmo and converting and mapping such output to an XML tagged document which can then be transferred more readily between linguists and displayed more effectively than in its current PC-Kimmo structured state. This system offers a suitable output mechanism assuring compatibility across linguistic research and search engine platforms along with an encoding method that makes such sharability possible (via databases). This XML document structure format is desirable by many organizations such as EMELD.

This XML tagged data file could, for instance, be mapped to specific applicable ontological meanings. This markup could then be mapped to the EMELD linguistic ontology, which expands a general ontology such as SUMO into the domain of linguistic concepts (Farrar, 2001). An RDF Schema could be utilized to link tags of a specific XML file to the EMELD ontology (Langendoen et al., 2001). This is just one such application for which PC-Kimmo’s current output structure is not suitable for.
the added ability of transforming PC-Kimmo data into XML through KimXM, this data now becomes useful.

As shown in Figure 3 of section 3.5., the output from PC-Kimmo is given as the input to KimXM which will, as previously stated, convert the output file into its final XML format. This data is now stored in an open standard which can now be displayed and transferred as desired by the individual user of the application. The final XML document produced by this process conforms to a schema specification called KOML. This markup language has been created specifically for PC-Kimmo data. The following shows the final XML output displayed in Internet Explorer:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE koml (View Source for full doctype...)>
  <koml>
    + <word id="1" lang=""
    + <word id="2" lang=""
    + <word id="3" lang=""
    + <word id="4" lang=""
  </koml>
```

Within the root element ‘koml’ there can be any number of ‘word’ elements with attributes ‘id’ and ‘lang’. A running total of how many words are contained within ‘koml’ is given by the ‘id’ attribute’. The ‘lang’ attribute is used to identify the particular language of the word.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE koml (View Source for full doctype...)>
  <koml>
    + <word id="1" lang="">
    - <word id="2" lang="">
      + <surfaceType sf="pagbabayad">
      </word>
    + <word id="3" lang="">
    + <word id="4" lang="">
  </koml>
```
Within each word, in this case word 2, can be only one ‘surfaceType’ element within a word. This element contains the ‘sf’ attribute representing the surface form of the word. So, here we see that word 2 is has the surface form pagbabayad.

```
- <koml>
  + <word id="1" lang="">
  - <word id="2" lang="">
    - <surfaceType sf="pagbabayad">
      - <lexicalType id="0" lf="pag+RE+bayad">
        + <morpheme sf="pag">
        + <morpheme sf="RE">
        + <morpheme sf="bayad">
      </lexicalType>
    </surfaceType>
  </word>
  + <word id="3" lang="">
    + <word id="4" lang="">
  </koml>
```

The next element level down is the ‘lexicalType’ element. There can be multiple ‘lexicalType’ or lexical forms for every ‘surfaceType’ or surface form within a word. In this case, pagbabayad has only one lexical form shown by the ‘lf’ attribute: pag+RE+bayad. Within each ‘lexicalType’ element can exist multiple morphemes, as determined by its lexical form. Here we see that the word is composed of three morphemes.
Finally, every ‘morpheme’ element contains a single gloss associated with that particular morpheme it is the daughter of. For instance, the morpheme pag means “on, if, or when.”

This hierarchal arrangement as shown above, which conforms to the KOML DTD, provides a means by where data mining, extraction, and searching can be done with the optionally of querying lexical forms, surface forms, or individual morphemes or concatenation groupings.

Currently, KimXM has produced correct XML output for not just Tagalog, but also from a two-level description of Farsi. This particular description was written specifically for use in PC-Kimmo and because of this KimXM was able to handle the data strings without complication (e.g. Romanized characters). Though this tool has not been tested with other two-level descriptions for PC-Kimmo, it was designed to be universally applicable to any language description written for PC-Kimmo. Thus it is
foreseeable that KimXM will work with all languages already described in the PC-Kimmo environment. For this reason KimXM should serve as a useful tool in updating traditional output of all natural language two-level descriptions for PC-Kimmo.

4.3. **RUNNING KIMXM.** This section explains how to run KimXM in order to create an XML document once you have produced a Recognition Output List File. If you still have open the DOS prompt you may close or exit now.

1. Browse to the following directory: `\VB7Kimmo2XML\KimXM\bin`. Within this folder is located the *KimXM.exe* program executable.

2. Begin running the program by double-clicking on the *KimXM.exe*.

3. A window will appear as shown below. In the Inputs frame under KOL: you will browse to the recognition output list file you just created called *RecOutput.lst*. Then you must select a location for the final output XML document by clicking the browse button and selecting a location.

4. When you have chosen an input and output file, you may click the Process button to create an XML document. Now locate your XML document and open it in Internet Explorer (NOTE: in order for the XML document to display properly in IE, the DTD called `koml v1.dtd` must be in the same folder as your XML document).
5.1. Future Work. One definite area for improvement and future work in this project is to improve the quality of the existing lexicon. Often entries are repeated in the current lexicon due to the format used by the original on-line source. This could be overcome manually, or in some cases by automating the process with search/replace methods, deleting the repeated entries. Also, it would be ideal for any realistic application of this project to obtain coverage of more than 90% to 95% of the language. To obtain such a high rate of efficiency, a few thousand more lexicon entries would need to be added and specific rules would need to be implemented for non-general correspondence patterns.

Another area for improvement is in the morphotactic constraints set forth in the main lexicon. In some cases a word is not recognized because of sublexicon classes not properly being implemented as alternation possibilities. More work should be done in lexicon class identification and alternation dispatching.

It is also the intent of the author to develop a confined set of lexical entries to test with a front-end speech recognition engine in order to begin the entire project process with not just simple text input, but with native speech input which then can be transcribed and transformed into valid XML.
CHAPTER VI: REFERENCES


HARMS, ROBERT T. *Two-level morphology as phonology: Parallel automata, simultaneous rule application, and the elsewhere condition.* Online: University of Texas at Austin. [http://www.utexas.edu/depts/linguistics/2lev-int.pdf](http://www.utexas.edu/depts/linguistics/2lev-int.pdf)


KIRAZ, GEORGE ANTON. *Arabic Computational Morphology in the West.* Bell Laboratories, Lucent Technologies. Murray Hill, NJ.


MOHRI, MEHRYAR, FERNANDO PEREIRA AND MICHAEL RILEY. 1996. *Weighted automata in text and speech processing.* In ECAI-96 Workshop, Budapest, Hungary. ECAI.


ROXAS, R. SANCHEZ. 2000. *Computational Linguistics Research on Philippine Languages*. Software Technology Department: De La Salle University, Manila, Philippines. [http://acl.ldc.upenn.edu//P/P00/P00-1074.pdf](http://acl.ldc.upenn.edu//P/P00/P00-1074.pdf)


1.1. **CODE INTRODUCTION.** This section serves to document integral portions of the source code functions of the C and Visual Basic code used within this project. This explanation consists of two parts: changes made to the C code of the recognition output function of PC-Kimmo and the source code written for KimXM in VB7.

1.2. **R4HANS.C CODE:** The code for PC-Kimmo’s recognition engine, written in C, has been slightly modified from its original version in order to re-structure its return strings for ease of processing in KimXM. This file is called `r4hans`. The changes made to the recognition function file called `r4hans.c` were mainly to structure the initial recognition results so as to make them useful to KimXM for conversion to XML. This regular output could then be used as the input for KimXM. The changing of the C code produced a file structure as shown below:

   SF:humihingi
   LF:Y+RE+hingi
   GL:-um-
   GL:RE+
   GL:(verb) to ask for

   The final XML transformation in KimXM took advantage of capitalized two letter codes, SF (surface form), LF (lexical form), and GL (gloss), produced by the modified C code in the PC-Kimmo recognition engine.
1.3. **KimXM Visual Basic.NET Code:** The KimXM Visual Basic .NET code consists of the following: *frmMain.vb, frmKimmo.vb, frmAbout.vb, Globals.vb,* and *modRec.vb.* *frmMain.vb* was developed as the main form interface which allows the user to input their PC-Kimmo Output List File and output their final XML Document (shown below).

![KimXM Form](image)

It contains open and save dialog boxes and a process button with a progress bar among other things. This is where the majority of the code which converts the text file into an XML document takes place. The next form *frmKimmo.vb* is a GUI for inputting your original recognition list and calling *r4hans.exe* for the outputting of the PC-Kimmo output list for input in *frmMain.* The functionality of this form is dependent upon a proper working compilation of the PC-Kimmo recognition code under Visual Studio. This can not yet been accomplished using Visual Studio, but was compiled using Cygwin\(^{22}\) under a Windows 2000 OS environment. The form is however shown below.

This form is intended to take the place of the command line operation of *r4hans.exe* and

---

\(^{22}\) Located at: www.cygwin.com
provide a user interface before you reach *frmMain.vb*. *modRec.vb* is used to call *r4hans.exe* within this form.

The *Globals.vb* contains certain global variables used in the project along with a *CleanXMLData* function which replaces all the 5 special xml characters with their appropriate entities. A sample of this important function is shown below:
Public Function CleanXMLData(ByVal astrData As String) As String

Dim strTemp As String
Dim chars As String
Dim i As Integer

' escape all 5 special xml characters: &, <, >, " , and '
astrData = Replace(astrData, "&", "&amp;")
astrData = Replace(astrData, "<", "&lt;")
astrData = Replace(astrData, ">", "&gt;")
astrData = Replace(astrData, """, "&#34;")
astrData = Replace(astrData, ",", "&#39;")

For i = 1 To Len(astrData)
    chars = Mid$(astrData, i, 1)
    If Asc(chars) < 128 Then
        strTemp = strTemp & chars
    Else
        strTemp = strTemp & "&quot; & Pad(CStr(Asc(chars)), "0", gn_Left, 4) & ";"
    End If
Next i
CleanXMLData = strTemp

End Function
APPENDIX II: TAGALOG RULES FILE

;trule.rul Version 2
;Tagalog PCKimmo Rule File

RULE "1 Consonant defaults" 1 23
   b c d f g h j k l m n ng p q r s t v w x y z @
   b c d f g h j k l m n ng p q r s t v w x y z @
   1: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

RULE "2 Vowel defaults" 1 6
   a e i o u @
   a e i o u @
   1: 1 1 1 1 1 1

RULE "3 Other Symbol defaults" 1 4
   + ' # @
   0 ' # @
   1: 1 1 1 1

RULE "4 Assimilation Correspondences" 1 7
   p b t d k g @
   m m n n ng ng @
   1: 1 1 1 1 1 1 1

;Nasal deletion and assimilation to nasal

RULE "5 ng:0 => __ (+:0) (R:C E:V +:0) :NAS"
   ;maN+bili
   ;ma00mili

   ;maN+takot
   ;ma00nakot

   ;maN+kuha
   ;ma00Nuha

;Assimilation to nasal

RULE "6 {P,T,K}:{m,n,ng} => ng: (+:0) (R:C E:V +:0) __"
RULE "7 ng:m => p a __ +:0 [p|b]"

RULE "8 ng:n => [p|m] a __ +:0 [[t|d]|l]"

RULE "9 X:0 => __ +:0 C 0:i 0:n V:V"

RULE "10 Y:0 => __ +:0 C 0:u 0:m V:V"

RULE "11 R:{b,d,g,h} => __ [(0:i 0:n)|(0:u 0:m)] E:V +:0 {b,d,g,h}"

RULE "12 R:{k,ng,m,n} => __ [(0:i 0:n)|(0:u 0:m)] E:V +:0 {k,k,p,t}:(k,ng,m,n)"

RULE "13 R:{p,r,s,t,y} => __ [(0:i 0:n)|(0:u 0:m)] E:V +:0 {p,r,s,t,y}"

RULE "14 R:{l,w,c,z} => __ [(0:i 0:n)|(0:u 0:m)] E:V +:0 {l,w,c,z}"

50
RULE "15 R:{f,j,q,v,x} => __ [(0:i 0:n)|(0:u 0:m)] E:V +:0 {f,j,q,v,x}"

;Vowel Reduplication before vowel initial word
;RE+abot
;0a0abot
;R00E+abot
;0ina0abot

RULE "16 E:a => (R:C) [(0:i 0:n)|(0:u 0:m)] (R:0) __ +:0 (C:C) a"

RULE "17 E:e => (R:C) [(0:i 0:n)|(0:u 0:m)] (R:0) __ +:0 (C:C) e"

RULE "18 E:i => (R:C) [(0:i 0:n)|(0:u 0:m)] (R:0) __ +:0 (C:C) i"

RULE "19 E:o => (R:C) [(0:i 0:n)|(0:u 0:m)] (R:0) __ +:0 (C:C) o"

RULE "20 E:u => (R:C) [(0:i 0:n)|(0:u 0:m)] (R:0) __ +:0 (C:C) u"

;Deletion of ‘h’
;Pag+sabi0+an
;Pag0sabih0an

RULE "21 0:h <=> V __ +:@ [a|i]  n"

;pam+bahala
;pam00ahala

RULE "22 b:0 <=> a m +:0 __"