Report on development and implementation of the Text-To-Speech Application

I. INTRODUCTION

After the thorough research of Khmer Language Processing in the concept of Text-To-Speech application, we arrive at the development and implementation phrase. In this phase we have developed one Text-To-Speech application that allows user to convert the sentences in Khmer language to the sound by taking into account all the results of the researches that have been done during phase 1.1, 1.2, and 2.1. It is important to note that in this phase we have built successfully not only a TTS application, but also the plug-in that allow this application to be integrated with the Internet browsers. Meaning, user can use the TTS application to read the Khmer text in internet browsers as well as in any web mail. In addition, we also developed our application for two platforms, one for Window and other for UNIX system. In this report we present about our works and results in phase 2.2. The difficulties, strength and weakness of our application will also be presented in this report. This report is divided into 6 parts; first is introduction. Second, it talks about the implementation of the TTS application in festival, in this section, we present about the TTS's application architecture and its components in festival. Section 3 introduces about the Diphone-databases and its difficulties while performing the clean up. Section 4 introduces the implementation of the word segmentation by using the algorithm which is the result from our previous research. Section 5 is the description of the result we obtained, included the weakness, strength and the points we have not completed. The last section will be the conclusion and our future plan for this project.

II. IMPLEMENTATION OF TTS IN FESTIVAL

This part describes the process of building Khmer Text-To-Speech using diphone concatenation technique by using Festival framework. The Festival Speech Synthesis System is an open-source, stable and portable multilingual speech synthesis framework developed at the Center for Speech Technology Research (CSTR), of the University of Edinburgh.

II.1 Development Process

II.1.1 Phone set

First of all, we have to define the phone set of Khmer language. Khmer phonetic inventory has been identified in phase 1.1 of this project "Research report on phonetic and phonological analysis of Khmer". We have defined and used the IPA to ASCII mapping below:

IPA	p	b	t	d	c	k	p^h	t^h	c^h	k^h	ʔ	m	n	ɲ	ŋ	r	s	h	l	w	j
ASCII	p	b	t	d	c	k	ph	th	ch	kh	?	m	n	n.	N	r	s	h	l	w	j

Table 1a IPA to ASCII mapping for consonants

IPA	i	i:	ə	ə :	u	u:	e	e:	œ:	o	o:	ɔ	ɔ:	a	a:	ɑ	ɑ:	ɨ	ɨ:	ɐ:	ɛ:
ASCII	i	i:	@	@ :	u	u:	e	e:	W:	o	o:	O	O:	a	a:	A	A:	ix	ix:	e.:	E:

Table 1b IPA to ASCII mapping for vowels

IPA	aɛ	aə	ɔo	uə	ao	iɜ	ɨə	iə	oa	ɛa
ASCII	aE	a@	Oo	u@	ao	iVx	ix@	i@	oa	Ea

Table 1c IPA to ASCII mapping for diphthongs

The phone set is defined in festvox/itc_kh_po_phoneset.scm.

II.1.2 Diphone database

After defining phone set we need to create diphone database which involves the following steps: identifying all possible diphones, creating non-sense or real words or phrases containing those possible diphones, recording those words or phrases and splitting those diphones. This process has been described in phase 2.1 of the project "Extension: Research report on Natural Language Processor (NLP) for Khmer TTS".
After defining and splitting the diphones, we follow steps described in [1] to build diphone database to be used in Festival. This diphone database is in group/polpc.group.

II.1.3 NLP Module

II.1.3.1 Word segmentation

As we do not use space between words in Khmer language, the first step is to do word segmentation. In our case we use maximum matching method by using the pronunciation lexicon in Khmer TTS as dictionary. The motivation behind this choice is that maximum matching is fast and as we use the pronunciation lexicon in Khmer TTS, we can assure that the pronunciation of all known words can be found in the lexicon. Word segmentation is described in "??Nipaul part??".

II.1.3.2 Sentence segmentation

An utterance structure is used to hold the information for what might most simply be described as a sentence. Sentence segmentation or utterance splitting is to split the whole text into utterances. This process is important as it allows synthesis to work bit by bit; allowing waveform

of the first utterance to be available more quickly than if the whole text was processed as one. In Khmer language, we use the symbol '។' (SIGN KHAN) to mark end of sentence, the symbol '៕' (SIGN BARIYOOSAN) to mark end of the text and the symbol '។ល។' (SIGN BEYYAL) to mean etc. We use the following decision tree to determine if a given token marks the end of an utterance.

```
(defvar kh_eou_tree
  '((lisp_max_num_tokens > 200)
    ((1))
    ((n.whitespace matches ".*\n.*\n\\(.\\|\n\\)*");; significant break (2
nls)
     ((1))
     ((name matches "--+")
      ((1))
      ((punc matches ".*[\\?:!;].*")
       ((1))
       ((name matches ".*\\(\\(។\\|៕\\)\\|។ល។\\)")
```

This is defined in festvox/itc_kh_po_utt_break.scm.

II.1.3.4 Text Normalization

This process is to convert non standard word (NSW) such as date, number into words. It is described in "??Samedi part??". It is defined in festvox/itc_kh_po_tokenizer.scm.

II.1.3.5 Word pronunciation

This part is about methods to find pronunciation of a word. The basic assumption in Festival is that we will have a large lexicon, tens of thousands of entries, that is used as a standard part of an implementation of a voice. Letter-to-sound rules are used as back up when a word is not explicitly listed. In addition to a large lexicon Festival also supports a smaller list called addenda this is primarily provided to allow specific applications and users to add entries that aren't in the existing lexicon.

II.1.3.5.1 Lexicons and addenda

The process of building lexicon has been described in phase 2.1 and adding new words to lexicon is described in "??Kimheng part??". The lexicon is defined in festvox/itc_kh_lex.out.
In the addenda, we defined the pronunciation of all Khmer characters, numbers and Latin characters. It is in festvox/itc_kh_po_lexicon.scm.

II.1.3.5.2 Letter to Sound Conversion

For letter to sound conversion, we used the statistical based described in the report of phase 2.1. It is defined in festvox/itc_kh_lts_rules.scm.

II.1.3.5.3 Syllabification and Stress Assignment

We implement syllabification and stress assignment algorithms described in the report of phase 1.1. It is defined in festvox/itc_kh_po_lexicon.scm.

II.1.3.5.4 Prosodic modules

II.1.3.5.4.1 Phrasing

Prosodic phrasing in speech synthesis makes the whole speech more understandable. Due to the size of people's lungs there is a finite length of time people can talk before they can take a breath, which defines an upper bound on prosodic phrases. However we rarely make our phrases this maximum length and use phrasing to mark groups within the speech. [1]
In our case we use the basic method provided by Festival: CART tree. For this method, a test is made on each word to predict it is at the end of a prosodic phrase. Our CART tree returns mB (small break), B (normal break), BB (bigger break) or NB (no break). The following tree is very simple and simply adds a break after the last word of a token that has following punctuation or we are more than 5 words from a punctuation symbol or previously identified break.

```
(define (since_punctuation word)
 "(since_punctuation word)
Number of words since last phrase break or beginning of utterance."
 (cond
  ((null word) 0) ;; beginning or utterance
  ((string-equal "mB" (item.feat word "pbreak")) 0)
  ((string-equal "0" (item.feat word "p.lisp_token_end_punc")) (+ 1 (since_punctuation (item.prev
word))))
  (t
   0)))
(define (until_punctuation word)
 "(until_punctuation word)
Number of words until next punctuation or end of utterance."
 (cond
  ((null word) 0) ;; beginning or utterance
  ((string-equal "0" (item.feat word "lisp_token_end_punc")) (+ 1 (until_punctuation (item.next word))))
  (t
   0)))
(set! simple_phrase_cart_tree_2
'((lisp_token_end_punc in ("?" "." ":" "%"))
  ((BB))
  ((lisp_token_end_punc in ("'" "\"" "," ";" "%"))
    ((B))
    ((n.name is 0) ;; end of utterance
      ((BB))
      ((lisp_since_punctuation > 5)
```

Phrasing module is defined in festvox/ itc_kh_po_phrasing.scm.

II.1.3.5.4.2 Accent/Boundary Assignment

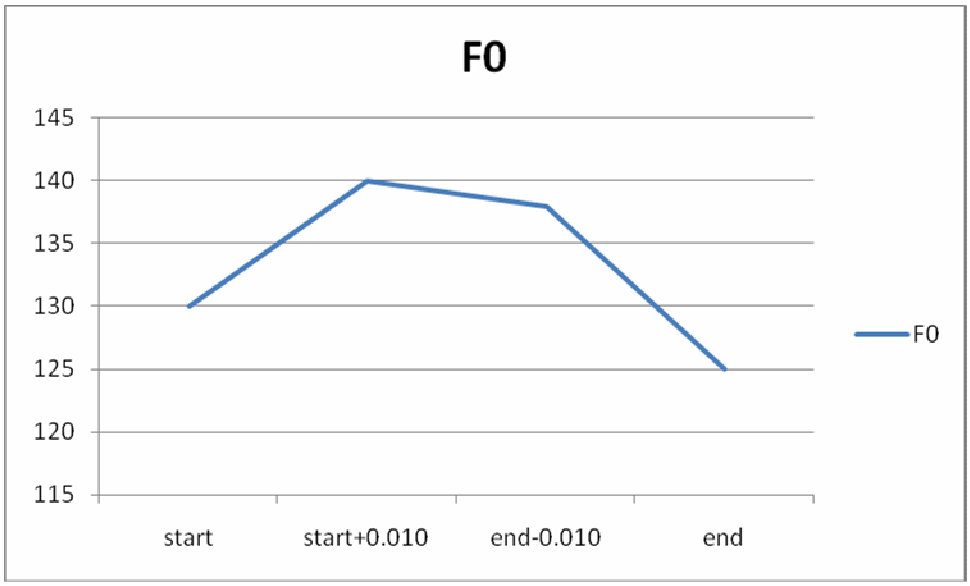
We just use the simple method described in [1] that is adding a simple hat accent on lexically stressed syllables in all content words.

```
(set! itc_kh_accent_cart_tree
 '
 (
  (R:SylStructure.parent.gpos is content)
   ( (stress is 1)
       ((Accented))
       ((NONE))
   )
 )
```

It is defined in festvox/itc_kh_po_intonation.scm.

1-1.1.1.1 F0 Generation

We use F0 generation by rule. For each accented syllable, F0 contour is generated as shown in the figure below:



Lisp function corresponding to F0 generation above is:

```
(define (itc_kh_po_targ_func1 utt syl)
 "(itc_kh_po_targ_func1 utt syl)
Simple hat accents."
  (let ((start (item.feat syl 'syllable_start))
        (end (item.feat syl 'syllable_end))
        (ulen (item.feat (utt.relation.last utt 'Segment ) 'segment_end))
        nstart nend fustart fuend fuend fstart fend)
        (set! nstart (/ start ulen))
        (set! nend (/ end ulen))
        (set! fustart 130)
        (set! f2 140)
        (set! f3 138)
        (set! fuend 125)
        (set! fstart  (+ (* (- fuend fustart) nstart) fustart))
        (set! fend    (+ (* (- fuend fustart) nend) fustart))

    (cond
      ((equal? (item.feat syl "R:Intonation.daughter1.name")
"Accented")
       (list
        (list start fstart)
        (list (+ start 0.010) f2)
        (list (- end   0.010) f3)
        (list end   fend)
            ))
      ((not (item.next syl))
       (list
            (list end fuend)))
```

F0 generation is defined in festvox/ itc_kh_po_f0model.scm.

II.1.3.5.4.3 Duration

For duration, we define average duration for each phone. We take average duration from what we have done in phase 1.1 with a little adaptation. As phone duration can vary in different contexts such as phrase initial, phrase final, stressed or unstressed syllable, we define a simple decision tree that returns a multiplication factor for a segment as follow:

```
(set! itc_kh_po::zdur_tree
 '
((name is pau)
 ((emph_sil is +)
  ((1.0))
  ((p.R:SylStructure.parent.parent.pbreak is BB)
   ((2.0))  ;;bigger break
   ((p.R:SylStructure.parent.parent.pbreak is B)
    ((1.0))  ;;normal break
    ((0.8)))))  ;; ;;small break

   ((R:SylStructure.parent.R:Syllable.p.syl_break > 1 ) ;; clause initial
    ((R:SylStructure.parent.stress is 1)
     ((1.4))
     ((1.2)))
    ((R:SylStructure.parent.syl_break > 1) ;; clause final
     ((R:SylStructure.parent.stress is 1)
      ((1.4))
      ((1.2)))
     ((R:SylStructure.parent.n.parent.name is 0)          ;; last syllable
      ((R:SylStructure.parent.p.parent.name is 0)         ;; first syllable
       ((1.5))     ;; word with only one syllable
       ((1.4)))   ;; last syllable
      ((R:SylStructure.parent.stress is 1)
       ((ph_vc is +)
```

II.2 Integration with various platforms

Festival offers a powerful platform for development and deployment of speech synthesis system. For Linux, most of the distributions come with Festival pre-installed, so it is straightforward to integrate Khmer voice in such system.

Motivated by the work carried out in the Welsh & Irish Speech Processing Resources (WISPR) project [3], steps were taken to integrate Festival along with the Sinhala voice into the Microsoft Speech Application Programming Interface (MSSAPI) which provides the standard speech synthesis and speech recognition interface within Windows applications [4]. Thanks to this API, any window application can easily use Khmer voice.

II.3 Conclusion and Future Work

We have described the development and evaluation of the first Text-To-Speech for Khmer language using Festival framework. As we do not have data to train probabilistic model, in prosodic modules, we just use simple rules.

Future work will mainly focus on improving naturalness of the synthesizer. This involves correct pronunciation of words, quality of diphone database and prosodic modules. Currently we are verifying to adding more words (collecting from Internet) to the lexicon. We also plan to have students involved in improving the synthesizer through school projects and internship.

III. DIPHONE DATABASES

Referring to the previous report of "the phonetic analysis of Khmer sound", Khmer phonetic can be divided into 21 consonants, 21 vowels (12 long vowels and 9 short vowels) and 10 diphthongs. Hence, if we do the combination for those 52 different sounds, we will get around 2500 diphones; however, not all the 2500 diphones are used in Khmer language. Khmer language is rich, there are hundred thousand of words, but we still borrow some new words from other language especially from French or English. With this reason and to avoid any problem for our application when the new words appear, we decided to build up a diphone databases that consists of 2704 diphones. The details explanation of this section is below

Suppose that:

C = consonant
LV = long vowel
SV = short vowel
D = diphthong
P = silent

Diphone form:

1. C - LV : 21 * 12 = 252

2. LV - C : 12 * 21 = 252
3. C – SV : 21 * 9 = 189
4. SV – C : 9 * 21 = 189
5. C – D : 21 * 10 = 210
6. D – C : 10 * 21 = 210
7. C – C : 21 * 21 = 441
8. LV – SV : 12 * 9 = 108
9. LV – D : 12 * 10 = 120
10. LV – LV : 12 * 12 = 144
11. SV – SV : 9 * 9 = 81
12. SV- D : 9 * 10 = 90
13. SV – LV : 9 * 12 = 108
14. D – SV : 10 * 9 = 90
15. D – D : 10 * 10 = 100
16. D – LV : 10 * 12 = 120
17. P – C : 21
18. C – P : 21
19. P – LV : 12
20. LV – P : 12
21. P – SV : 9
22. SV – P : 9
23. P – D : 10
24. D – P : 10
25. P – P : 1

So the total diphone is 2809. However some word we borrow from English doesn't exist in our phonetic here, for example the phone "er" in the word "teacher" in English. So to make our speech processing tools handle this thing, supplemental diphones are needed.

III.1. Phonetic symbol

Some specials character in the real phonetic are difficult to do the treatment in computer so we propose some symbol for all phonetics to solve this problem.

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
ប៉ព	បបិ	ផភ	តទ	ដឌ	ថធឋឍ	ចជ	ឆឈ	កគ	ខឃ
P	b	ph	t	d	th	c	ch	k	kh
P	b	ph	t	d	th	c	ch	k	kh

C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21
អអ	មម៉	ណន	ញញ	ងង	សស	ហហ	រ	លឡ	វ	យយ
ʔ	m	n	ɲ	ŋ	s	h	r	l	w	j
?	m	n	n.	N	s	h	r	l	w	j

Table-01 Consonants symbol

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
◌ិ	◌ី	◌ឹ	◌ឺ	◌ុ	◌ូ	◌ិ	◌ោ	◌ឺ	◌ុ
អ◌ិ	អ◌ី	អ	អ◌ឺ	អ◌ឹ	អ◌ឺ	អ	អ	អ	អ
I	i:	ə	ə:	u	u:	E	e:	Œ:	o
I	i:	@	@:	u	u:	E	e:	W:	o

V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
◌ោ	អ◌ៈ	អ◌	◌ាៈ	◌ា	អ◌ៈ	អ	◌ឺ	◌ឺ	◌ោ	◌ៃ
អ◌ុ	អ◌ឹ	អ◌ឺ	អ	អ◌	អ	អ◌	អ◌ឹ	អ◌ឺ	អ◌ឹ	អ◌ឺ
o:	ɔ	ɔ:	a	a:	ɑ	ɑ:	ɨ	ɨ:	e:	ɛ:
o:	O	O:	a	a:	A	A:	Ix	ix:	e.:	E:

Table-02 Vowels symbol

D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
◌ៃ	◌ើ	◌ូ	◌ួ	◌ោ	◌ៀ	◌ឿ	◌ា	◌ាៈ	◌ាៈ
អ◌	អ◌	អ◌	អ◌ឺ	អ◌	អ◌ឺ	អ◌ឺ	អ◌ឺ	អ◌ឺ	អ◌ឺ
a:ɛ	a:ə	ɔ:o	u:ə	a:o	i:ɜ	ɨ:ə	i:ə	oa	ɛa
aE	a@	Oo	u@	ao	iVx	ix@	i@	oa	Ea

Table-03 Diphthongs symbol

III. 2. Non sense word

The technique to record all khmer diphones is to create non sense word which include each diphone inside. To get the best diphone signal we add unnecessary word before and after the diphone which is needed in our project.

- **C-LV and LV-C**

The format C – LV and LV – C can be combining by using a non sense word C - LV - C
Example:

តាប៉ែប៉ (ta pe: pe:)

<ta> is an unnecessary word
<pe:> is diphone we need in the format of C – LV
<e:p> is another diphone we need in the format of LV- C
<e:> the last e: is also an unnecessary sound

- **C-SV and SV-C**

The format C – SV and SV – C could not be combining by using a non sense word C – SV- C, because between SV and the last C there existe an embedded consonent sound C11 (ʔ). So we need one non sense word for each format C – SV and SV – C.

តាបិបិ(ta pE pE) we get <pE> diphone

តាបុិពតា (ta pEp ta) and we get <Ep> diphone

To get a complete non sense words, reference to the non sense words document.

III.4. Remark

After recording all the non sense word we can uses the praat software to notice starting point, middle point and ending point of a diphone in one non sense word signal. The problems we face in this phase are:

- Incorrect phonetic in the lexicon library file
- Define the starting point and the ending point of a diphone from the non sense word signal
- The duration of the vowel (short or long)
- Some mistake during recording

To solve the problems above, we check the incorrect diphones and do the record again, incorrect phonetic can be modified in the lexicon library file, short and long duration of vowel will be applied by using the technique of how to make an intonation of the word and add duration to those vowels.

By using the words in khmer lexicon library we found out some diphones mistake such as:

- C-P and P-C groups diphones
- C-@ and C-ix groups diphones

And some particular diphones (i-s, i-r, r-ch, r-a:, j-i:, j-i@, ?-?)

Since the format of the mistake has been noticed, we need to create new diphones and do the record again and using the same process as the previous task. However, to improve our sound quality as it is the weak point of using diphone to generate speech, we had collected all the phrases from some khmer unicode web sites about 75Mb unicode text file and search for the missing word to add more lexicons which does not exist in our lexicon library and check the existing words in both text from those web sites and our lexicons to test the phonetics of frequency used lexicons. In conclusion, in order to improve the quality of the sound we should collect of the missing words to add to our lexicon library with good phonetic associate to them and have a good non sense words as well as notice a good position of the diphone signals.

IV. WORD SEGMENTATION

Normally, in all Text-To-Speech applications, we always find word segmentation. Word segmentation is a process in which a whole sentence is divided into a small unit of language which is called **word**. In this section, we will present the implementation of word segmentation in our application. The previous report shows that we have defined a solution to enhance segmentation process's speed based on the implementation and solutions provided by PAN Localization Cambodia. However, in our implementation period, we decided to implementation the segmentation process by using our implementation algorithm.

In this section, we divided into two parts. For the first part, we will introduce about our new algorithm whereas in the second one we will present our implementation and the result we have during the implementation.

IV.1. Algorithm

As the matter of fact, there are many algorithms provided to solve the problems which can be found in the segmentation process. However, we are interested in two algorithms where the first one is Maximum Matching Algorithm (MMA) – which provides a fast segmentation process but its accuracy is not acceptable – and the second one is Word Bi-Gram Algorithm (WBGA) – which provides a good accuracy but it is not fast. The choice of word segmentation algorithm is importantly based on its speed and its accuracy result. In [1], PAN Localization Cambodia have implemented word segmentation algorithm using the two algorithms mentioned above. They have introduced the use of Khmer Character Cluster (KCC) which is considered as the smallest unit of the segmentation. This KCC is used to generate all possible segments and it can provide another advantage because the boundary of the KCC can be the boundary of a word which speeds up the matching process in the segmentation. Then, they have presented the notion of Khmer Common Expression (KCE) which is a representation of all words with the same pronunciation but different spelling. And also, it is a solution to deal with misspelled word identification issue. However, by applying these two notions into the segmentation process will reduce the speed and it increases the complexity of the algorithm. Moreover, even with the solution provided in the previous report, we are sure that it will not speed up the process reaching what we are expecting. Hence, we need to define another method to implement these algorithms.

In our case, we have been using MMA because we think that speed for the process is very greatly important. Moreover, we have defined another solution to solve the problem of accuracy which is the originality of the algorithm. Our solution is based on our notice to the result of our test. Notice that, instead of using the smallest unit KCC, we use the advantage of Khmer character. We found that after the word segmentation process words that are not matched in the dictionary are separated into Khmer characters such as consonants, vowels, various sign… etc. Therefore, we defined a process whose task is to combine all separated Khmer characters into a

word. In Khmer language, there are some characters such as ុ ូ ួ cannot be alone and they must be standed with the next character. Hence, the process needs to take into account this notice. Our implementation does not consider the misspelled word identification issue because in our lexicon there are definition of word of that type.

Here below, we present globally our using algorithm for Khmer word segmentation. As input for the algorithm, there can be a Khmer sentence or name of the file containing Khmer sentence(s). In case, we provide a file name to the algorithm, it will read all the content of the file and then passes the content to the algorithm. Algorithm will return an segmented sentence corresponding.

Pre-processor

This step checks for the delimitors which can be others special Khmer characters or non-Khmer characters and which will be used to cut a big sentence into many small sentences.

Generate all possible segmented sentences

This step take a sentence which considered having no delimitors and try to generate all possible semgented sentences based on the dictionary. In our algorithm, we try to optimize the process here in order to speed up our whole segmentation process.

Select the best segmented by checking the character seperated

Here, a first possible segmented sentence is selected and then it will be used to combine all seperated Khmer characters together.

Post-processor

This part consists of applying all neccessary process in order to generate a final segmentation sentence response to our real need. For example, in our case, we try to replace all <space character> with <,> because the character represent a "pause" in our application. However, if we leave much post in our application or leave much post betweens less than 5 word might cause the application unacceptable. Thus, in our post processor, we try to eliminate some of them.

Figure 1: Steps representing our global algorithm

Dictionary

Our algorithm is dictionary-based type algorithm. And we have define a process which will load all the dictionary into the memory before the word segmentation process starts. In general, size of dictionary is considered to be large so we try to use the data structure as normal as possible. Here, we use **Array of String** and we have been using **Binary Search** algorithm to find a word in dictionary. Regarding to the dictionary format, we use the same format as lexicon like showing below. The information neccessary in this format is Khmer word, other information are not taken into account.

```
("ក" nil (((k A:) 1)))
("កក" nil (((k A: ?) 1)))
("ककि" nil (((k A:) 0) ((k A:) 1)))
("ककाक" nil (((k A:) 0) ((k a: t) 1)))
```

IV.2. Implementation & Result

We have implemented our word segmentation in C++ because it will be package with Festival in our final product. We have created a class called WordSegmentation which contains all the methods necessary for the segmentation process as shown in the **Figure 2**. The dictionary used in our test containing 18670 words. During our test, we used many file containing Khmer sentences. As result, the application provides words which are really well segmented and words which are not well segmented as presented in **Figure 3**. This is because of a Khmer word can be composed by many other Khmer words.In another hand, one can see in **Figure 4**, the result shows that there are fewer words that cannot be readable. The problem of this type happened because we do not define all cases in step of generating best segment from all possible segments yet.

During our test, we define the clock time to evaluate the speed need for the segmentation. In a file of Khmer text whose size is about 46 KB, the time needs for our segmentation process is 2s and for the text of 100KB, the segmentation needs only 5s to finish.

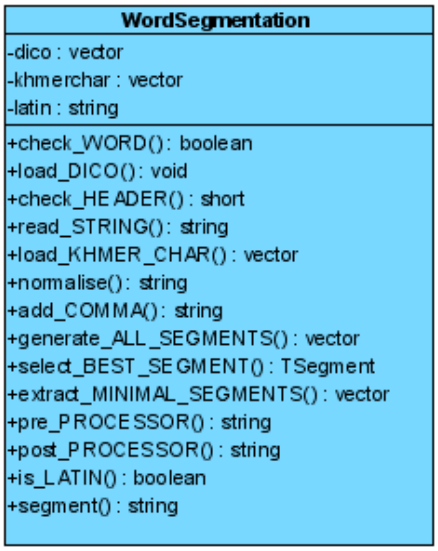


Figure 1 : Class diagram of WordSegmentation

Properties

- **dico**: stores all the words within our dictionary.
- **khmerchar**: stores all the Khmer characters' code.
- **latin**: a string containing a list of all characters considered as LATIN character.

Methods

- **check_HEADER**: it checks starting character of UNICOE character to see whether it is 3-byte character, 2-byte character or 1-byte character. With this method, we can easily define method for reading character from a given string.
- **check_WORD**: checks whether a word exists in dictionary or not.
- **load_DICO**: it loads all words within the dictionary to **dico.**
- **read_STRING**: it is used to read all the contents into a string.
- **load_KHMER_CHAR**: it loads all Khmer characters, converts each of them into corresponding code, and stores them in **khmerchar.**
- **normalise**: normally the content of UNICDE file in Linux and in Windows have a bit difference. In Windows, the file contains header character whereas it is not available in Linux. This method will delete the header character of the string loaded in Windows.
- **add_COMMA**: it is used to find all defined characters and replace them by (,) character.
- **generate_ALL_SEGMENTS**: this method generates all possible segments from a given string.
- **select_BEST_SEGMENT**: this method selects the best segment and applies the predefined methods on the best segment.
- **extract_MINIMAL_SEGMENTS**: selects all the segments containing minimum number of words.
- **pre_PROCESSOR**: this method does the same thing as the **pre-processor** presented in **Figure 1.**
- **post_PROCESSOR**: this method does the same thing as the **post-processor** presented in **Figure 1.**
- **is_LATIN**: it checks whether a character is a LATIN character or not.
- **segment**: it gets a string and segments it.

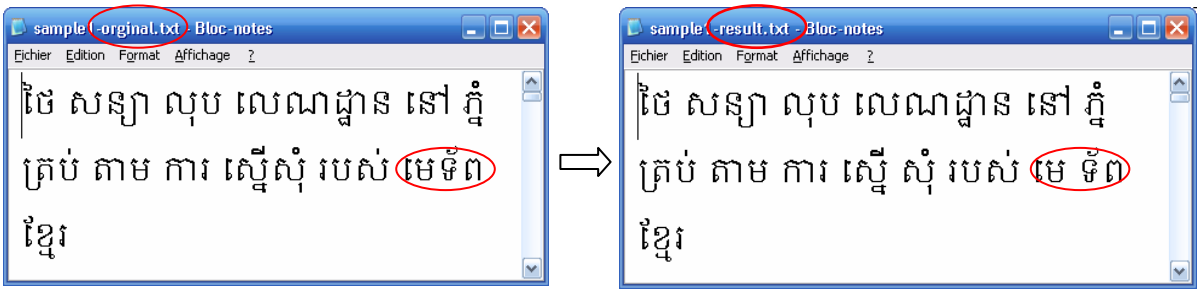


Figure 3: Shows that after segmentation process, a word can be segmented into many

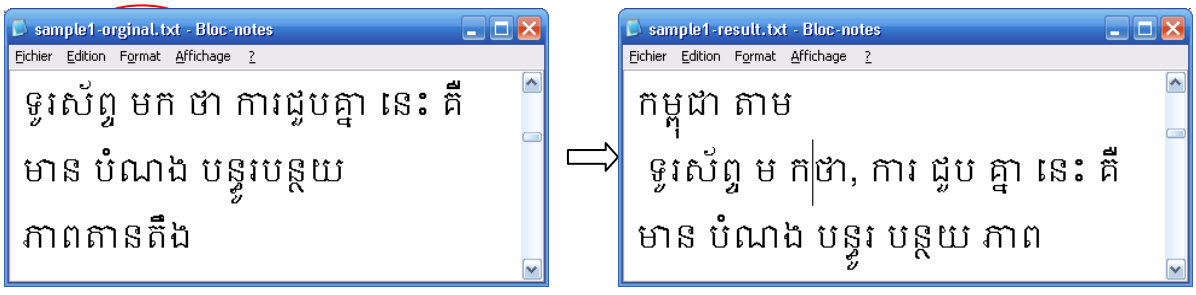


Figure 4: Shows that word segmentation can generate nonsense word.

IV.3. Remark

We presented our new algorithm for Khmer word segmentation which bases on MMA. Moreover, we have defined a solution to settle with words which do not exist in dictionary.
To validate our algorithm, we have done our test by implement a small testing application in C++. As the result, we found that our algorithm provides a good result which corresponds to 97 percents and it runs a lot faster. Thought, we found some problems during our test is 1) words which composed by many other words are separated, and 2) algorithm generates nonsense word because we have not defined all cases to apply for the selection of the best segment which requires a lot of notices. To conclude, we think that our algorithm can be used for our Khmer TTS application because it provides mostly correct segmentation. Even if word doesn't exist in the dictionary, our algorithm still provides a good segmentation. In order to improve our algorithm, we have considered to continued defines new cases applying on all possible in selection best segment provided in order to select the best one. With this application, we will be able to settle the two problems stated above.

V. OUTCOME OF THE TTS PROJECT

This is a research and development of Khmer Text-to-Speech application that will enable us to convert the Khmer scripts into sound. This project involved by many background included computer specialized and linguist. After the two years of the thorough works. We arrive at the completion of the project with the remarkable result. However, this application can not work perfectly; it is an acceptable application that can be used in different platforms such as window or Linux. More over our group have developed a plug-in that allows user to install and be able to use it in an internet-browser in the purpose of converting all Khmer script in webpage or email.

The sound is generated by our application is like the robot, but understandable, caused by the absence of the Khmer Intonation module that we are not able to implement it on time. But this is not a great obstacle that prevents user from understanding the converted-sound.

In general, we can say that we have completed the project 95% because of the absence of Khmer intonation module that prevents us to reach 100%. The accuracy of this application is 98%. The missing of 2% is because of the insufficient/missing words in lexicon or news words that present in the script, which are not included in the lexicon. One more problem is the diphone database as we still have some minor error while performing diphone slicing.

VI.CONCLUSION

This project makes a lot of contribution to Cambodian people. It makes information more accessible to communities, especially blind and illiterate people.
In term of research, it is the important force that helps us start to do research on Khmer Natural Language Processing. The most appreciated thing is that this project creates a very good network

among different teams in different countries. Thanks to other teams, we always got answers to problems we have encountered.
In academic context, this project also contributes to the improvement of our computer science department as it gives us more knowledge on natural language processing domain.

Acknowledgement

This work was made possible through the PAN Localization Project, (http://www.PANL10n.net) a grant from the International Development Research Center (IDRC), Ottawa, Canada, administered through the Center for Research in Urdu Language Processing, National University of Computer and Emerging Sciences, Pakistan.

References

1)Building new voice
2)Sinhala TTS System
3)Williams, B., Jones, R.J., Uemlianin, I.: Tools and Resources for Speech Synthesis Arising from a Welsh TTS Project. Fifth Language Resources and Evaluation Conference (LREC), Genoa, Italy (2006)
4) Microsoft Corporation.: Microsoft Speech SDK Version 5.1. Retrieved from: http://msdn2.microsoft.com/en-s/library/ms990097.aspx (n.d.)