CHULA TTS: A Modularized Text-To-Speech Framework

Natthawut Kertkeidkachorn
[1]Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University Bangkok, Thailand
[2]Department of Informatics
The Graduate University for Advanced Studies, Tokyo, Japan
Natthawut@nii.ac.jp

Supadaech Chanjaradwichai
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University Bangkok, Thailand
Supadaech.C@student.chula.ac.th

Proadpran Punyabukkana
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University Bangkok, Thailand
Proadpran.p@chula.ac.th

Atiwong Suchato
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University Bangkok, Thailand
Atiwong.s@chula.ac.th

Abstract

Spoken and written languages evolve constantly through their everyday usages. Combining with practical expectation for automatically generating synthetic speech suitable for various domains of context, such a reason makes Text-to-Speech (TTS) systems of living languages require characteristics that allow extensible handlers for new language phenomena or customized to the nature of the domains in which TTS systems are deployed. ChulaTTS was designed and implemented with a modularized concept. Its framework lets components of typical TTS systems work together and their combinations are customized using simple human-readable configurations. Under .NET development framework, new text processing and signal synthesis components can be built while existing components can simply be wrapped in .NET dynamic-link libraries exposing expected methods governed by a predefined programming interface. A case of ChulaTTS implementation and sample applications were also discussed in this paper.

1 Introduction

A Text-to-Speech (TTS) system is a system which artificially produces human speech by converting a target text into its corresponding acoustic signal. TTS systems are crucial components to many kinds of computer applications, particularly applications in assistive technology, E.g. applications for assisting the visually-impaired to access information on the Internet (Chirathivat et al. 2007), applications for automatically producing digital talking books (DTB) (Punyabukkana et al. 2012), and etc.,

Over the past decades, several TTS systems had been developed to fulfill applications on various computing platforms including mobile devices (Chinathimatmongkhon et al. 2008). Given specific domains, some applications of TTS systems require the systems to produce word pronunciations or generating speech signals that sound more natural to the listeners than ones generated with systems designed for texts of more general domains. For example, an application to read text from a social media web site might need a TTS system that performs a normalization of wordplays rather than attempting to pronounce them straightforwardly according to their exact spellings. While such a TTS system produced more

28th Pacific Asia Conference on Language, Information and Computation pages 414–421

naturally-sounded speech utterances (Hirankan et al. 2014), the normalization process might degrade a TTS's performance on a domain involving more formal texts where wordplays are scarce. For a TTS system aiming for expressive speech utterances, with multiple handlers, each of which is responsible for handling a different expression, the system could produce better results as well as easier handler development. A TTS system that allows interoperation of components, such as Grapheme-To-Phoneme (G2P) or signal generation components, deploying different speech and text processing algorithms without re-compiling of the system is obviously desirable. Still, many TTS systems were not designed with such abilities.

In this paper, we therefore reported our recent attempt on designing and implementing a modularized TTS framework, namely ChulaTTS. The goal of the design of ChulaTTS was to allow a TTS system to incorporate multiple speech and text processing components and allow them to work together with minimal development efforts. Components with similar classes of functionality must interoperate despite the differences in their underlying algorithms or the differences in phonetic units primitive to each of the components. With that goal in mind, ChulaTTS is suitable for conducting speech synthesis experiments to observe the performance of newly-developed algorithms in a complete TTS system conveniently. Furthermore, ChulaTTS can be easily configured into a TTS system expected to handle special phenomena appearing in the domain that it is deployed.

The rest of the paper was organized as follows. Related works were reviewed and discussed in the Section 2. In Section 3, we reported the design of our modularized TTS framework, and described the details of an implementation of a TTS system based on the modularized framework in Section 4. Section 5 discussed real applications of ChulaTTS systems. Finally, we concluded the paper in the last section.

2 Literature Review

In order to allow a TTS system to incorporate extensible handlers, several TTS frameworks (Orhan et al. 2008; Malcangi and Grew 2009; Wua et al. 2009) had been introduced. Orhan (2008) presented the Turkish syllable-based concatenation TTS framework. In their work, linguistic rules on Turkish were designed for handling exceptional cases such as special characters or symbols in Turkish. Although their framework installed the handler to provide a choice for applications, its choice was very limited to normal text and some special characters. Consequently, when a language had been evolved, the framework could not be extensible to support that evolution. Malcangi (2009) therefore introduced the rule-based TTS framework for mixed-languages, which allowed linguists to define multiple rule-based handlers to cope with various kinds of text. Even though their framework could be extensible to support the evolution of languages by simply adding a new rule-based handler, the new handler might cause ambiguity in the selecting handler process, in which an input text might follow conditions of many handlers, especially when handlers were become more and more. For this reason, the framework was not flexible to directly install new handlers, since we might have to modify the existing handlers in order to avoid ambiguity among handlers. Later, Wua (2009) proposed a unified framework for a multilingual TTS system. Their framework was designed to support extensible handlers of a TTS system by using a speech synthesis markup language (SSML) specification in which the mark-up tag provided a name of a particular method which should process the value in the mark-up. Unlike Malcangi's framework, the SSML markup clearly identified a handler which had to operate in order to avoid unclear situation in the handler selection. By following the SSML specification the framework could properly allow extensible handlers without causing any trouble to existing handlers. Still, some parts of their framework did not allow extensible handlers such as their waveform production.

Considering many related works above, we found that the aim of TTS frameworks was to enable ability to install extensible handlers. Still, there were many limitations to incorporate and extend new handlers in such frameworks. Our recent attempt therefore was to design and implement the modularized TTS framework, which supported extensible handlers in any stages of TTS systems without troubling other existing handlers.

3 The Modularized Framework

Typically, TTS systems have a common architecture similar to the illustration shown in Figure 1. This architecture consisted of two parts: the text analysis part and the speech synthesis part. An input text is fed into a text analysis block to generate its sequence of phonetic representation comprising phoneme and prosody annotation and then the sequence is passed to the speech synthesis block in order to generate real signal associated with the sequence of phonetic representation. Algorithms implemented in each processing step usually vary from system to system. According to the architecture in Figure 1, there are components whose underlying algorithms could be varied or allowing options in applying different algorithms to different portions of the text input. These components involve how the input texts are processed in order to obtain both underlying phonetic sequences and their suprasegmental information such as prosodic information governing how each phonetic unit in the sequence should be uttered and how speech signal should be generated. Typically, algorithms used for each component in a TTS system are predetermined and developed as an entire system.

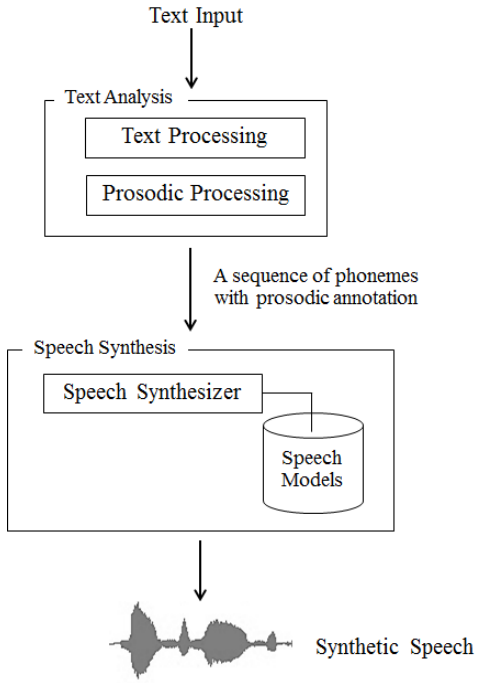

Figure 1. An architecture of a typical TTS system

Contrary to the architecture of a typical TTS system, we proposed a modularized TTS framework called ChulaTTS in which implementation of different text and speech signal processing are considered modules that can interoperate with one another. The aim of the framework is to provide flexibility in experimenting with different algorithms that could affect only a part of the whole system as well as to enable interoperability of multiple modules responsible for similar tasks of the TTS process. The latter makes a TTS system extensible when a new module is introduced and incorporated among existing ones in the system. Programming-wise, neither shuffling modules of a system nor adding additional modules to the system requires re-compiling of the source code of any modules already deployed in the system. To build a functional TTS system with the ChulaTTS framework, ones implement the TTS system by exposing components involving in the TTS process in the forms of modules consistent with the framework's specification and configuring the framework to utilize them.

Before elaborating on the classes of module in ChulaTTS, let's consider the typical architecture in Figure 1. Based on the architecture, if multiple processors were to simply process the input texts in parallel, there would be situations when ambiguities arisen from the different processors produced inconsistent results in some parts of the input. Some decision making components could be introduced to handle such inconsistent parts. In the ChulaTTS framework, we adopted multiple (or single) segment taggers that independently tagged each segment of the input with different algorithms as well as different sets of tags. A tag selector was deployed to determine how all the tagged segments be processed later on in the TTS process. With the mentioned segment tagging part, the overall architecture of the ChulaTTS framework is shown in Figure 2. The architecture is divided into three stages: 1) Segment tagging, 2) Text analyzer, and 3) Speech synthesizer. The details of the tasks to be performed in each of the three stages, classes of modules and their contractual (programming) interfaces, software implementation requirements, and how the resulting TTS system is configured are elaborated in Section 3.1 to Section 3.5.

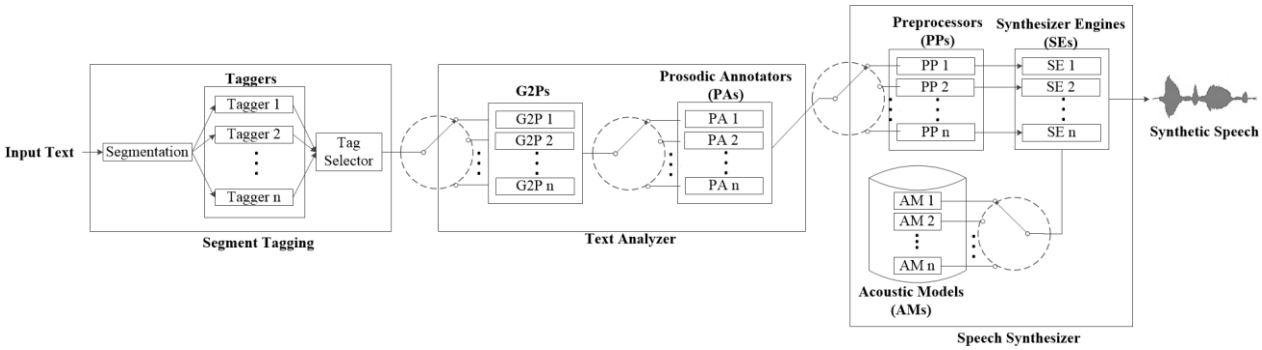

Figure 2. The Modularized Text-To-Speech Framework

3.1 Segment Tagging Stage

Segment Tagging in ChulaTTS is dedicated to segmenting an input text into smaller pieces of text, each of which with a proposed tag. Segment tags identify which modules process the tagged segments in later stages of the TTS process. Three steps are performed in this segment tagging stage: 1) Segmentation step, 2) Segment tagging step, and 3) Tag selector step.

Segmentation: The segmentation step inserts word or phrase boundaries into the input text string. Portions of texts located between adjacent boundaries are called "segments", each of which will then be marked with a tag in the next step. In an implementation of the ChulaTTS framework, one segmentation module can be selected via the corresponding configuration. All modules performing as a segmentation module must provide at least one segmentation function that receives the input text in the form of a string of characters and returns its corresponding sequence of segments.

Segment tagging: The segment tagging assigns an appropriate tag to each segment. Modules performing this step can have their own set of tags and conduct the tagging independently from other modules. An implementation without alternative algorithms for steps of the TTS process needs only a single tagger. Figure 3 depicts a conceptual example of the need for the later steps of the TTS process to heterogeneously handle different parts of input text motivates the inclusion of segment tagging. In the figure, segment tags can be used to process and synthesize speech with different personalities or expressions.

	Professor McGonagall gasped. "Lily and James... I can't believe it... I didn't want to believe it... Oh, Albus..."	
Segment	Professor McGonagall gasped.	"Lily and James... I can't believe it... I didn't want to believe it... Oh, Albus..."
G2P Tag	Narrator	Conversation
Prosodic Tag	Normal	Emotion
Synthesizer Tag	Narrator	Emotion
Model Tag	Narrator	Female

	Dumbledore reached out and patted her on the shoulder. "I know... I know..." he said heavily.		
Segment	Dumbledore reached out and patted her on the shoulder.	"I know... I know..."	he said heavily.
G2P Tag	Narrator	Conversation	Narrator
Prosodic Tag	Normal	Normal	Normal
Synthesizer Tag	Narrator	Normal	Narrator
Model Tag	Narrator	Old Male	Narrator

Figure 3. Conceptual examples of tags for the later stages[1]

All modules performing as a segment tagging module must provide at least one tagging function that receives a sequence of segments and provides a single tag for each of the input segment.

Tag selector: In cases of conflicting segment tags due to multiple segment tagging modules, this step decides on which of the conflicting tags should be kept and used as parameters in selecting modules in the later steps of the TTS process. A single tag selector module capable of handling all tags produced by all active segment tagging modules is required in a ChulaTTS implementation. The tag selector modules provide at least one function returning a sequence of tagged segments.

3.2 Text Analyzer Stage

The text analyzer stage is for producing a sequence of phonetic units with prosodic parameters. It consists of two steps: 1) G2P conversion, and 2) Prosodic annotation. The first step produces

[1] The example text from Harry Potter and the Sorcerer's Stone

phonetic units from the input sequence of segments. One or more G2P conversion module can be deployed in a single ChulaTTS implementation providing that they cover all possible tags in the implementation. Each segment tag must be associated with a G2P module while each G2P module can handle multiple segment tags. Segments are fed to G2P modules according to the implementation configuration. For a segment, the G2P module responsible for the segment produces a sequence of corresponding phonetic units, each of which can be declared by the module itself. Different phonetic units must use unique symbols. Phonetic units with similar symbols are considered the same type of units regardless of which modules handle the G2P conversion.

Prosodic annotator modules are deployed in the prosodic annotation step. Different modules are activated based on the segment tag according to the configuration of the implementation. Similarly to the phoneme units, prosodic markers produced by the modules must be supported in the Speech Synthesizer stage of the implementation.

3.3 Speech Synthesizer Stage

The role of this stage is to generate synthetic speech signals based on the phonetic representation and the prosodic parameters provided by the Text Analyzer stage. This stage involves three configurable parts: 1) Pre-processing, 2) Synthesizer Engine, and 3) Acoustic Models. A pair of Synthesizer Engine module and its corresponding Pre-processing module, responsible for adjusting the format of the phonetic representation and prosodic parameters so that they are consistent with the input interface of the Synthesizer Engine, must be configured to handle all segments tagged with a segment tag, while Acoustic Models can also be selected by the configuration, providing that their phonetic units and file formats are supported by the associated Synthesizer Engine module. All modules performing as a Synthesizer Engine module must provide at least one signal synthesis function that generates a waveform file that will be treated as the final synthesized speech by the ChulaTTS framework.

3.4 Module Development

An option that we chose in order to maximize interoperability of modules and, at the same time, avoid steep learning curves for researchers who wish to evaluate algorithms in ChulaTTS is to adhere to the .NET development framework on Windows platform for module development. The framework was written in C# and all classes of modules (described in Section 3.1 to Section 3.3) to be integrated to an implementation of the framework are expected to be in the form of .NET Dynamic-Link Library (DLL) exposing functions whose signatures are consistent with the contractual interface defined by the framework according to their module classes. New modules can be developed using any .NET targeted programming languages while existing executables can be wrapped inside .NET

3.5 Implementation Configurations

Configuring the ChulaTTS implementation is performed by modifying three key configuration files: Segment Tagging configuration which determines how the framework should execute steps in the three stages listed in Section 3. Configuration files are all in plain text format read by the framework at run-time. In each configuration file, the name of the DLL file together with the name of the function residing in that DLL file associated with its corresponding step in the TTS process must be specified in a pre-defined format. The framework checks for the consistency of these functions with their corresponding contractual interface defined by the framework.

The next section reports an example case of the implementation of the ChulaTTS framework. The case showed a sample scenario in which a newly developed algorithm was evaluated via subjective tests in a complete TTS system using the ChulaTTS framework.

4 Implementation

4.1 System Implementation

We put ChulaTTS framework to the test by implementing a complete TTS system called ChulaTTS. ChulaTTS inherently employ .NET

framework and C#, where all handlers are implemented and compiled as DLL.

Segment Tagging Implementation: To identify segments in ChulaTTS, we consider all white spaces in input text and break them into segments. We use single Tagger handler that was implemented by using regular expression to determine the tags for each segment. The four available tags are (1) Thai, (2) English, (3) Number, and (4) Symbol. Table 1 shows example of segments and their corresponding tags. Because ChulaTTS only uses one tagger handler, naturally, there is no confusing tag. Thus, tag selector was not executed in this case.

Segment	Results of Tagging
สวัสดี[2]	<1>สวัสดี</1>
Hello	<2>Hello</2>
2014	<3>2014</3>
น่ารักจุงเบยยยย[3]55[4]	<1>น่ารักจุงเบยยยย</1> <3>55</3>
ขอบคุณ[5]:)	<1>ขอบคุณ</1> <4>:)</4>

Table 1. The example of segments and tags

Text Analyzer Implementation: Four G2P handlers; G2P1, G2P2, G2P3, and G2P4, corresponding to the four tags were developed for ChulaTTS. The G2P1 handler was responsible for parsing Thai text into phonemes. It employed TLEX (Haruechaiyasak and Kongyoung 2009) to extract Thai words from each segment. Then, the phonemes were generated by looking a Thai dictionary. In addition, because Thai is a tonal language, tone marker was also supplied for each and every word. G2P2 handler employed an English dictionary to produce phonemes. Moreover, with the situation of out-of-vocabulary, the resulting phonemes would be the spelling pronunciation. G2P3 handler was to convert numbers into the right pronunciation using Thai rule-based technique for numbers. Finally, G2P4 handler was used for converting symbols to pronunciation using dictionary-based method. In this implementation, prosodic annotator, namely tone parameter, were embedded in all four GSP handlers.

[2] 'Hello' in Thai
[3] 'So cute' in Thai
[4] Pronounced as 'haha' in Thai
[5] 'Thank you' in Thai

Speech Synthesizer Implementation: In Speech Synthesizer, an acoustical model was implemented. One male speaker spoke 600 utterance sentences randomly selected from the T-Sync speech corpus (Hansakunbuntheung et al. 2003), in order to construct a speech corpus for training the acoustical model. The recording process was conducted in the sound proof chamber with the sampling rate of 16,000 Hz. After the recording process, a transcriber manually added short pause marks into the transcriptions and force align phoneme and recorded audio. In the ChulaTTS-based system, HTS (PukiWiki 2013) was selected as the synthesizer engine handler, and use it to train our acoustical model. Furthermore, we also developed a preprocessor handler to transform the results from text analyzer block into the format compatible to that of the HTS engine.

4.2 System Testing

To learn about the performance of ChulaTTS, a subjective test was conducted, using five-scaled Mean Opinion Score (MOS) approach (Orhan and Görmez 2008; Zeki et al. 2010). Six participants were recruited in order to perceive a set of stimuli synthesized from randomly selected text from BEST corpus (Nectec 2009), in which each stimulus was randomly presented and played from the same handset. Each participant was asked to listen to 30 stimuli and score each utterance on a five-scale basis, excellent (5), good (4), fair (3), poor (2) and bad (1). The overall MOS was 3.64.

4.3 System Improvement

Since ChulaTTS framework provides the ability to add extensible handlers to cope with new tasks, we implemented a new handler to evaluate how users may opt to prefer the new system. We used the implementation of ChulaTTS system described above as baseline. Curious how social media played its role in TTS, we extended our baseline by implementing a Tagger handler which could tag wordplay following the algorithm reported by (Hirankan et al. 2014).We defined tag of wordplay as "5". An example of Tagging results between baseline system and the extended system were shown in Table 2. We also implemented a new G2P handler, G2P5, which corresponded to tag "5" to handle wordplay as the technique introduced by (Hirankan et al. 2014).

Systems	Results of Tagging
Baseline	<1>น่ารักจุงเบยยยย</1> <3>55</3>
Extended	<5>น่ารักจุงเบยยยย</5> <3>55</3>

Table 2. The example of tagging chunks of "น่ารักจุงเบยยยย55"

To understand the performances of both the baseline and the extended systems, another subjective test was conducted. Eight users were recruited to give the opinion on the stimuli produced from both systems. All stimuli were synthesized from randomly selected text on Facebook. Each user was asked to compare ten stimuli produced from the two systems. We use ten-scaled MOS and asked the users to rate the quality of the sound. Score of five signifies indifference between the two systems. Scores less than five means the user prefers sounds generated from the baseline system, the lower the number, the more confidence the user have with the baseline system. On the contrary, Scores greater than five shows that the users prefer the extended system, the higher the score, the more confidence. The score of comparing performances was at 7.19, which indicated higher preference of the extended system.

5 Applications

ChulaTTS system has been implemented in two applications: Chula DAISY (Punyabukkana et al. 2012), an audio book generation system; and Chula FungPloen (Limpanadusadee et al. 2012), a universal listening device. Since ChulaTTS employs .NET framework, applying it to applications built on .NET framework was a simple task, regardless of the difference in domains.

Since Chula DAISY aimed to handle Thai book contents, the domain of the application was generally Thai well-written text. Consequently, a standard Thai G2P handler and a standard Thai synthesizer engine handler were sufficient Punyabukkana et al. 2012). However, For Chula Fungploen, the domain of input text became more sophisticated because the task in Chula Fungploen largely dealt with text appeared on the internet. For this reason, only the standard Thai G2P, and the Thai synthesizer engine handler were insufficient. Without ChulaTTS framework, one would have to implement another TTS system to fit each task. However, with the nature of ChulaTTS framework, it allowed flexibility to enhance new handlers to support this task without the redesign of the system. In Chula Fungploen, there were needs to cope with non-Thai text, especially numbers, symbols and English texts. The number tagger handler, the symbol tagger handler, the English tagger handler, the number G2P handler, the symbol G2P handler, the English G2P handler and the English synthesizer engine handler were simply installed into the existing TTS system. By adding those new handlers, Chula TTS was able to support the task of Chula Fungploen as reported in (Limpanadusadee et al. 2012). This scenario clearly demonstrated the extensibility of Chula TTS framework, which implies time savings as well as extra efforts.

6 Conclusion

Conventional TTS development cycle can be improved with the proposed ChulaTTS framework, which provides extensibility and flexibility for implementing a TTS system in a modular fashion. ChulaTTS framework comprises three parts, Segment Tagging, Text Analyzer, and Speech Synthesizer. This paper describes not only the framework itself, but also the sample of a real-world implementation scenario that proved to be effective.

References

Jirasak Chirathivat, Jakkrapong. Nakdej, Proadpran Punyabukkana and Atiwong Suchato. 2007. Internet explorer smart toolbar for the blind, In Proceedings of i-CREATe 2007: 195-200.

Proadpran Punyabukkana, Surapol Vorapatratorn, Nat Lertwongkhanakool, Pawanrat Hirankan, Natthawut Kertkeidkachorn and Atiwong Suchato. 2012. ChulaDAISY: an automated DAISY audio book generation, In Proceedings of i-CREATe 2012.

Nipon Chinathimatmongkhon, Atiwong Suchato and Proadpran Punyabukkana. 2008. Implementing Thai text-to-speech synthesis for hand-held devices, In Proceedings of ECTI-CON 2008.

Pawanrat Hirankan, Atiwong Suchato and Proadpran Punyabukkana. 2014 Detection of wordplay generated by reproduction of letters in social media texts, In Proceedings of JCSSE 2014.

Zeynep Orhan and Zeliha Görmez, The framework of the Turkish syllable-based concatenative text-to-speech system with exceptional case handling. 2008. In WSEAS Transactions on Computers, 7(10):1525-1534.

Mario Malcangi and Philip Grew. 20009 "A framework for mixed-language text-to-speech synthesis, In Proceedings of CIMMACS 2009: 151-154.

Zhiyong Wua, Guangqi Caoa, Helen Menga and Lianhong Caib. 2009. A unified framework for multilingual text-to-speech synthesis with SSML specification as interface, In Tsinghua Science and Technology, 14(4): 623-630.

PukiWiki. HMM-based Speech Synthesis System (HTS) http://hts.sp.nitech.ac.jp/ 2013.

Choochart Haruechaiyasak and Sarawoot Kongyoung, 2009. TLex: Thai Lexeme Analyzer Based on the Conditional Random Fields, In Proceedings of 8th International Symposium on Natural Language Processing 2009.

Chatchawarn Hansakunbuntheung, Virongrong Tesprasit and Virach Sornlertlamvanich. 2003. Thai tagged speech corpus for speech synthesis, In processing of O-COCOSDA 2003: 97-104.

Mustafa Zeki, Othman O. Khalifa and A. W. Naji. 2010. Development of an Arabic text-to-speech system, In Proceedings of ICCCE 2010: 1-5

Nectec. 2009. BEST 2009 : Thai Word Segmentation Software Contest", http://thailang.nectec.or.th/best/

Worasa Limpanadusadee, Varayut Lerdkanlayanawat, Surada Lerkpatomsak, Proadpran Punyabukkana and Atiwong Suchato, 2012. Chula-FungPloen: assistive software for listening to online contents, In Proceedings of i-CREATe 2012.