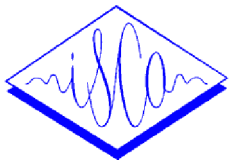

7th ISCA Workshop on
Speech Synthesis (SSW-7)
Kyoto, Japan
September 22-24, 2010

Speech Synthesis in the Mobile User Interface

Pieter E. Scholtz[12], *Justus C. Roux*[3], *Jacques P. du Toit*[2]

[1]Department of Electrical and Electronic Engineering, Stellenbosch University, South Africa
[2]CatchWord Language & Speech Technologies (Pty) Ltd, Stellenbosch, South Africa
[3]School of Languages, North-West University, Potchefstroom, South Africa

pieterscholtz@gmail.com, justus.roux@nwu.ac.za, jpdutoit@gmail.com

Abstract

This paper describes the development of a mobile application platform featuring an integrated text-to-speech synthesis engine as one of the core components. The work reported herein formed part of an international consortium project entitled *Mobile E-learning for Africa* (MELFA), which aimed to develop an application featuring reading and literacy training components, in English and one African language, isiXhosa. Particular attention is paid to the design and development of the mobile application platform, the embedding of the speech generation component and the multimodal user interface. The primary aims of the proposed platform are to support the widest range of applications that could benefit from speech output and for the applications to reach the widest possible audience.

Index Terms: speech synthesis, embedded, mobile, user interface, e-learning

1. Introduction

The proliferation of internet connected mobile devices in countries like South Africa presents unique opportunities for expanding the reach of existing E-learning material and applications. Bringing on-demand access to multilingual text-to-speech on mobile devices can further transform these devices into even more effective learning and communication aids.

1.1. Speech in the Mobile User Interface

Considerable work has been done in integrating speech recognition technology into the mobile user interface. Many mobile devices provide the user with some means of speech input as applications like VoiceControl on the iPhone or Google Search by Voice on Android become more widespread. The utility of such applications are immediately obvious, presenting the user with a simple, hands-free method of operating the device and for retrieving information from the web.

The utility of speech output on the other hand is often marginalised to more obvious applications, being widely used on GPS navigation devices to provide turn-by-turn driving directions and screen readers. Screen readers are invaluable in providing blind persons the ability to navigate visual interfaces and consume text-based content, but they are ill-suited to multimodal applications requiring on-demand access to spoken output of user selected content. In this paper we present a novel method of invoking the speech generation component that is more natural and less obtrusive than what is typically found in an accessibility scenario.

Furthermore, it is relatively simple to integrate ASR into a visual user interface: typically a button is used to instruct the device to listen for speech input, and a text field is displayed to record the recognised text. Speech synthesis on the other hand presents subtle challenges for the user interface designer. We address these challenges directly and present novel solutions.

1.2. Mobile Devices

When referring to mobile devices we typically imply a modern cellphone, a portable media player, a tablet computer or a laptop, including Apple's range of iDevices running iOS, including iPhone, iPod Touch and iPad, as well as devices from other vendors running modern mobile operating systems, including devices based on Android, Symbian, webOS, Blackberry and Windows.

1.3. WebKit and the Mobile Web

As we are particularly interested in multimodal user interfaces particular care has been taken in choosing the platform on which to build the visual components of our interface.

There is no question that the iPhone has redefined what is possible on mobile devices. One often overlooked contribution made by Apple with the iPhone in particular, has been the development of the open-source WebKit rendering engine, which is the core engine driving most modern mobile browsers from almost all major vendors, including Apple, Google, Nokia, Palm and Research in Motion (RIM).

Besides providing an open implementation that complies with existing web standards, Apple is also using WebKit to drive the adoption of new standards, especially HTML5. With this continued effort, not only from Apple but all other contributors, WebKit is fast becoming a true application development platform that is genuinely capable of delivering rich, interactive content to the widest possible audience.

The open-source nature of WebKit and its almost universal adoption makes deployment of WebKit-based applications to platforms which do not include a native, re-usable WebKit component, like Microsoft's Windows Mobile platform, a very real possibility. For example, our Windows Mobile version of the platform is built on the excellent cross-platform Qt Toolkit's QtWebKit component.

2. The Speech Engine

The TTS Engine is the fundamental component of the proposed application development platform. The wide range of applications we envisaged the platform to be useful for, informed the formalisation of a shortlist of requirements for the TTS engine itself.

In short, the speech engine must

1. produce effortlessly intelligible speech,

2. run natively on the device,
3. synthesise speech faster than real-time,
4. provide real-time feedback to the user interface,
5. have a small footprint and
6. be adaptable to novel languages.

Based on our past experience with the HMM-based Text-to-speech Synthesis (HTS) we deemed it the ideal solution.

2.1. HMM-based Text-to-speech Synthesis

Our previous work on HTS [1, 2] have found it to be particularly adept at producing highly intelligible and natural speech from limited data and linguistic resources. As the HTS runtime engine, HTS_Engine, is written in ANSI C it is also fast and highly portable.

However, both the parameter generation and waveform synthesis routines rely heavily on floating-point operations. Until recently most mobile devices did not provide hardware floating-point units and these operations had to be emulated in software, which is very slow and inefficient. Fortunately, mobile hardware have become increasingly sophisticated and most modern devices now include the ability to perform floating-point operations in hardware. This, coupled with our investment in the much faster LSP vocoder have all but eliminated initial performance concerns. Our HTS-based voices are now capable of running several times faster than real-time on modern hardware.

Furthermore, HTS voices are widely known for their small footprint [3] and its adaptability to new languages [4]. Having access to the source code of the HTS_Engine also enabled us to enhance its functionality with a number of specialised extensions to fulfil the other requirements.

2.1.1. LSP Vocoder

HTS and its runtime engine ships with two basic vocoders. The default vocoder uses the mel log spectrum approximation technique (MLSA) during synthesis. Recently another vocoder was added using Line spectral pairs (LSP). The MLSA vocoder has clear advantages over the LSP variant in speaker adaptation applications as the mel-cepstrum parameters (MCP) are widely known and exploited for their speaker independent nature, making them ideally suited for speech recognition as well.

However, when single-speaker MLSA and LSP voices were compared in subjective listening tests the results were less clear cut, indicating a slight preference towards LSP voices, possibly due to the retention of more speaker specific information [5]. Quality issues aside, the LSP vocoder was chosen because it is significantly faster, somewhere between 4 to 6 times faster, than MLSA at the synthesis stage. LSP voices also tend to be slightly smaller as 18th order LSP feature vectors typically achieve quality comparable to 24th order MCP voices.

To further improve the quality of our LSP voices we implemented the post-filtering formant enhancement technique from [6].

2.2. Flite Integration

HTS_Engine contains no text-processing component and is typically used with Festival for this purpose [3]. The unsuitability of Festival for embedding on mobile devices was the primary driving force behind the development of Flite, or Festival Lite [7]. Flite was painstakingly designed for resource constrained environments. It is fast and like HTS_Engine is written in ANSI C, facilitating a high degree of portability.

Flite is a complete TTS system featuring text processing modules as well as various waveform generation backends. As we have already settled on HTS as our preferred backend, Flite was required for its text processing faculties.

There is an open-source project, Flite+HTS_Engine, that integrates these systems, but unfortunately we found it to be not quite sufficient for our purposes. One particularly problem we found was that the text-processing is actually slower than waveform generation using this engine.

Instead we have written our own integration code, which adopts the Flite API and uses HTS_Engine as a waveform generation module.

2.2.1. Custom Linguistic Feature Set

The typical linguistic feature set used for English voices contain about 53 unique features, including phonetic, linguistic and prosodic contexts [3].

As part of our effort to reduce complexity and to improve performance we devised a smaller, more language-independent feature set. As certain African languages, isiXhosa in particular, have a tendency to accentuate ante-penultimate syllables, we count all segments from the front upwards and from the back downwards. The efficacy of these bidirectional counting features for languages such as English has not been established, but it is quite effective for a language such as isiXhosa [1].

2.2.2. HTS Label Generation

The HTS label sequence is the primary input to the HTS waveform synthesis backend. We found the bottleneck in Flite+HTS_Engine's text-processing to be the conversion of the Flite utterance structure to the HTS label sequence. The method used generates an HTS label for each segment, with no attempt to cache suprasegmental features which remain constant across multiple segments. Furthermore, as these features are calculated on the fly using Flite's so-called feature functions, also implemented without any caching, the process executes a large amount of code repeatedly and redundantly. The bottleneck was so severe that despite it having no reliance on floating point operations, the entire HTS label generation procedure was found to be slower than waveform generation itself.

Instead the label generation routine was rewritten from the ground up using a more hierarchical approach, with nested iterations over the different levels (e.g. phrases, words, syllables, phonemes) of the heterogeneous relational graph (HRG) structure that is used to present an utterance in Flite, caching values which remain constant for multiple segments lower down the hierarchy.

2.3. Interfacing with Audio Hardware

As the TTS engine is required to run on the mobile device providing on-demand access to speech output, latency remains a problem even if the engine is capable of rendering speech several times faster than real-time. Almost all TTS engines are capable of outputting generated audio samples to file. However, if the user were to wait for the entire utterance to be generated before the playback can commence, it will result in large and variable latency times. For example, if the engine is capable of rendering speech at 4x real-time, the latency for an 8 second utterance is 2 seconds, and becomes even greater the longer the utterance.

Not only would such high and variable latency be detrimental to the user experience, but it would also nullify some in-

teresting opportunities for real-time feedback from the speech engine to the user interface such as required by synchronised highlighting, see section 3.5 for more details.

To alleviate latency problems, while opening the door to real-time speech events, the TTS engine must directly interface with the audio hardware on the device. All mobile operating systems with adequate development frameworks should provide APIs to interface with audio hardware at a relatively low level.

For basic playback a simple push style API is sufficient. However, to support synchronised speech events, it is also necessary to monitor the state of the audio driver to trigger events at certain locations in the generated waveform.

2.4. Interfacing with Browser Engine

Mobile platforms that include a WebKit-based browser would typically also include a re-usable WebKit component that can be embedded in an application. Almost all modern smartphone operating systems now include WebKit, with Microsoft's Windows Phone platform being a notable exception.

As almost all application logic is implemented using web technologies such as HTML, CSS and Javascript, we require bidirectional communication between the Javascript environment running the application and the native environment running the TTS engine. Fortunately this communication is relatively easy to achieve as Android and Qt provide simple methods to expose the interfaces of native objects to the Javascript environment. Unfortunately, iPhone OS does not provide such functionality, instead requiring the use URL commands, which are intercepted at the native side and delegated to the appropriate native functions. All platforms provide straightforward means of communicating from the native environment to the Javascript environment.

The Javascript-Native bridge is used to invoke the TTS directly from the user interface and for the TTS to communicate state information directly to the user interface, providing a seamless and consistent multimodal user experience across multiple devices.

3. User Interface Development

As mentioned earlier we chose the web as our application development platform and WebKit, in particular. In WebKit we have found a visual user interface platform that was not only future-proof, but already widely available on target devices, showing a positive growth trend that would enable us to reach the widest possible audience.

We actively avoided platforms that impose a vendor lock-in, but required something sophisticated enough to provide a simple, compelling and consistent user experience across a wide range of devices. It was also of utmost importance that we could integrate seamlessly with our speech interfaces.

3.1. Touch Input

Touch was chosen to be the primary input method to the user interface. Ever since Apple launched the iPhone in 2007 other device vendors have scrambled to flood the market with competing touchscreen devices. The success of the iPhone and its derivatives attest to the efficacy of touch as a natural and immersive input mechanism. Another advantage of touchscreen based devices is that they typically have large displays suitable for a wider array of content than previously available on mobile devices.

3.2. Multilingual User Interfaces

To facilitate language learning scenarios the user interface framework was developed from the ground up with multilingual content in mind. To achieve this we extended the basic capabilities of internationalisation and localisation frameworks by allowing multilingual content within single page views. Language attributes are used in the content to enable the application to render the text as speech in the correct language.

Though there is no technical constraint on the number of languages a specific application can use, bilingual applications tend to be more sensible, while providing opportunities to simplify the interface. As part of our deliverables for the MELFA project we developed a bilingual English-isiXhosa application for the building industry, *MELFA for Builders*. The basic idea was to present the user with different types of text based content, such as phrases, terminology, exercises and more general texts. Throughout the application the user can switch between the English and isiXhosa versions and have all text on screen read aloud.

3.3. Interface Components

Application development frameworks typically consists of a variety of purpose-built re-usable graphical components. As one of our requirements were the offline availability of all content, including speech, all user interface components must be rendered from supporting data directly in the browser, with no reliance on server-side rendering technologies like PHP.

To support a reasonable range of applications we have developed a small set of re-usable user interface components. All components are built on our multilingual framework and supports any arbitrary language, and are driven by our custom built Javascript templating engine, which is responsible for transforming data into HTML fragments which are rendered by the browser.

Some of our key re-usable user interface components are:

PhraseBook Container for a list of translated phrases. The translations are presented in an expanded view of the phrase.

TermBook Container for a list of terms. The definition of a term is presented in an expanded view of the term. Useful for reference guide applications.

QuestionBook Container for a list of questions, with simple Yes/No answers. The question can be read aloud in the other language, which is not currently displayed. Useful for communicating across languages.

ExerciseBook Container for multiple choice questions useful for evaluating the comprehension of supporting content. Useful for educational applications.

3.4. The Reading Gesture

One fundamental challenge in the UI design was how to provide on-demand reading of on-screen content. We wanted to provide the user with a means to read all the content available on the screen at once, but also provide finer grained control, should the user want to pick out specific words, phrases or sentences.

Multiple on-screen buttons are impractical and quickly clutter the user interface. By default text selection operates on the character level, making it almost impossible to achieve accurate text selection on small touchscreen devices.

To overcome these limitations we devised a specialised touch-based reading gesture that allows the user to invoke the

TTS for a variable sized text segment by simply swiping a finger from left-to-right (LTR) over the text. Visual cues are used in the UI to indicate a successful gesture. The content selected by the reading gesture can also expanded to include multiple nodes by sliding the finger up or down over those nodes once the reading gesture has been activated. The speech output commences immediately after the user lifts the finger from the screen.

Furthermore, to provide the user and the application developer with variable levels of reading granularity, we typically pre-process and markup the text. This allows applications the ability to chunk the text at variable levels, including word, phrase and sentence level. Each individual chunk can be read in isolation or as part of a whole.

The LTR reading gesture is very natural and makes immediate sense for languages which are written from left to right. A similar gesture is also used on printed text by children learning to read, by people with reading difficulties, and those practicing speed reading. The exact same principal can be applied for languages with a right-to-left script.

3.5. Synchronised Highlighting

To support the educational aims of our platform we also designed with speech events in mind. Speech events are triggered by the TTS engine when interesting things happen in the speech. These events can be the start or end of sentences, phrases, words and even syllables and phonemes. This information is recorded during HTS label generation step, converted to an event timeline after duration prediction and delivered to the application during the synchronised waveform generation and playback process.

The application typically use these events to provide synchronised highlighting of the text content as it is being read. Similar to the reading gesture selection, the highlighting can also happen at various levels, with the word level being most typical. Not only is such information useful for the user to track the progress of the speech, it is invaluable for a person learning a foreign language, as it presents a direct link between the sounds of the language and its orthography.

Refer to section 4.1 for a powerful application of this technology when the highlighting is taken down to the syllable level.

3.6. Dynamic Speech Output

TTS engines have many parameters that control the generated speech. Some of these parameters are typically used to adjust the speaking rate, to shift or scale the pitch or to control the loudness. As the TTS engine is embedded on the device with its interface exposed to the application we can provide the user with a limited set of controls to affect the speech output in desired ways.

4. Applications

As we have spent considerable time in this paper detailing our design and development of our application platform we find it fitting to conclude with short descriptions of applications we have already developed within our targetted domains.

4.1. Education

Besides the *MELFA for Builders* application, we have developed a small demo application to showcase even more advanced E-learning functionality, highlighting the potential of our technology in advancing literacy. The user is presented with a page containing a simple sentence in isiXhosa *(The children are playing in the garden)*, followed by its syllabic breakdown:

Abafana badlala egadini
A/ba/fa/na ba/dla/la e/ga/di/ni

If the user taps on the underlined word, it cycles to the next available options from a small pool of words:“egadini”, “esikolweni”, “entabeni” and “ekhaya”. The syllabified version is also updated.

The user uses the LTR reading gesture to read the full sentence which highlights each word as it is read. The user then reads the syllabified version. The exact same speech is rendered, except each syllabic unit is highlighted as it is read. The user also has the option to insert artificial pauses between each unit, allowing some time for each unit to be heard in isolation, creating a speak & spell effect. The speaking rate can also be adjusted on a continuous scale using a simple slider control.

4.2. Communication

Multingual phrasebooks with synthetic speech output are compelling applications in their own right. Not only can such applications be developed very easily using our platform, but they can serve a variety of users, from a tourist in a foreign country who wishes to communicate in a local language, to someone who wishes to learn a new language.

Moreover, we have developed a prototype application aimed at medical practitioners based on our QuestionBook component. It features a questionnaire in English and isiXhosa which the operator uses in his or her own language to question a patient in the other language. As answers are carefully restricted to Yes/No they can be communicated and recorded readily.

5. Conclusion

We have described the design and development of a mobile speech-enabled application development platform which is built on open-source web and speech technologies. We found the fusion of these two technologies to be more than capable of supporting compelling and useful applications in a variety of domains. We have made a case for the wider application of speech synthesis and presented new ways of integrating speech synthesis effectively and inobtrusively in touch-based user interfaces.

6. References

[1] J. C. Roux and A. S. Visagie, “Data-driven approach to rapid prototyping Xhosa speech synthesis,” *SSW6*, pp. 143–147, 2007.

[2] Pieter Scholtz, Albert Visagie, and Johan du Preez, “Statistical Speech Synthesis for the Blizzard Challenge 2008,” *Blizzard Challenge 2008 Workshop*, 2008.

[3] H. Zen, T. Nose, J. Yamagishi, S. Sako, T. Masuko, A.W. Black, and K. Tokuda, “The HMM-based Speech Synthesis System Version 2.0,” *Proc. of ISCA SSW6*, pp. 294–299, 2007.

[4] A.W. Black, H. Zen, and K. Tokuda, “Statistical Parametric Speech Synthesis,” *Proc. of ICASSP*, pp. pp.1229–1232, 2007.

[5] S.-J. Kim, J.-J. Kim, and M. Hahn, “Implementation and Evaluation of an HMM-Based Korean Speech Synthesis System,” *IEICE - Trans. Inf. Syst.*, vol. E89-D, no. 3, pp. 1116–1119, 2006.

[6] Zhen-Hua Ling, Yi-Jian Wu, Yu-Ping Wang, Long Qin, and Ren-Hua Wang, “USTC System for Blizzard Challenge 2006 an Improved HMM-based Speech Synthesis Method,” *Blizzard Challenge 2006 Workshop*, 2006.

[7] A. W. Black and K. A. Lenzo, “Flite: A Small Fast Run-Time Synthesis Engine,” in *4TH ISCA Tutorial and Research Workshop on Speech Synthesis, Perthshire*, 2001, pp. 20–4.