

Speect: a multilingual text-to-speech system

J.A.Louw

Human Language Technologies Research Group
Meraka Institute, Pretoria, South Africa

jalouw@csir.co.za

Abstract

This paper introduces a new multilingual text-to-speech system, which we call *Speect* (**S**peech synthesis with extensible architecture), aiming to address the shortcomings of using Festival as a research system and Flite as a deployment system in a multilingual development environment. *Speect* is implemented in C with a modular object oriented approach and a plugin architecture, aiming to separate the linguistic and acoustic dependencies from the run-time environment. A scripting language interface is provided for research and rapid development of new languages and voices. This paper discusses the motivation for a new text-to-speech system as well as the design architecture and implementation of the system. We also discuss what is still required in the development to make the new system a viable alternative to the Festival - Flite tool-chain.

1. Introduction

Text-to-speech (TTS) synthesis introduces a multitude of communication possibilities, which are especially important in developing countries for cheap and effective conveyance of information. Multilingual text-to-speech is especially important in countries with more than one official language as is the case in South Africa. Multilingual text-to-speech, as used in this paper, refers to *simple multilingual speech synthesis* [1] where language switching is usually accompanied by voice switching. There are many high-quality commercial text-to-speech systems available for the major spoken languages, but not so for languages with a small geographical distribution or a small number of speakers relative to the major languages. Development of these technologies is a daunting task, and in multilingual environments even more so.

Text-to-speech synthesis is the automated process of mapping a textual representation of an utterance into a sequence of numbers representing the samples of synthesized speech [2]. This conversion is achieved in two stages as depicted in figure 1.

- *Natural Language Processing (NLP)*: Converting the textual representation of an utterance into symbolic linguistic units.
- *Digital Signal Processing (DSP)*: Mapping the symbolic linguistic units into samples of synthesized speech.

The *Natural Language Processing* stage consists of the following major modules:

- Text *pre-processing* involves the transformation of the textual input into a format suitable for the phonetization module. The specifics of this task is dependent on the

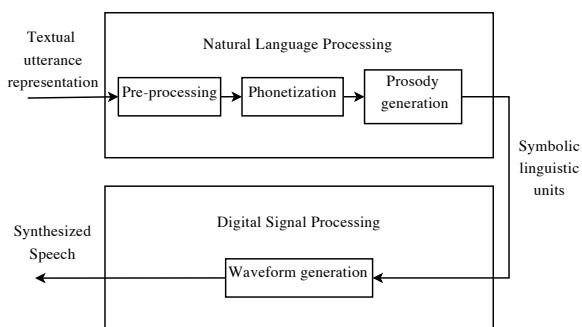


Figure 1: Functional blocks of a text-to-speech synthesizer.

type of textual input given to the system and includes utterance chunking and text normalization.

- The normalized text of the pre-processing module is converted into a phonetic representation by the *phonetization* block.
- *Prosody generation* involves the generation of intonation and duration targets through some form of prosody models.

The data generated by the NLP stage represents the *symbolic linguistic units*, which are then converted into synthetic speech by the *Digital Signal Processing* stage. The DSP stage can be realized by means of unit selection [3], statistical parametric synthesis [4], formant synthesis [5], or some other type of synthesizer technology. Each of the modules in the two stages adds some type of information to the initial given utterance which enables the final module, *waveform generation*, to generate synthetic speech based on this information.

The NLP stage is language dependent, whereas the DSP stage is dependent on the synthesizer technology of the implemented synthetic voice. Therefore, a multilingual text-to-speech system must be able to apply different NLP and DSP modules for different synthetic voices based on the language and synthesizer technology of the specific voice.

The next section discusses the motivation behind the need for a new speech synthesizer, followed by the design and implementation. We then conclude with a discussion.

2. Motivation

Over the last decade, the Festival speech synthesis system [6] has become the de facto standard free toolkit for speech synthesis research [7]. Festival provides a modular architecture whereby it is possible to modify each of the sub-tasks involved in the NLP and DSP stages in a text-to-speech conver-

sion process. Festival is implemented in two languages, C++ and Scheme (a lisp dialect), providing an integrated interpreted language for run-time manipulation. Festival, together with the Festvox project [8], aims to make the building of text-to-speech voices a structured and well defined task.

While being a fine example of a research system there are drawbacks to using Festival as a component within a speech enabled technology solution such as an *integrated voice response* (IVR). Festival has a large memory footprint and is relatively slow as a result of having a self contained interpreted language. A Festival compatible alternative is the Flite [9] synthesis engine, and while having a similar modular architecture and utterance structure representation, it provides improvements with regards to [9];

- speed,
- portability,
- maintenance,
- code size,
- data size, and
- thread safety.

Flite was written in ANSI C and has no interpreted language. In Festival a synthetic voice is loaded into internal data structures into memory, while in Flite all voice data is represented in C code. Therefore one still needs to use Festival and the Festvox toolkit for research and development of new voices, and then convert these voices with appropriate scripts into a Flite compatible version. The process of building a new voice in a new language (a language where the NLP modules do not exist in either Festival or Flite) will require one to first develop the NLP modules in C++ and/or Scheme in Festival and then rewrite these modules in C code for use in Flite. This is time consuming and requires expert knowledge of the Festival and Flite code base.

As a result of our experience with multilingual text-to-speech development we decided to design and implement a new text-to-speech system that combines the best features of the existing Festival and Flite synthesis engines while also addressing the shortcomings of these systems with regards to our requirements. The most important requirements for the new system, which we call *Speect* (Speech synthesis with extensible architecture), can be summarized as follows:

- **A single synthesis engine:** Having one synthesis engine reduces the code base and will eliminate any discrepancies between a development system and deployment system. This also leads to less maintenance.
- **Extensible architecture:** It should be easy to extend and modify the system with regards to the NLP as well as DSP stages of the text-to-speech conversion process.

3. Design

A synthetic voice in a TTS system can be seen as a combination of two parts

- **linguistic component:** providing language models and data for the NLP stage of the synthesis process.
- **acoustic component:** the acoustic models and data required by the DSP stage for waveform generation.

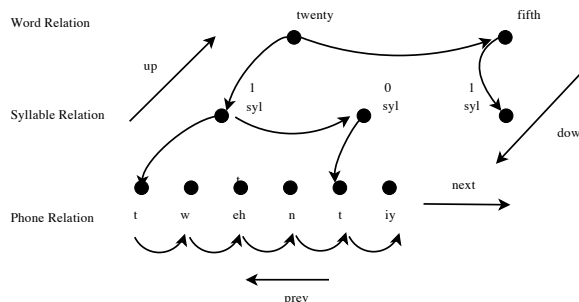


Figure 2: An example representation of an utterance structure using a heterogeneous relation graph.

The linguistic component is language dependent and can be shared by voices of the same language while the acoustic component is unique to a specific voice. Speect aims to provide control of the synthesis process and its design is intended to be independent of the underlying linguistic or acoustic models and data. Speect is not meant to replace speech processing tool-kits, the linguistic and acoustic models and data still needs to be generated by packages such as Edinburgh Speech Tools [10], Festvox and the Speech Signal Processing Toolkit [11].

To allow existing linguistic and acoustic Festival models and data to be reused, the internal representation of an utterance follows the same formalism as used in Festival and Flite. The *utterance structure* is represented internally as a *Heterogeneous Relation Graph* [12] (HRG), which consists of a set of relations, where each relation contains some items (the items need not be unique to a relation). The relations represent structures such as words, syllables, phonemes or even duration targets and the items are the content of these structures. Figure 2 shows an example representation of an utterance structure using a HRG with three relations and their items.

The individual NLP and DSP modules of figure 1 are called *utterance processors*. Utterance processors create relations in the utterance structure and add information (items with features) to the relations based on the linguistic and acoustics models and data. For example in figure 2 the *syllable relation* of the utterance has three items, with syllable stress as a feature of the items.

Speect has an object oriented design which allows the same modular approach to text-to-speech as Festival and Flite. A plugin architecture is used for the utterance processors, thereby restricting the language dependencies within the data and resources of the specific voice implementation and not in the synthesis platform. This plugin architecture allows different implementations of the same voice and/or language to be used during run-time, as the voice and language specifications load the required plugins.

4. Implementation

Speect is implemented in ANSI C to provide maximum portability and speed. The implementation of an object oriented paradigm in C requires more discipline from the programmer, but allows for code reuse and a modular design. Figure 3 shows the implementation architecture of Speect.

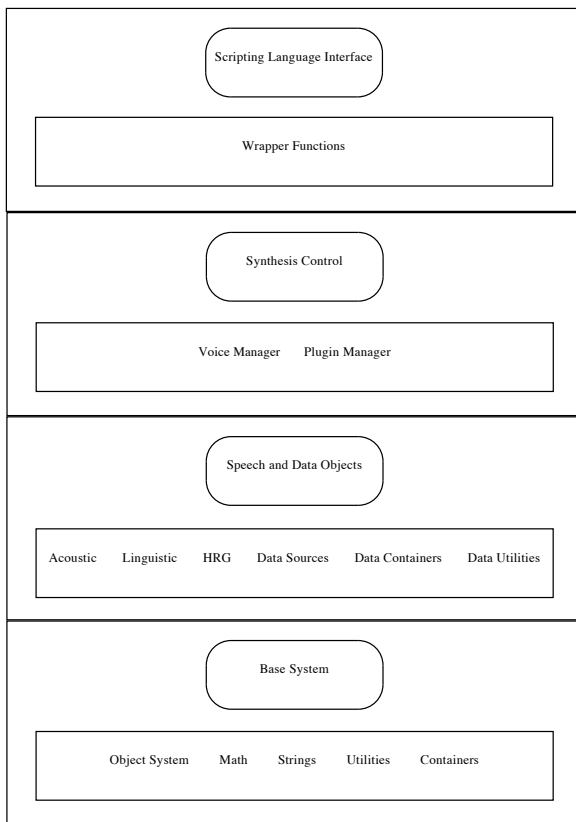


Figure 3: The Speect architecture.

The Speect architecture is divided into 4 major sections

- The **base system** provides a library of basic functions that are used by the upper levels of the system.
 - *object system*: the objects system implements a object oriented paradigm in C, whereby an object is described by two structures, one for it's data members and one for it's methods. The object system provides basic encapsulation, polymorphism, and inheritance.
 - *math routines*: basic mathematical routines.
 - *utility functions*: memory allocation and logging utilities.
 - *string functions*: basic string functions and UTF 8 support.
 - *basic containers*: doubly-linked lists and a hash table as basic data containers.
- **Speech and Data Objects** offer higher level objects specific to speech synthesis and data handling.
 - *acoustic objects*: provides *interfaces* to waveforms, data tracks, etc. Interfaces are implemented by plugins, therefore removing data dependencies from the synthesis system.

- *linguistic objects*: provides *interfaces* to phoneset, lexicon, etc. Plugins implement the linguistic interfaces.
- *HRG objects*: the utterance structure implementation. Follows the implementation of Festival and Flite for representing utterances.
- *data sources*: objects and interfaces for reading and writing data from/to files and memory. An *Extensible Binary Meta Language* [13] protocol is implemented as the standard format for reading/writing to files.
- *data containers*: Abstract objects that encapsulate the use of the base system containers.
- *data utilities*: the basic data object used in the HRG system. All objects that inherit from this object can be used as a feature in the utterance structure.

- **Synthesis Control** is provides the top level control of voices.
 - *plugin manager*: handles requests for specific plugin implementations. Dynamically loads and unloads plugins as required by the system.
 - *voice manager*: loads and unloads voices and handles synthesis requests.
- **Scripting language interface** connects interpreted scripting languages to the Speect library.
 - *wrapper functions*: the connection between the Speect library and scripting languages through SWIG (Simplified Wrapper and Interface Generator) [14].

The scripting language interface enables one to use Speect in an interpreted language setting, therefore speeding up research and development of new voices and languages. The speed of the Speect library is not influenced by the scripting language as it is external to the library implementation.

The work-flow of Speect is as follows: a synthesis request must be accompanied by the desired voice. The voice specification, which consists of a list of linguistic and acoustic utterance processors and associated data, is loaded by the *voice manager*. The desired utterance processor plugins are loaded dynamically by the *plugin manager* on request from the voice manager. The voice manager then proceeds to execute each of the utterance processors on the textual utterance representation, building an utterance structure. The utterance structure is synthesized and the synthetic speech returned.

5. Discussion

The Festival speech synthesis system provides a research and development platform for building synthetic voices in different languages. However, it is challenging to use in a real world deployment environment because of it's size and speed. Flite aims to correct these deficiencies with a much smaller and more efficient implementation, but lacks the development environment and suffers from language dependencies in the data and resources. Therefore, to develop synthetic voices for deployment one needs to create the voice in Festival and

convert it to a Flite suitable format. This is a complicated task, especially for new languages and requires extensive knowledge of the Festival and Flite code base.

Speect aims to be an alternative to the Festival - Flite tool-chain by providing a single speech synthesis engine for research, development and deployment in multilingual environments. This is achieved by a modular object oriented design with a plugin architecture, thereby separating the synthesis engine from the linguistic and acoustic dependencies. The improvements of the proposed Speect synthesis system with regards to the Festival - Flite tool-chain can be summarized as follows:

- The research, development and deployment cycle is done with one synthesis engine, reducing the size of the code base as well as the required maintenance. Therefore, implementation of new NLP or DSP plugins requires expert knowledge of just one synthesis engine.
- Run-time performance comparable with that of Flite, while retaining the research and development advantages of the Festival design, without the speed and size penalties associated with the integrated interpreted language because of the separation of the core library and the interpreted language.
- Footprint size comparable to Flite due to plugin architecture, therefore only the required modules for a particular voice are loaded.

The modular object oriented design combined with the SWIG interface enables the use of the Speect library through native calls from multiple scripting languages, and other languages such as Java, C#, Scheme and Ocaml, while encapsulating the underlying implementation through the use of the plugin architecture.

The Speect system has been completed up to a stage where utterance processor plugins can be loaded and run on basic input text and a concatenative unit selection method as described in [7], but to be a viable alternative to the current system the following still needs to be addressed:

- SWIG interface files for Python,
- Python scripts for the creation of unit selection voices,
- NLP modules for different languages,
- complete documentation on the implementation,
- manual for writing and extending plugins,
- documentation for building voices, and
- scripts for converting existing Festival voices into a Speect format.

6. References

- [1] Traber, C., Huber, K., et al. "From multilingual to polyglot speech synthesis", In Proceedings of Eurospeech, pp. 835-838, Budapest, Hungary, September, 1999.
- [2] Stylianou, Y., "Harmonic plus noise models for speech, combined with statistical methods, for speech and speaker modification", Ph.D. Thesis, Ecole Nationale Supérieure des Telecommunications, Paris, France, 1996.
- [3] Hunt, A. and Black, A. "Unit selection in a concatenative speech synthesis system using a large speech database", In Proceedings of ICASSP, vol 1, pp. 373-376, Atlanta, Georgia, 1996.
- [4] Black, A., Zen, H., and Tokuda, K. "Statistical Parametric Synthesis", Proceedings of ICASSP, pp. 1229-1232, Hawaii, 2007.
- [5] Högberg, J. "Data driven formant synthesis", In Proceedings of Eurospeech, pp. 565-568, Greece, 1997.
- [6] Taylor, P., Black, A.W., and Caley, R. "The architecture of the Festival Speech Synthesis System", 3rd ESCA Workshop on Speech Synthesis, pp. 147-151, Jenolan Caves, Australia, 1998.
- [7] Clark, R.A.J., Richmond, K., and King, S. "Multi-syn: Open-domain unit selection for the Festival speech synthesis system.", Speech Communication 49:317-330, 2007.
- [8] Black, A. and Lenzo, K. "Building Voices in the Festival Speech Synthesis System", <http://www.festvox.org/festvox/bsv.ps.gz>, 2003.
- [9] Black, A. and Lenzo, K. "Flite: a small fast run-time synthesis engine", 4th ISCA Speech Synthesis Workshop, pp. 157-162, Scotland, 2001.
- [10] The Centre for Speech Technology Research, The University of Edinburgh, The Edinburgh Speech Tools Library, http://www.cstr.ed.ac.uk/projects/speech_tools/.
- [11] Department of Computer Science, Nagoya Institute of Technology, Speech Signal Processing Toolkit, SPTK 3.1. Reference manual, <http://downloads.sourceforge.net/sptk/SPTKref-3.1.pdf>.
- [12] Taylor, P., Black, A.W., and Caley, R. "Heterogeneous relation graphs as a mechanism for representing linguistic information", Speech Communication 33:153-174, 2001.
- [13] Extensible Binary Meta Language, <http://ebml.sourceforge.net/>.
- [14] Beazley, D., "Swig: An easy to use tool for integrating scripting languages with c and c++", Presented at the 4th Tcl/Tk Workshop, Monterey, California, 1996.