FreeTTS
Open Source Speech Synthesis

Sun Microsystems Laboratories

Speech Integration Group
William Walker, Philip Kwok, Paul Lamere

http://freetts.sourceforge.net

FreeTTS is an open source speech synthesis system developed entirely in the Java™ programming language. FreeTTS is based upon *Flite*, a synthesis engine written in the C programming language developed at Carnegie Mellon University.

The Java™ platform has a stigma of being a poor performer and has often been shunned as an environment for developing speech engines. To better understand this stigma, we developed a speech synthesis engine entirely in the Java programming language. We discovered that the Java platform is an excellent platform for a synthesis engine and can significantly out perform a similar native C implementation.

What is FreeTTS? FreeTTS is based upon *Flite*, a small runtime speech synthesis engine developed at Carnegie Mellon University (cmuflite.org). Flite is derived from the Festival Speech Synthesis System from the University of Edinburgh and the FestVox project from Carnegie Mellon University (festvox.org).

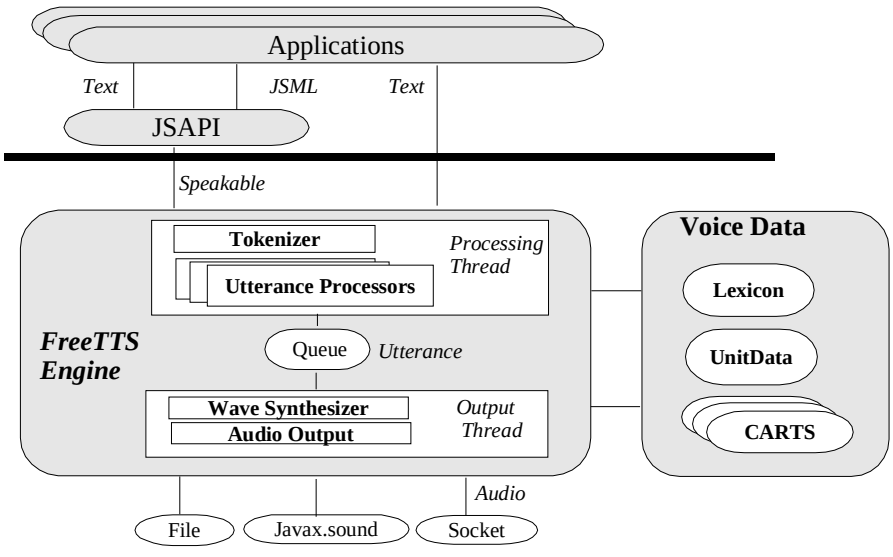
FreeTTS includes:

- A core synthesis engine
- Support for five general domain voices: an 8 kHz diphone voice, a 16 kHz diphone voice, and three MBROLA 16kHz diphone voices
- Supports a high-quality 16 kHz limited domain voice
- Partial support for JSAPI 1.0
- Extensive API documentation
- Several demonstration programs

The core synthesis engine of FreeTTS is quite flexible and easily allows new voices to be added to the system.

How does FreeTTS Work? There are a number of steps in the synthesis process. Many of these steps need to be customized depending on the locale and the type of synthesis employed. Speech researchers also need the ability to plug in new algorithms easily. FreeTTS provides a general framework for the synthesis process that allows the various steps in the process to be customized. Figure 1 shows the overall architecture of FreeTTS.

Figure 1 – FreeTTS Architecture



To support a new FreeTTS voice, a developer provides a set of *UtteranceProcessors* and associated data that define the processing for the voice. An UtteranceProcessor takes as input an *utterance*, performs some processing on the utterance, and typically adds some annotation or data to the utterance as a result of this processing. With this framework, the synthesis process becomes the following:

1. Break the input text into a series of utterances
2. For each utterance, apply each of the UtteranceProcessors
3. Output the generated audio data

A typical FreeTTS voice will define UtteranceProcessors that perform the following functions:

- **Text Normalization –** Converts the input text into a stream of words. For example, the text "Dr. Smith lives on 33 Garden Dr." would be converted to "doctor smith lives on thirty three garden drive." The text normalization process deals with a wide variety of cases including numbers, dates, times, titles, and place names.
- **Linguistic Analysis –** Attaches semantic information to the utterance. This can include phrasing and part-of-speech information.
- **Lexical Analysis** – Determines the pronunciation for each word of the utterance. Typically, a FreeTTS voice will use a lexicon to determine the pronunciation for a word. If a word is not in the lexicon, a set of sophisticated letter-to-sound rules are applied to generate a pronunciation.
- **Prosody Generation –** Attaches to the utterance information about pauses, pitch, duration, tone, stress, and amplitude. These processors will typically use classification and regression trees (CARTS) to generate this prosody information.
- **Speech Synthesis –** Generates audio data for the utterance. Typically a synthesis processor concatenates speech units based on diphones or other units of speech. Synthesis can be particularly CPU intensive since it involves a great number of floating point operations. Synthesis runs in a separate thread to reduce latency and to increase the possibility of parallelism on multi-CPU systems.

How does FreeTTS Perform?

Since one goal of the FreeTTS project was to investigate the performance characteristics of a speech synthesis engine on the Java platform, we instrumented FreeTTS to collect performance metrics. We also collected comparable performance data for Flite, the native C engine. The initial implementation of FreeTTS processed utterances slightly more slowly than Flite. We used profiling tools to identify performance bottlenecks and optimized FreeTTS based on these results. The optimized FreeTTS ran significantly faster than Flite as seen in the following table:

Task	*Flite*	*FreeTTS*
Process 8 hours of speech 1-CPU 296Mhz sparcv9 with 128MB memory	00:17:09	00:05:41
Process 8 hours of speech 2-CPU 360Mhz sparcv9 with 512MB memory	00:14:02	00:03:10
Time to first sample 5 word sentence 1-CPU 296 MHz sparcv9 with 128 MB	107 ms	19 ms

Table 1: Engine Performance Comparison

A number of features of the Java platform contributed to the high performance of FreeTTS:

- **Java HotSpot™ technology** - provides automatic optimization of the critical inner loops.
- **Collections –** provides high performance data structures such as lists and hash tables.
- **Threading –** allows for better utilization of CPUs on multi-CPU systems.

Other features including garbage collection and dynamic loading of code allow for a cleaner and more flexible system that can be extended easily.

Conclusions

Our implementation of a speech synthesis engine in the Java™ programming language demonstrates that the performance stigma is unfounded: FreeTTS significantly outperforms its native C counterpart. The resulting synthesizer is not only a high-performance engine, but also provides substantial flexibility and can easily be extended with new voices.

FreeTTS has been released to the open source community. The project is hosted on the open source web site SourceForge at:

FreeTTS.sourceforge.net

Acknowledgments We would like to thank Alan Black and Kevin Lenzo for creating Flite, the starting point for FreeTTS.