Distributed Text to Speech Synthesis for Embedded Systems – An analysis

Samuel Thomas, Hema A. Murthy and C. Chandra Sekhar
Department of Computer Science and Engineering,
Indian Institute of Technology Madras, Chennai, India.
Email: {samuel@lantana.cs.iitm.ernet.in, hema@lantana.cs.iitm.ernet.in, chandra@cs.iitm.ernet.in}

Abstract

In this work we describe the design of a Text to Speech (TTS) synthesizer for embedded devices using Flite, a small footprint text to speech system. We outline two methods for implementation of a TTS on a low resource device. The first method focuses on porting the entire synthesizer onto an embedded device directly. The second method explores possibilities of using distributed speech synthesis when the available memory and computational resources are limited. The paper gives results of simulating distributed synthesis on a low bandwidth network with a small memory footprint executable.

1. Introduction

Owing to the proliferation of mobile devices, there is an increasing need for speech interfaces on low-end portable devices like PDAs, mobile phones, digital diaries, information points on cars and home appliances. Text to speech synthesis (TTS) is a vital component of the speech interfaces that allows low bandwidth text to be converted into speech on these devices. It also finds applications in situations where a visual interface for embedded devices is inappropriate, as in the case of devices to be used by the visually or physically challenged and the illiterate.

In embedded devices where the constraints are speed and size, it would be necessary to incorporate a text to speech system that has a small memory footprint and is fast. With text to speech synthesis improving in quality over the recent years, the size of these engines has grown considerably large that they cannot be used directly on embedded devices. One such large system is the Festival system [1], which is slow and consumes considerable amount of memory. Flite [2] is a small, fast, run-time synthesis library suitable for embedded applications and is designed as an alternative run-time synthesis platform for Festival. But Flite itself has a fairly large footprint, requiring about 10 megabytes of RAM.

The focus of this paper is to reduce computation and memory requirements still further by using a client-server approach to distribute tasks. Section II describes the Flite system we work on. In section III we talk on the computation and memory requirements of different phases in speech synthesis in the Flite System. The section analyzes the amounts of computation that must be performed at the client end, given that the client-server link is a WLL [6] based low bandwidth IP network. We also discuss an implementation to achieve continuous synthesized speech using distributed speech synthesis. Finally in section IV we bring out the performance of the system designed using this approach. In section V we present the conclusion.

2. The Flite System

The Flite system consists of four modules -

a **core library module** : C objects that are used to build the Flite system.

a **language module** : language definitions in the form of sound units of the language (phones), tokenization rules.

a **voice module** : models for speaker-specific prosody and intonation.

a **language lexicon module** : letter to sound rules and pronunciation models for out of vocabulary words.

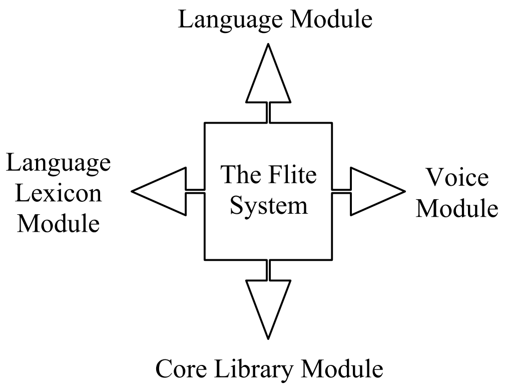

Fig. 1. Modules of the Flite system

We use a Flite system for American English with a size of six megabytes in this work.

3. Text to Speech Synthesizer on Embedded Systems

It has been observed that many embedded devices that require a text to speech interface already have a digital speech processor on them. Moreover, these machines come with real time operating systems that can manage the device. Applications written for these machines can hence take advantage of these resources, the main constraint being the available memory. We consider two approaches to the design of TTS for embedded devices. One model aims at devices with a memory of 10MB. The other model aims at devices with about 100KB of memory. While in the first model the entire synthesizer to can be ported onto the device, the second model requires distributed speech synthesis.

3.1 A Standalone Text to Speech Synthesizer for Embedded Systems

To investigate the first option, we ported the Flite synthesizer to a DSP processor - the ADSP-21535 Blackfin DSP processor from Analog Devices [5]. The porting was done using VisualDSP++ release 3.0 for the Blackfin DSP family from Analog Devices. With the programmable environment supporting component implementation in C/C++ or assembly language, the entire code was compiled after several unsupported constructs in the Flite code were removed. With Blackfin DSP processor supporting four gigabytes of memory as a single block, the memory requirement of six megabytes for the Flite system is not a problem. It is however evident that in applications where the processor would be at work, the amount of memory available for the TTS application would be much less. This approach is not a feasible option in those cases.

3.2 Network Distributed Speech Synthesis

With most embedded systems having memory capacities of only a few hundreds of kilobytes, an option to run applications with high memory requirements would be to make them network distributed provided adequate network bandwidth can be ensured. The first issue in implementing a distributed speech synthesizer for an embedded device is to identify phases based on memory and computational requirements, into which the whole process of synthesis can be split. Once such phases have been identified, they can be used to design client and server applications depending on the resources available on the host machines which are designated to run them.

In the context of embedded systems the client end of an application can consist of phases that have low computing requirements while the remaining phases can be part of the server end of the application. Table 1 shows the various stages of speech synthesis of the Flite system and their contribution to the total code size and network traffic. Figure 2 shows the possible different stages in distributed speech synthesis.

The network distributed speech synthesis module has been designed taking into consideration an embedded device like a mobile phone that is bound to use an embedded text to speech synthesis system. With the advent of powerful mobile phones, memory of about 20MB as heap size and 80MB of shared memory is available for programs to use [3]. These devices normally operate at an average clock speed of 100Mhz. However with several other real time operations and programs running on these devices concurrently, only a small portion of these resources are bound to be available for a TTS. It is therefore necessary to design a client that would require only a few hundred kilobytes of memory and can be run on a real time basis

Table 1. Contribution of different stages to the total code size and network traffic

Phase	Stage	Contribution to total inter-stage data for synthesis (%)	Contribution to total code size (%)and prominent contributor to size
I	Tokenization of text	7.3	1.1 Text Parsers
	Token analysis	2.7	
	Part of speech tagging	1.0	
	Building phrasing relationships	0.03	
II	Creating relationships between words	13.7	35.2 Language Lexicons
	Prediction of pauses, inserting silences		
	Applying Intonation		
	Application of post lexicon rules		
III	Duration Prediction of segments	5.1	63.6 Diphone Databases
	Creating the f0 contour	10.9	
	Generation of the waveform	59.7	
IV	Transfer of PCM values and playback.	Variable - depends on text to be synthesized	0.1 Network components

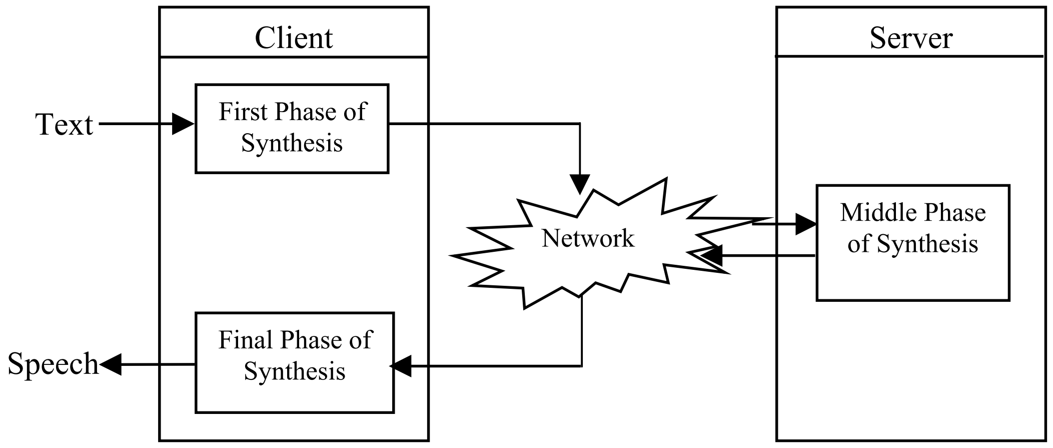

Fig. 2. Schematic diagram of distributed speech synthesis

on low clock speeds. These devices operate on networks like the GSM network (9.6 to 14.4 Kbps), High-Speed Circuit-Switched Data service (Up to 56 Kbps), GPRS (56 to 114 Kbps) and future 3G networks (384 Kbps to 2 Mbps) [4]. Depending on the type of network to which the device is connected and the network conditions at the time of use, variability in bandwidth speeds can be expected. Keeping such a scenario in mind we decided on a client design that could accommodate a maximum memory requirement of 100KB and was made up of low computation intensive tasks. An average bandwidth scenario is also considered by using a WLL based IP network with a low bandwidth of 35-70Kbps [6].

Table 1 shows that any design of a client and server is basically a trade off between code size permitted using available memory resources and network bandwidth available. The network client would essentially be required to take in text, get it synthesized and do a play back. It would accept text from users for synthesis, perform text tokenization on the input text, and send the intermediate form of synthesis to the server over network for further processing. The server then processes the tokens into basic units for synthesis, applies prosodic rules and creates the speech waveform for the text. The PCM values of the waveform are then sent back to the client for playback. By offloading the steps that take up most of the memory requirements from the client, the code size dwindles to 64KB, a size that is acceptable on many embedded systems. The server accounts for 5.5 MB.

3.3 Streaming Synthesis

Once the boundaries of the client and server have been fixed, it is necessary to decide on an effective network communication protocol to facilitate data transfers and handshakes. Distributed speech synthesis involves streaming of data instead of having indefinite delays for the entire text to be processed by the server and received by the client. The client also needs to be designed to run concurrent processes that receive speech data and at the same time play back buffered data as they get assembled.

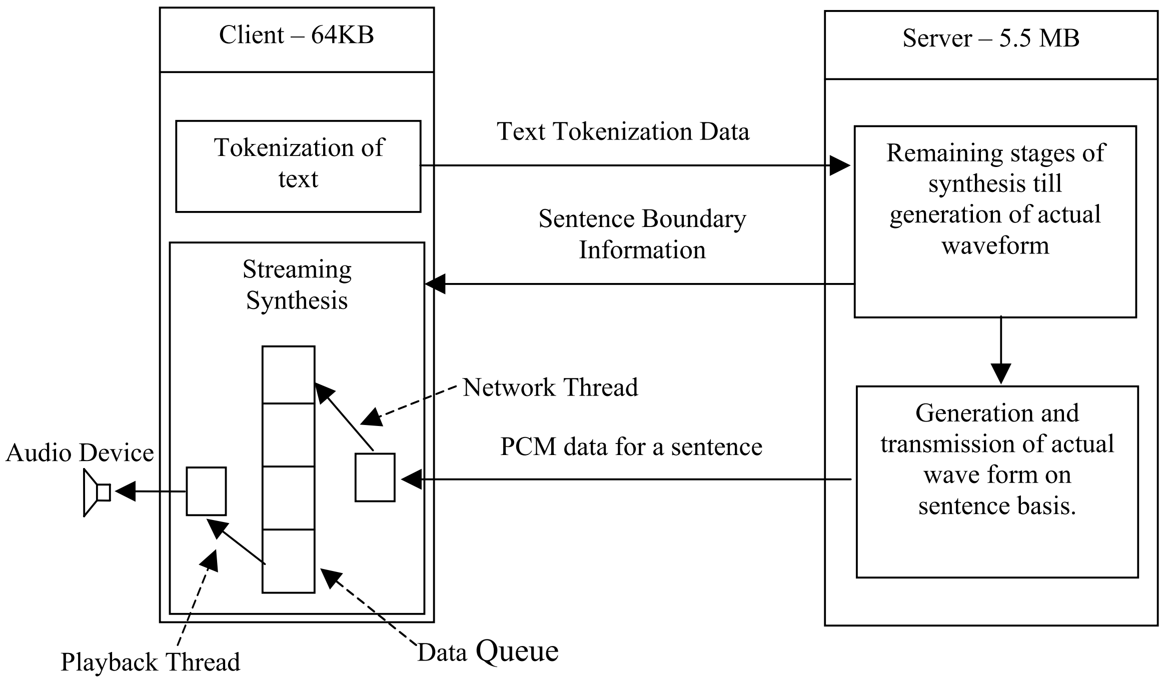

Fig. 3. Distributed text to speech synthesis as a client-server architecture

Semaphores are used to ensure concurrency and synchronization of the processes while using the common data structures. The client-server architecture is shown in Figure 4.

It is seen that streaming or buffering based on fixed sized buffers is not suitable as these buffers partition sentences not necessarily at word boundaries. Using fixed size buffers especially when the network is slow, creates long pauses in the middle of words at times, affecting the quality of speech and its perception. This is tackled by using variable size buffers instead of fixed size buffers. The variable size buffers are calculated by locating sentence boundaries and then estimating the necessary buffer sizes to hold them. The server determines these boundaries and sends them to the client prior to transmission of speech data. The information regarding sentence boundaries is used by the client to determine the size of the buffers to use for an entire sentence. Though this incurs an extra overhead of having to transmit information about sentence boundaries, it ensures continuos speech output. We implemented streaming synthesis using a queue data structure that buffers speech data even as speech play back is in progress. Optimum performance was achieved with a queue size of four and with queue elements being reused.

4. Evaluations

The two models for TTS are implemented and tested on high-end machines to ensure that functionality requirements are met. The distributed Speech synthesizer was tested on a WLL based IP network with a low bandwidth of 70Kbps to find the minimum network requirements for effective network operations.

Table 2. A comparison of time taken by distributed speech synthesis over a low bandwidth network and as a standalone system.

Number of characters in sentence	Time for synthesis as a standalone (in seconds)	Time for distributed synthesis using 70 Kbps PPP Link (in seconds)
10	0.65	5.22
20	1.40	6.89
30	2.09	9.48
40	2.57	11.92
50	2.98	13.82
60	3.21	16.87
70	4.19	21.03

A performance evaluation based on Mean Opinion Scores (MOS) on a five point scale with 20 listeners using two paragraphs of text gave an acceptability score of 75%.

The two paragraphs of text had a total number of 24 sentences with character lengths ranging from 10 to 70. The time taken for text to speech synthesis by a standalone system and the distributed synthesis system is given in Table 2.

It is seen that the time taken by the distributed synthesis system is about five times more than that of the standalone system. The mean optimum scores are low primarily because of the prominent gaps that appear when synthesizing long sentences. The acceptability ratio can be increased by using dynamically dividing large sentences at suitable points. Increasing the network bandwidth can also reduce the time required for synthesis.

5. Conclusions

We presented two methods by which a speech synthesizer can be implemented depending on the memory constraints of the system. Both the systems are now implemented on high-end machines and need to be implemented on embedded systems for which they are meant. Issues related to actual implementation need to be addressed. These results could also be lead to the development of an ASIC for text to speech systems for embedded systems.

References

[1] P. Taylor and A. Black and R. Caley, "The architecture of the Festival Speech Synthesis System", 3rd ESCA Workshop on Speech Synthesis, pp. 147-151, 1998, Jenolan Caves, Australia.

[2] A. Black and K. Lenzo, "Flite: a small fast run-time synthesis engine", ISCA 4th Speech Synthesis Workshop, pp 157-162, 2001, Scotland, UK.

[3] Nokia, Symbian Specifications Matrix, August 2004.

[4] Speed of Important end-user and Backbone Transmission technologies, http://whatis.techtarget.com/definition/0,,sid9_gci214198,00.html, August 2004.

[5] Analog Devices Inc., ADSP-BF535 Blackfin Hardware Reference, April 2003.

[6] Analog Devices Inc., USA, Indian Institute of Technology, Madras, India, Midas Communication Technologies (P) Ltd., Madras, India, corDECT Wireless Local Loop Technical Report, July 1997.