Design and Implementation of

Three-tier Distributed VoiceXML-based Speech System

BY

Thomas C.K. Cheng

Supervised by

Professor Michael R. Lyu

A report submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Computer Science and Engineering

The Chinese University of Hong Kong

AUTHORIZATION

SIGNATURE

APPROVED:

Prof. Michael R. Lyu, SUPERVISOR

Department of Computer Science and Engineering

May, 2002

ACKNOWLEDGEMENT

I would like to acknowledge my supervisor Professor Michael Lyu, who provided many valuable opinions and guidance for me throughout the project.

I would also like to thank my parents, brother, sister and my fiancée for their support.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

There are plenty of speech recognition research results and development applications targeting on embedded systems, desktops and servers. Each of them has its own limitation. We may have a command and control application on mobile devices, a speaker-dependent dictation application on home PCs and a speaker-independent interactive voice response (IVR) application on call-center servers. Advances in speech recognition technology, wireless technology and growth of the Internet lead us to establish a universal solution to break all the limitations.. In particular, we propose a new three-tier speech system which can make us access information more seamlessly and more naturally.

In this project, we design and implement a three-tier distributed VoiceXML-based speech system to demonstrate our model. The system is comprised of three components, namely terminal, speech browser and document server, forming the three-tier model. The terminal usually has small sizes with limited computing powers, small memories and limited bandwidth networks; for example, mobile phones, PDA with bluetooth, analog phone plus PC with telephone interface. Owing to the capabilities, it acts as a front end that uses data channel to send/receive the parameterized representation of speech to/from the speech browser. The speech browser is a cluster of powerful servers acting as a back-end that performs the core recognition, synthesis processes and the interpretation of VoiceXML files retrieved from the document server. The document server is a web site containing the VoiceXML files.

This model adds mobile devices the dialog-based speech capability to access the enormous multilingual information in the Internet without breaking the size, memory and computation limitations. By adding the features of load balancing and fault tolerance to the speech browser, a reliable back-end service can be guaranteed. Our goal is to find a complete solution to fit into this model by applying various technologies.

Chapter 1 INTRODUCTION

Nowadays, most mobile devices as well as computers utilize a graphical user interface (GUI), based on graphically represented interface objects and functions such as windows, icons, menus, and pointers. Most computer operating systems and applications also depend on a user's keyboard strokes, mouse clicks, button presses and pen actions. Speech, however, is the most natural and efficient way for human to communicate. Starting from 1970s, many researchers have built a theoretical framework on the speech recognition technology and until recently, advances in computer technology and speech algorithm brought the speech recognition technology to the practical uses in various areas.

Today, the speech recognition technology is quite mature and we can have a speaker independent large vocabulary continuous speech recognition system running on a powerful PC. In fact, the capability of speech recognition system depends on how much computational and memory resources are used. Mobile devices normally have a CPU running at the range from several to hundreds of million instructions per second (MIPS) and a RAM device of size hundreds of kilobytes to tens of megabytes. With this limited resources in mobile devices, only a simple command and control application or a speaker dependent small vocabulary speech recognition application can be achieved.

In order to break all the above limitations, a three-tier model is proposed to achieve a unified speech system in different environments. To demonstrate how this three-tier distributed VoiceXML-based speech system works, a credit card authorization application was developed by using JAVA. The outline of the report is as follows. Chapter 1 is the introduction. Chapter 2 is the speech technology. Chapter 3 is the system requirement. Chapter 4 is the system design. Chapter 5 is the implementation. Chapter 6 is the result and discussion. Chapter 7 is the conclusion.

Chapter 2 SPEECH TECHNOLOGY

2.1 *Speech Recognition*

After years of research and development, accuracy of automatic speech recognition remains one of the most important research challenges. A number of well-known factors determine accuracy; those most noticeable are variations in context, in speaker, and in environment. Acoustic modeling plays a critical role in improving accuracy and is arguably the central part of any speech recognition system.

For the given acoustic observation $X = X_1X_2 \ldots X_n$, the goal of speech recognition is to find out the corresponding word sequence $\hat{W} = w_1w_2 \ldots w_m$ that has the maximum posterior probability $P(W \mid X)$ as expressed by

$$\hat{W} = \arg\max_{w} P(W \mid X) = \arg\max_{w} \frac{P(W)P(X \mid W)}{P(X)}$$ **Equation 1**

Since the maximization of Equation 1 is carried out with the observation X fixed, the above maximization is equivalent to maximization of the following equation

$$\hat{W} = \arg\max_{w} P(W)P(X \mid W)$$ **Equation 2**

The practical challenge is how to build accurate acoustic models, $P(X \mid W)$, and language models, $P(W)$, that can truly reflect the spoken language to be recognized.

For large vocabulary speech recognition, since there are a large number of words, we need to decompose a word into a subword sequence. Thus $P(X \mid W)$ is closely related to phonetic modeling. $P(X \mid W)$ should take into account speaker variations, pronunciation variations, environment variations, and context-dependent phonetic coarticulation variations. Last, but not least, any static acoustic or language model will not meet the needs of real applications. So it is vital to dynamically adapt both $P(W)$ and $P(X \mid W)$ to maximize $P(X \mid W)$ while using the spoken language system. The decoding process of finding the best matched word sequence W to match the input speech signal X in speech recognition systems is more than a simple pattern recognition problem, since in continuous speech recognition you have an infinite number of word patterns to search.

2.1.1 Variability in the Speech Signal

The research community has produced technologies that, with some constraints, can accurately recognize spoken input. Admittedly, today's state-of-the-art systems still cannot match human performance. Although we can build a very accurate speech recognizer for a particular speaker, in a particular language and speaking style, in a particular environment, and limited to a particular task, it remains a research challenge to build a recognizer that can essentially understand anyone's speech, in any language, on any topic in any free-flowing style, and in almost any speaking environment.

Accuracy and robustness are the ultimate measures for the success of speech recognition algorithms. There are many reasons why existing algorithms or systems did not deliver what people want. In the sections the follow we summarize the major factors involved.

2.1.1.1 Context Variability

Spoken language interaction between people requires knowledge of word meanings, communication context, and common sense. Words with widely different meanings and usage patterns may have the same phonetic realization. Consider the challenge represented by the following utterance.

Mr. Wright should write to Ms. Wright right away about his Ford or four door Honda.

For a given word with the same pronunciation, the meaning could be dramatically different, as indicated by *Wright*, *write*, and *right*. What makes it even more difficult is that *Ford or* and *Four Door* are not only phonetically identical, but also semantically relevant. The interpretation is made within a given word boundary. Even with smart linguistic and semantic information, it is still impossible to decipher the correct word sequence, unless the speaker pauses between words or uses intonation to set apart these semantically confusable phrases.

In addition to the context variability at word and sentence level, you can find dramatic context variability at phonetic level. The acoustic realization of phoneme

/ee/ for word *peat* and *wheel* depends on its left and right context. The dependency becomes more important in fast speech or spontaneous speech conversation, since many phonemes are not fully realized.

2.1.1.2 Style Variability

To deal with acoustic realization variability, a number of constraints can be imposed on the use of the speech recognizer. For example, we can have an *isolated* speech recognition system, in which users have to pause between each word. Because the pause provides a clear boundary for the word, we can easily eliminate errors such as *Ford or* and *Four Door*. In addition isolated speech provides a correct silence context to each word so that it is easier to model and decode the speech, leading to a significant reduction in computational complexity and error rate. In practice, the word-recognition error rate of an isolated speech recognizer can typically be reduced by more than a factor of three (from 7% to 2%) as compared with to a comparable continuous speech recognition system. The disadvantage is that such an isolated speech recognizer is unnatural to most people. The throughput is also significantly lower than that for continuous speech.

In continuous speech recognition, the error rate for casual, spontaneous speech, as occurs in our daily conversation, is much higher than for carefully articulated read-aloud speech. The rate of speech also affects the word recognition rate. It is typical that the higher the *speaking rate* (words/minute), the higher the error rate. If a person whispers, or shouts, to reflect his or her emotional changes, the variation increases more significantly.

2.1.1.3 Speaker Variability

Every individual speaker is different. The speech he or she produces reflects the physical vocal tract size, length and width of the neck, a range of physical characteristics, age, sex, dialect, health, education, and personal style. As such, one person's speech patterns can be entirely different from those of another person. Even if we exclude these interspeaker differences, the same speaker is often unable to precisely produce the same utterance. Thus, the shape of the vocal tract movement and rate of delivery may vary from utterance to utterance, even with dedicated effort to minimize the variability.

For *speaker-independent* speech recognition, we typically use more than 500 speakers to build a combined model. Such an approach exhibits large performance fluctuations among new speakers because of possible mismatches in the training data between existing speakers and new ones. In particular, speakers with accents have a tangible error-rate increase of 2 to 3 times.

To improve the performance of a speaker-independent speech recognizer, a number of constraints can be imposed on its use. For example, we can have a user enrollment that requires the use to speak for about 30 minutes. With the *speaker-dependent* data and training, we may be able to capture various speaker-dependent acoustic characteristics that can significant improve the speech recognizer's performance. In practice, speaker-dependent speech recognition offers not only improved accuracy but also improved speed, since decoding can be more efficient with an accurate acoustic and phonetic model. A typical speaker-dependent speech recognition system can reduce the word recognition error by more than 30% as compared with a comparable speaker-independent speech recognition system.

The disadvantage of speaker-dependent speech recognition is that it takes time to collect speaker-dependent data, which may be impractical for some applications such as an automatic telephone operator. Many applications have to support walk-in speakers, so speaker-independent speech recognition remains an important feature. When the amount of speaker-dependent data is limited, it is important to make use of both speaker-dependent and speaker-independent data using *speaker-adaptive* training techniques. Even for speaker-independent speech recognition, you can still use speaker-adaptive training based on recognition results to quickly adapt to each individual speaker during the usage.

2.1.1.4 Environment Variability

The world we live in is full of sounds of varying loudness from different sources. When we interact with computers, we may have people speaking in the background. Someone may slam the door, or the air conditioning may start humming without notice. If speech recognition is embedded in mobile devices, such as PDAs or cellular phones, the spectrum of noises varies significantly because the owner moves around. These external parameters, such as the characteristics of the environmental noise and the type and placement of the microphone, can greatly affect speech recognition system performance. In addition to the background noises, we have to deal with noises made by speakers, such as lip smacks and non-communication words. Noise may also be present from the input device itself, such as the microphone and A/D interference noises.

In a similar manner to speaker-independent training, we can build a system by using a large amount of data collected from a number of environments; this is referred to as *multistyle training*. We can use adaptive techniques to normalize the mismatch across different environment conditions in a manner similar to speaker-adaptive training. Despite the progress being made in the field, environment variability remains as one of the most severe challenges facing today's state-of-the-art speech systems.

2.1.2 Feature Extraction

The human ear resolves frequencies non-linearly across the audio spectrum and empirical evidence suggests that designing a front-end to operate in a similar non-linear manner improves recognition performance. A popular alternative to linear prediction based analysis is therefore filterbank analysis since this provides a much more straightforward route to obtaining the desired non-linear frequency resolution. However, filterbank amplitudes are highly correlated and hence, the use of a cepstral transformation in this case is virtually mandatory if the data is to be used in a HMM based recognizer with diagonal covariances.

The filters used are triangular and they are equally spaced along the mel-scale which is defined by

$$Mel(f) = 2595\log_{10}(1+\frac{f}{700}) \qquad \textbf{Equation 3}$$

To implement this filterbank, the window of speech data is transformed using a Fourier transform and the magnitude is taken. The magnitude coefficients are then binned by correlating them with each triangular filter. Here binning means that each FFT magnitude coefficient is multiplied by the corresponding filter gain and the results accumulated. Thus, each bin holds a weighted sum representing the spectral magnitude in that filterbank channel.

Mel-Frequency Cepstral Coefficients (MFCCs) are calculated from the log filterbank amplitudes $\{m_j\}$ using the Discrete Cosine Transform

$$c_i = \sqrt{\frac{2}{N}}\sum_{j=1}^{N} m_j \cos(\frac{p \cdot i}{N}(j-0.5)) \qquad \textbf{Equation 4}$$

where N is the number of filterbank channels.

MFCCs are the parameterization of choice for many speech recognition applications. They give good discrimination and lend themselves to a number of manipulations. In particular, the effect of inserting a transmission channel on the input speech is to multiply the speech spectrum by the channel transfer function. In the log cepstral domain, this multiplication becomes a simple addition that can be removed by subtracting the cepstral mean from all input vectors. In practice, of course, the mean has to be estimated over a limited amount of speech data so the subtraction will not be perfect. Nevertheless, this simple technique is very effective in practice where it compensates for long-term spectral effects such as those caused by different microphones and audio channels.

2.1.3 Hidden Markov Models

Let each spoken word be represented by a sequence of speech vectors or observations *O*, defined as

$$O = o_1, o_2, ..., o_T$$ **Equation 5**

where o_T is the speech vector observed at time *t*. The recognition problem can then be regarded as that of computing

$$\arg\max_i \{P(w_i \mid O)\}$$ **Equation 6**

where w_i is the i'th vocabulary word. This probability is not computable directly but using Bayes' Rule gives

$$P(w_i \mid O) = \frac{P(O \mid w_i)P(w_i)}{P(O)} \qquad \textbf{Equation 7}$$

Thus, for a given set of prior probabilities $P(w_i)$, the most probable spoken word depends only on the likelihood $P(O|w_i)$. Given the dimensionality of the observation sequence O, the direct estimation of the joint conditional probability $P(o_1,o_2,\ldots|w_i)$ from examples of spoken words is not practicable. However, if a parametric model of word production such as a Markov model is assumed, then estimation from data is possible since the problem of estimating the class conditional observation densities $P(O|w_i)$ is replaced by the much simpler problem of estimating the Markov model parameters.

In HMM-based speech recognition, it is assumed that the sequence of observed speech vectors corresponding to each word is generated by a Markov model as shown in Figure 7. A Markov model is a finite state machine which changes state once every time unit and each time t that a state j is entered, a speech vector o_t is generated from the probability density $b_j(o_t)$. Furthermore, the transition from state i to state j is also probabilistic and is governed by the discrete probability a_{ij}. Figure 7 show an example of this process where the six state model moves through the state sequence X = 1, 2, 2, 3, 4, 4, 5, 6 in order to generate the sequence o_1 to o_6. Notice that in HTK, the entry and exit states of a HMM are non-emitting. This is to facilitate the construction of composite models as explained in more detail later.

The joint probability that O is generated by the model M moving through the state sequence X is calculated simply as the product of the transition probabilities and the output probabilities. So for the state sequence X in Figure 7

$$P(O, X \mid M) = a_{12} b_2(o_1) a_{22} b_2(o_2) a_{23} b_3(o_3) \ldots \qquad \textbf{Equation 8}$$

However, in practice, only the observation sequence O is known and the underlying state sequence X is hidden. This is why it is called a Hidden Markov Model.

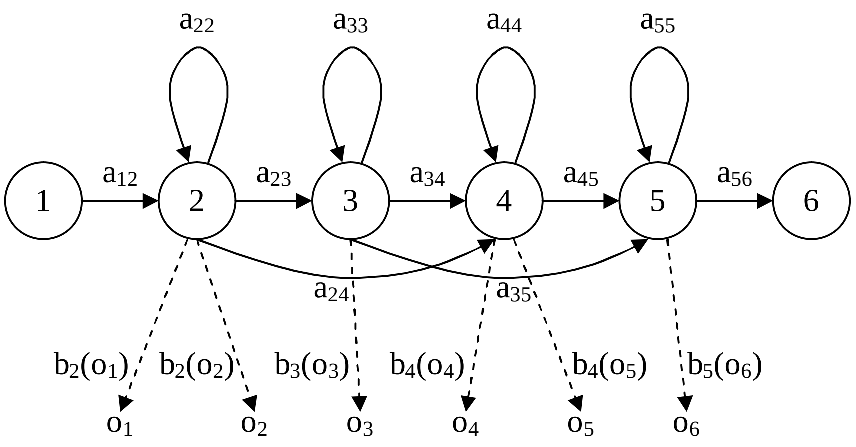


Figure 1 The Markov generation model M

Given that X is unknown, the required likelihood is computed by summing over all possible state sequences $X = x(1), x(2), x(3), \ldots, x(T)$, that is

$$P(O \mid M) = \sum_X a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(o_t) a_{x(t)x(t+1)} \qquad \textbf{Equation 9}$$

where *x(0)* is constrained to be the model entry state and *x(T+1)* is constrained to be the model exit state.

Given a set of models M_I corresponding to words w_i, equation 2 (i.e. recognition problem) is solved by using equation 3 and assuming that

$$P(O \mid w_i) = P(O \mid M_i) \qquad \textbf{Equation 10}$$

Given a set of training examples corresponding to a particular model, the parameters of that model can be determined automatically by a robust and efficient re-estimation procedure. Thus, provided that a sufficient number of representative examples of each word can be collected then a HMM can be constructed which implicitly models all of the many sources of variability inherent in real speech [1].

2.1.4 Decoding

Given a grammar network, its associated set of HMMs, and an unknown utterance, the probability of any path through the network can be computed. The task of a decoder is to find those paths which are most likely.

Decoding in HTK uses the token passing paradigm to find the best path and, optionally, multiple alternative paths. In the latter case, it generates a lattice containing the multiple hypotheses which can if required be converted to an N-best list.

Decoding is controlled by a recognition network compiled from a word-level network, a dictionary and a set of HMMs. The recognition network consists of a set of nodes connected by arcs. Each node is either a HMM model instance or word-end Each model node is itself a network consisting of states connected by arcs. Thus, once fully compiled, a recognition network ultimately consists of HMM states connection by arcs. Thus, once fully compiled, a recognition network ultimately consists of HMM states connected by transitions. However it can be viewed at three different levels: word, model and state.

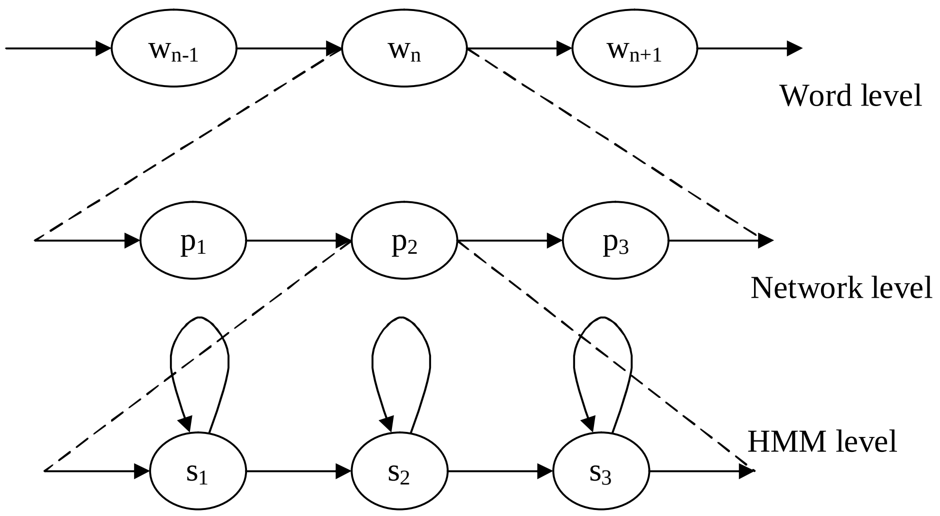


Figure 2 Recognition Network Levels

For an unknown input utterance with T frames, every path from the start node to the exit node of the network which passes through exactly T emitting HMM states is a potential recognition hypothesis. Each of these paths has a log probability which is computed by summing the log probability of each individual transition in the path and the log probability of each emitting state generating the corresponding observation. Within-HMM transitions are determined from the HMM parameters, between-model transitions are constant and word-end transitions are determined by the language model likelihoods attached to the word level networks.

The job of the decoder is to find those paths through the network which have the highest log probability. These paths are found using a Token Passing algorithm. A token represents a partial path through the network extending from time 0 through to time t. At time 0, a token is placed in every possible start node.

Each time step, tokens are propagated along connecting transitions stopping whenever they reach an emitting HMM state. When there are multiple exits from a node, the token is copied so that all possible paths are explored in parallel. As the token passes across transitions and through nodes its log probability is incremented by the corresponding transition and emission probabilities. A network node can hold at most N tokens. Hence, at the end of each time step, all but the N best tokens in any node are discarded.

As each token passes through the network it must maintain a history recording its route. The amount of detail in this history depends on the required recognition output. Normally, only word sequences are wanted and hence, only transitions out of word-end nodes need be recorded. However, for some purposes, it is useful to know the actual model sequence and the time of each model to model transition. Sometimes a description of each path down to the state level is required. All of this information, whatever level of detail is required, can conveniently be represented using a lattice structure.

Of course, the number of tokens allowed per node and the amount of history information requested will have a significant impact on the time and memory needed to compute the lattices. The most efficient configuration is N = 1 combined with just word level history information and this is sufficient for most purposes.

A large network will have many nodes and one way to make a significant reduction in the computation needed is to only propagate tokens which have some chance of being amongst the eventual winners. This process is called pruning. It is implemented at each time step by keeping a record of the best token overall and de-activating all tokens whose log probabilities fall more than a beam-width below the best. For efficiency reasons, it is best to implement primary pruning at the model rather than the state level. Thus, models are deactivated when they have no tokens in any state within the beam and they are reactivated whenever active tokens are propagated into them. State-level pruning is also implemented by replacing any token by a null (zero probability) token if it falls outside of the beam. If the pruning beam-width is set too small then the most likely path might be pruned before its token reaches the end of the utterance. This results in a search error. Setting the beam-width is thus a compromise between speed and avoiding search errors.

2.2 Text-to-Speech Synthesis

The task of a text-to-speech system can be viewed as speech recognition in reverse – a process of building a machinery system that can generate human-like speech from any text input to mimic human speakers. The basic components in a TTS system are shown in Figure 3.

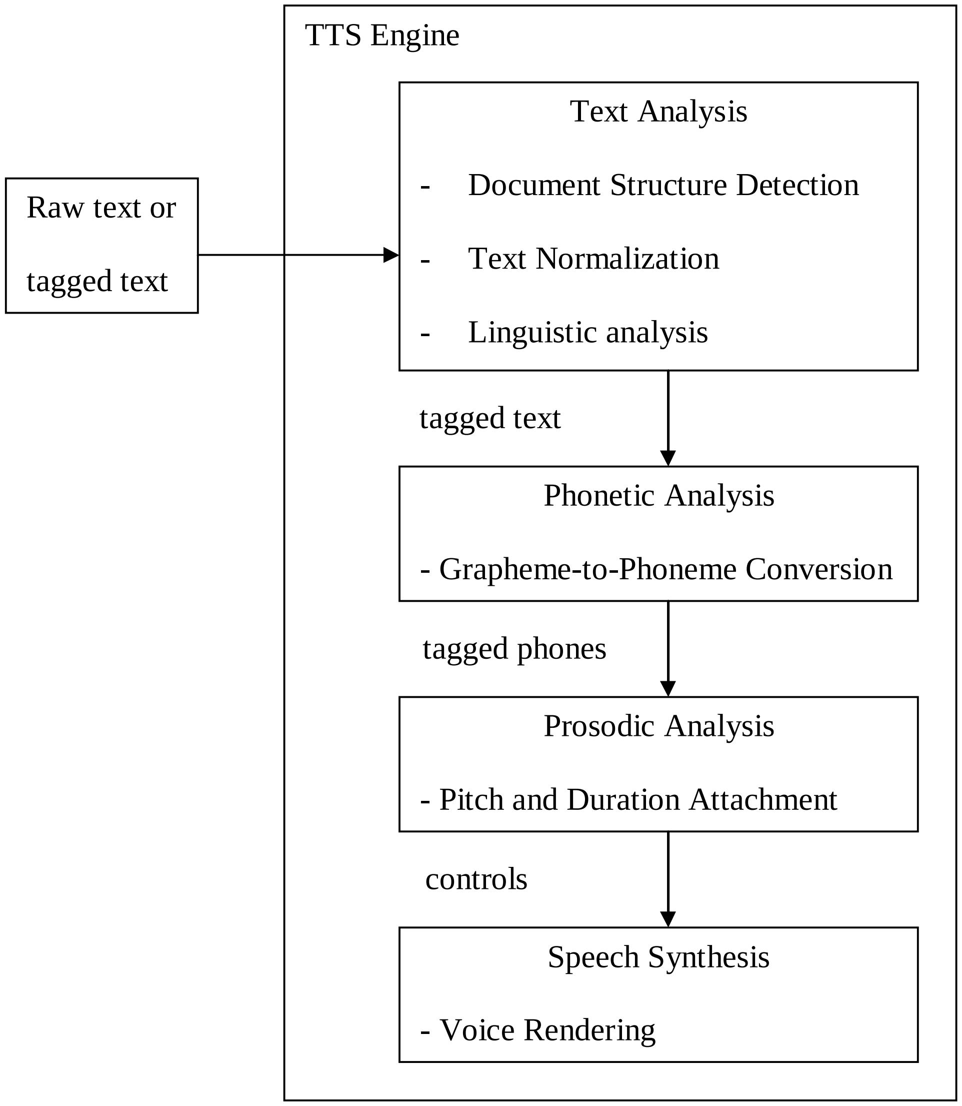


Figure 3 Basic system architecture of a TTS system

The text analysis component normalizes the text to the appropriate form so that it becomes speakable. The input can be either raw text or tagged. These tags can be used to assist text, phonetic, and prosodic analysis. The phonetic analysis component converts the processed text into the corresponding phonetic sequence, which is followed by prosodic analysis to attach appropriate pitch and duration information to the phonetic sequence. Finally, the speech synthesis component takes the parameters from the fully tagged phonetic sequence to generate the corresponding speech waveform.

The text analysis module is responsible for indicating all knowledge about the text or message that is not specifically phonetic or prosodic in nature. Very simple systems do little more than convert nonorthographic items, such as numbers, into words. More ambitious systems attempt to analyze whitespaces and punctuations to determine document structure, and perform sophisticated syntax and semantic analysis on sentences to determine attributes that help the phonetic analysis to generate correct phonetic representation and prosodic generation to construct superior pitch contours. As shown in Figure 3, text analysis for TTS involves three related processes:

- Document structure detection – Document structure is important to provide a context for all later processes. In addition, some elements of document structure, such as sentence breaking and paragraph segmentation, may have direct implications for prosody.

- Text normalization – text normalization is the conversion from the variety symbols, numbers, and other nonorthographic entities of text into a common orthographic transcription suitable for subsequent phonetic conversion.
- Linguistic analysis – Linguistic analysis recovers the syntactic constituency and semantic features of words, phrases, clauses, and sentences, which is important for both pronunciation and prosodic choices in the successive processes.

The task of the phonetic analysis is to convert lexical orthographic symbols to phonemic representation along with possible diacritic information, such as stress placement. Phonetic analysis is thus often referred to grapheme-to-phoneme conversion. The purpose is obvious, since phonemes are the basic units of sound. Even though future TTS systems might be based on word sounding units with increasing storage technologies, homograph disambiguation and phonetic analysis for new words (either true new words being invented over time or morphologically transformed words) are still necessary for systems to correctly utter every word.

Grapheme-to-phoneme conversion is trivial for languages where there is a simple relationship between orthography and phonology. Such a simple relationship can be well captured by a handful of rules. Languages such as Spanish and Finnish belng to this category and are referred to as phonetic languages. English, on the other hand, is remote from phonetic language because English words often have many distinct origins. It is generally believed that the following three services are necessary to produce accurate pronunciations.

- Homograph disambiguation – It is important to disambiguate words with different senses to determine proper phonetic pronunciations, such as object (/ah b jh eh k t/) as a verb or as a noun (/aa b jh eh k t/).
- Morphological analysis – Analyzing the component morphemes provides important cues to attain the pronunciations for inflectional and derivational words.
- Letter-to-sound conversion – The last stage of the phonetic analysis generally includes general letter-to-sound rules (or modules) and a dictionary lookup to produce accurate pronunciations for any arbitrary word.

2.3 *Distributed Speech Recognition*

In a distributed speech recognition (DSR) architecture the recognizer front-end is located in the terminal and is connected over a data network to a remote back-end recognition server. DSR provides particular benefits for applications for mobile devices such as improved recognition performance compared to using the voice channel and ubiquitous access from different networks with a guaranteed level of recognition performance. To enable all these benefits in a wide market containing a variety of players including terminal manufacturers, operators server providers and recognition vendors, a standard for front-end is needed to ensure compatibility between the terminal and the remote recognizer. The STQ-Aurora DSR Working Group within ETSI has been actively developing this standard and as a result of this work the first DSR standard was published by ETSI in February, 2000 [2].

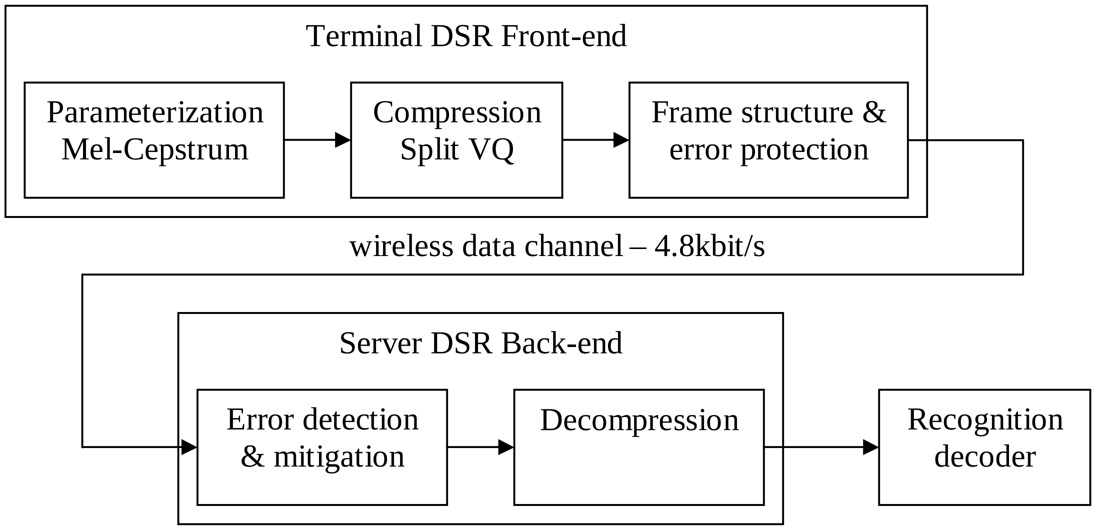

Terminal DSR Front-end
Parameterization Mel-Cepstrum
Compression Split VQ
Frame structure & error protection
wireless data channel – 4.8kbit/s
Server DSR Back-end
Error detection & mitigation
Decompression
Recognition decoder

Figure 4 Block diagram of DSR system

2.4 *Voice eXtensible Markup Language*

The VoiceXML Forum is an industry organization founded by AT&T, IBM, Lucent and Motorola. It was established to develop and promote the Voice eXtensible Markup Language (VoiceXML), a new computer language designed to make Internet content and information accessible via voice and phone [3].

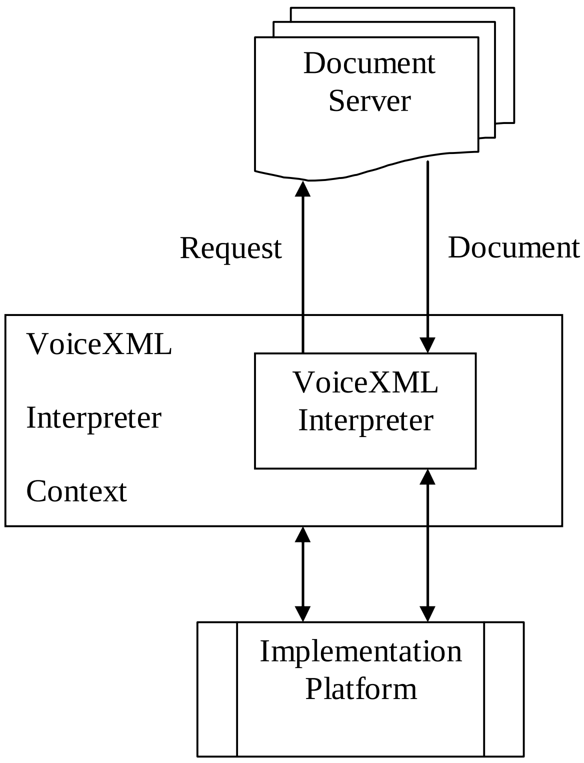


Figure 5 VoiceXML architecture model

A document server (e.g. a web server) processes requests from a client application, the VoiceXML Interpreter, through the VoiceXML interpreter context. The server produces VoiceXML documents in reply, which are processed by the VoiceXML Interpreter. The VoiceXML interpreter context may monitor user inputs in parallel with the VoiceXML interpreter. For example, one VoiceXML interpreter context may always listen for a special escape phrase that takes the user to a high-level personal assistant, and another may listen for escape phrases that alter user preferences like volume or text-to-speech characteristics.

The implementation platform is controlled by the VoiceXML interpreter context and by the VoiceXML interpreter. For instance, in an interactive voice response application, the VoiceXML interpreter context may be responsible for detecting an incoming call, acquiring the initial VoiceXML document, and answering the call, while the VoiceXML interpreter conducts the dialog after answer. The implementation platform generates events in response to user actions (e.g. spoken or character input received, disconnect) and system events (e.g. timer expiration). Some of these events are acted upon by the VoiceXML interpreter itself, as specified by the VoiceXML document, while others are acted upon by the VoiceXML interpreter context.

```
<?xml version=”1.0”?>
<vxml version=”1.0”>
    <form id=”tapered”>
        <block>
            <prompt bargein=”false”>Welcome to the ice cream survey.</prompt>
        </block>
        <field name=”flavor”>
```

```
            <grammar>vanilla | chocolate | strawberry</grammar>

            <prompt count=”1”>What is your favorite flavor?</prompt>

            <prompt count=”2”>Say chocolate, vanilla, or strawberry.</prompt>

        </field>

    </form>

</vxml>
```

The dialog might go something like this:

```
C: Welcome to the ice cream survey.

C: What is your favorite flavor?

H: Pecan praline.

C: I do not understand.                      (the “flavor” field’s prompt counter is 1)

C: What is your favorite flavor?

H: Pecan praline.

C: Say chocolate, vanilla, or strawberry.    (the prompt counter is now 2)

H: What if I hate those?

C: I do not understand.
```

Figure 6 A sample VoiceXML document and one possible corresponding dialog

Chapter 3 SYSTEM REQUIREMENT

The Automatic Speech Recognition (ASR) has a history of around 50 years, which can be traced back to 1952, when a first word recognizer, Audrey was built by Daveis, Biddulph and Balashek to recognize digits by approximating the formants. The progress made in ASR was very slow until in 1974, Hidden Markov Models (HMMs) was applied to speech recognition by Baker in the Dragon project. It was later developed by IBM (Baker, Jalinek, Bahl, Mercer) in 1976-1993. Now, it is the dominant technology for both isolated word and continuous speech recognition. The extraction of speech feature plays a critical role in the robustness of the speech recognizer. John Bridle proposed Mel Cepstra, which was used in the second major (D)ARPA ASR project in 1980's. Its variant, Mel Frequency Cepstral Coefficients (MFCC) is still one of the best feature sets used in speech recognition today [4][5][6].

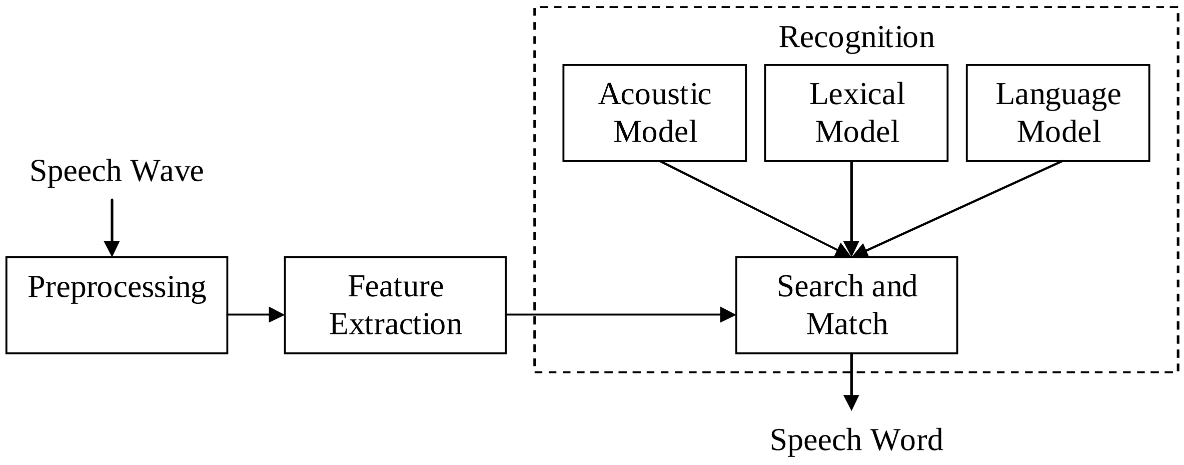


Figure 7 Components of a typical speech recognition system

We can represent all speech recognition systems ranging from command and control, dictation to IVR applications as in Figure 8. Each of them is comprised of a preprocessor, a feature extractor and a recognizer with different algorithm and complexity (Figure 2).

Applications Components	Command and control	Dictation	IVR
Preprocessing	simple noise filtering	low-complexity noise cancellation	high-complexity noise cancellation
Feature Extraction	formant	MFCCs	MFCCs
Recognition	dynamic time warping (DTW) without models	HMM with acoustic, lexical and language model	HMM with acoustic, and lexical model and grammar

Figure 8 Description of components in various speech recognition system

We are proposing a new three-tier distributed VoiceXML-based speech system, which is modified from the typical one as above. To summarize all the requirements in such a system, we have,

1. Low computational power and small memory in terminal
2. Low communication bandwidth between terminal and speech browser
3. Easy incorporation of various recognition engines and text-to-speech (TTS) synthesizers onto speech browser
4. Provision of reliable service by speech browser

Chapter 4 SOFTWARE DESIGN

4.1 System Architecture

Several modifications shown in Figure 10 are done to meet the above requirements [7].

1. Make the terminal to perform the low-complexity preprocessing and feature extraction parts excluding the high-complexity recognition part, which is now assigned to speech browser
2. Add the compressor and decompressor in between terminal and speech browser to achieve a low-bit-rate data transmission between them.

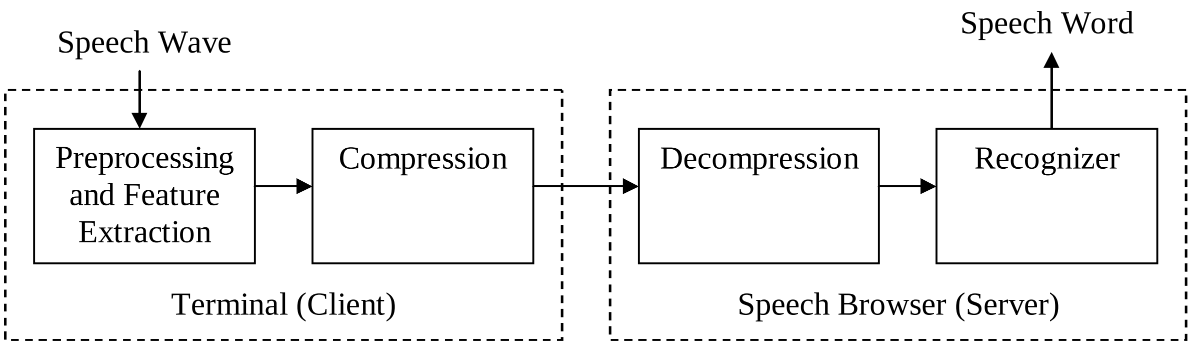


Figure 9 A client-server model of speech recognition system

Similarly, we can represent all TTS synthesis system as in Figure 4. To meet the requirements, similar modifications shown in Figure 5 are done [8].

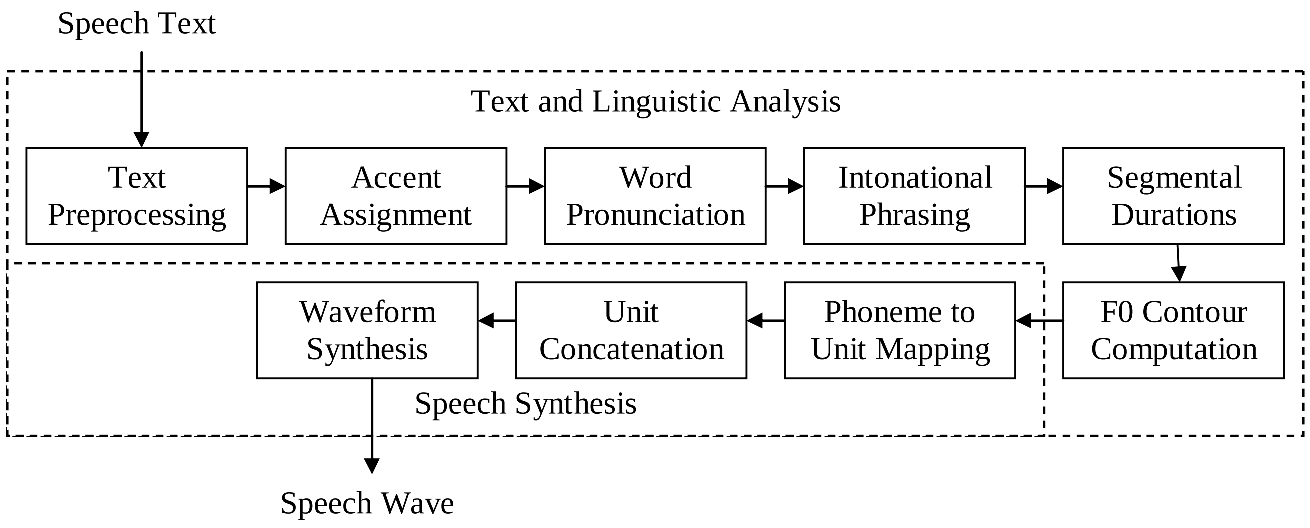


Figure 10 Components of a typical TTS synthesis system

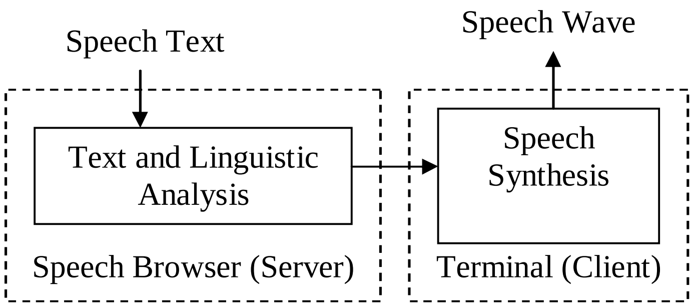


Figure 11 A client-server model of TTS synthesis system

Furthermore, load balancing and replication strategy will also be considered in speech browser

Our objectives are to:

1. Design the preprocessing and feature extraction algorithm specially for mobile devices

2. Design the compression algorithm to achieve a low-bit-rate data transmission between terminals and speech browser
3. Implement the speech browser as distributed such that recognition engines and TTS synthesizers are server objects that may locate at different hosts.
4. Break down and replicate the high-complexity above-mentioned server objects so as to achieve the advantages of load balancing, fault tolerance, scalability and parallelism.
5. Implement a recognition engine and a TTS synthesizer.

The three-tier speech system model is formed by three components, namely terminal, speech browser and document server. To realize this model, we will make a demonstration.

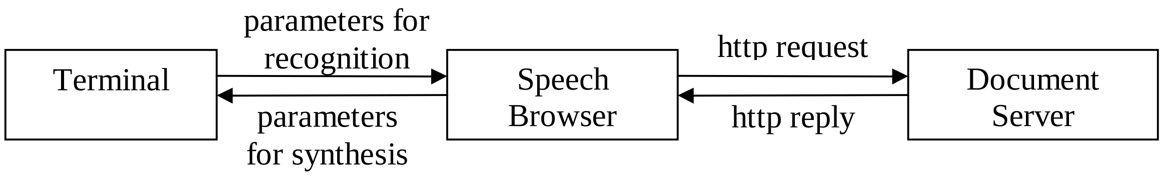


Figure 12 Three-tier speech system model

4.2 Platform

The language, platform and tools we will use in this project are as follows. Our Java platform is Jbuilder 6.0 Enterprise which incorporates Java SDK 1.4.

Components	Tools
Terminal	Java Sound API [9]
VoiceXML interpreter	Open VXI, SAXParser
Recognition engine	HTK
TTS synthesizer	FreeTTS [10]

4.3 VoiceXML file

A VoiceXML file for the credit card authorization was prepared as follows.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE vxml SYSTEM "voicexml1-0.dtd">

<vxml version="1.0">

  <form>

    <block>We now need your name, credit card type, number and expiration date.</block>


    <field name="name">

      <prompt>What is your name?</prompt>

      <!-- This is an in line grammar. -->

      <grammar>

          thomas cheng          {thomascheng}

        | wendy how             {wendyhow}

        | peter sun             {petersun}

        | may wong              {maywong}

       </grammar>

    </field>


    <field name="card_type">

      <prompt>What kind of credit card do you have?</prompt>

      <!-- This is an in line grammar. -->

      <grammar>

          visa                  {visa}

        | master [card]         {mastercard}
```

```
            | american [express]      {americanexpress}
          </grammar>
        </field>

        <field name="card_num" type="digits">
          <prompt>What is your card number?</prompt>
        </field>

        <field name="expiry_date" type="digits">
          <prompt>What is your card's expiration date?</prompt>
        </field>

        <field name="confirm" type="boolean">
          <prompt><value expr="name"/>, I have a <value expr="card_type"/>
            number <value expr="card_num"/> expiring on <value expr="expiry_date"/>.
            To go on say yes, to reenter say no.</prompt>
            <filled>
              <if cond="confirm">
                <submit next="place_order.asp"
                  namelist="name card_type card_num expiry_date"/>
              </if>
            </filled>
        </field>
    </form>
</vxml>
```

The dialog might go something like this:

S – Speech browser

T – Terminal

S: Welcome to credit card centre.

S: We now need your name, credit card type, number and expiration date.");

S: What is your name?

T: Thomas Cheng.

S: What kind of credit card do you have?

T: Visa

S: What is your card number?

T: 1 2 3 4 5 6 7 8

S: What is your card's expiration date?

T: 1 2 0 3

S: Thomas Cheng, I have a Visa number 1 2 3 4 5 6 7 8 expiring on 1 2 0 3.

S: To go on say yes, to reenter say no.

T: Yes

4.4 HMM training by HTK

Here are the ten examples of sentence that might be recognized by the speech browser based on the above VoiceXML file.

1. ZERO SIX SIX NINE

2. MASTER CARD

3. NO

4. MAY WONG

5. WENDY HOW

6. SEVEN TWO TWO SEVEN

7. AMERICAN

8. YES

9. PETER SUN

10. THOMAS CHENG

In this project, we implement a speech recognition engine using a open source HMM toolkit (HTK), that is primarily designed for building HMM-based speech processing tools, in particular recognizers. Firstly, the HTK training tools are used to estimate the parameters of a set of HMMs using training utterances and their associated transcriptions. Secondly, unknown utterances are transcribed using the HTK recognition tools [11].

Our target is to build a large-vocabulary dialog-based speaker-independent speech recognizer (LVCSR). Large-vocabulary generally means that the systems have a vocabulary of roughly 5,000 to 60,000 words. The term dialog-based means that the words are run together in a restricted way defined in grammars. Speaker-independent means that the systems are able to recognize speech from people whose speech the system has never been exposed to before.

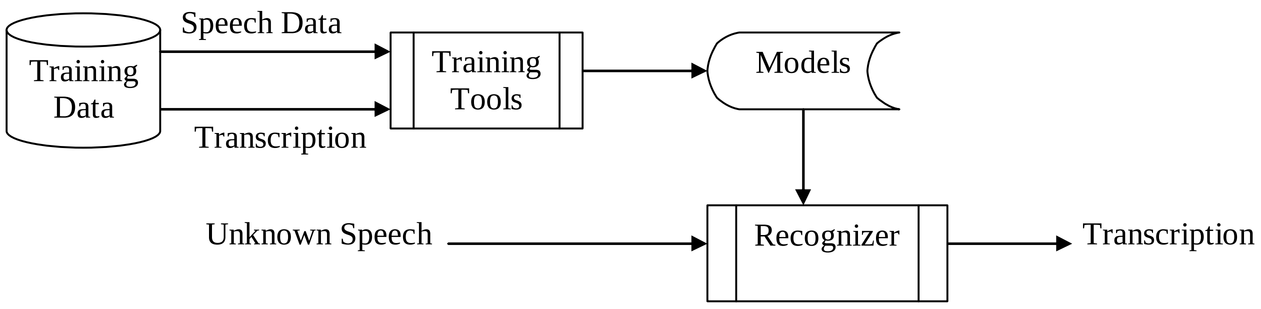


Figure 13 Scheme of ASR system functions

The thirty-year old research activity in the speech recognition area has produced a well-consolidated technology based on Hidden Markov Models (HMMs) that is also firmly supported theoretically. The HMM technology, that has the powerful capability of modeling complex (non-stationary phenomena and the availability of efficient algorithms managing very large amounts of data, is now available for Automatic Speech Recognition (ASR) tasks owing to low-cost high-performance commercial products [12][13].

4.5 Terminal

Terminal usually has small sizes, limited computing powers, small memories and limited bandwidth networks. It acts as a front-end distributed speech recognizer as well as a front-end distributed text-to-speech synthesizer. A Java applet, running on the top of JVM inside a web browser, makes use of Java Sound API and Java Network API.

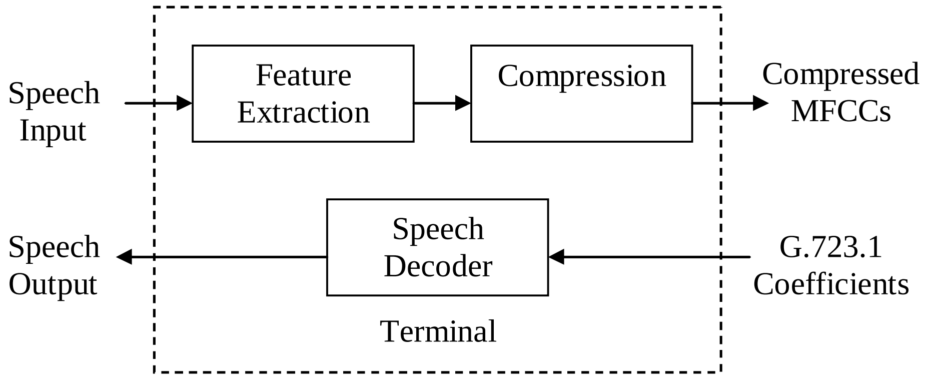


Figure 14 System architecture (Terminal)

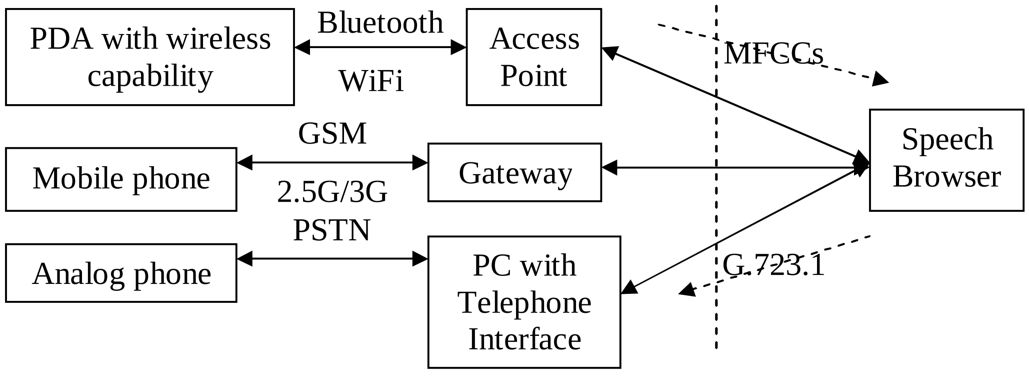


Figure 15 Examples of terminals

In the recognition path, it performs voice capturing, feature extraction, compression and then data sending to speech browser. In the synthesis path, it performs data retrieval from speech browser, speech decoder and then voice playback.

Our configuration of speech input will be 16-bit linear quantization and 16kHz sampling rate while that of speech output will be 8-bit linear quantization and 8kHz sampling rate, which are adequate to maintain the recognition accuracy and speech intelligibility.

The feature vectors after feature extraction are 13-dimensional vectors consisting of mel-cepstral features and they are computed every 10ms, which is the typical frame rate in most speech recognition systems. Owing to the high correlation (in time) between subsequent feature vector, we take the difference between adjacent feature vectors, which is a 1-step (scalar) linear prediction such that it can be compressed very much before sending to speech browser. Further compression can be achieved by multi-stage vector quantizing these error vectors.

We cannot use the text and linguistic processing part in IBM ViaVoice TTS SDK for Windows as a separate module, which originally will be our back-end TTS. To solve it, we add a pair of 5.3kbps G.723.1 speech encoder and decoder into the whole system. In speech browser, ViaVoice TTS accompanying with the encoder, which is responsible for compressing the TTS output, will be our back-end TTS. In terminal, a corresponding decoder is responsible for recovering the TTS output.

Figure 16 shows the terminal running as a Java applet. When the applet starts, it makes a socket connection to the speech browser to request a particular VoiceXML file by using Java Network API. The speech browser goes to retrieve the file in the Internet via HTTP protocol. This file is then processed by the VoiceXML interpreter running in the speech browser and the dialog will start automatically. A welcome speech specified in the dialog is always generated by the speech browser and it is sent to the terminal for playback.

The terminal can respond the speech browser in three different ways. By selecting the "Enable Audio Input" box, it starts the recording. A endpoint detector, based on the energy threshold and zero crossing, distinguish between the speech and silence. The feature extraction works only when the speech is detected and the resulting feature will be sent to speech browser.

You can also input the text directly or playback a pre-stored wave file to respond the speech browser. These methods are normally used in the debug mode or when the environment is too noisy. All the speeches coming from the terminal and the speech browser will be displayed on the screen.

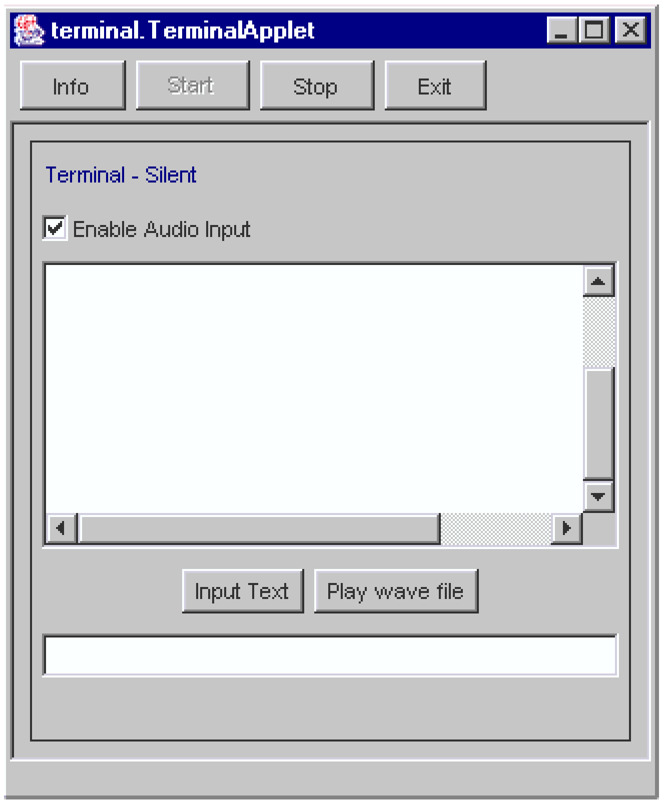


Figure 16 A Java applet running as a terminal

Figure 17 shows a block diagram of the DSR front-end

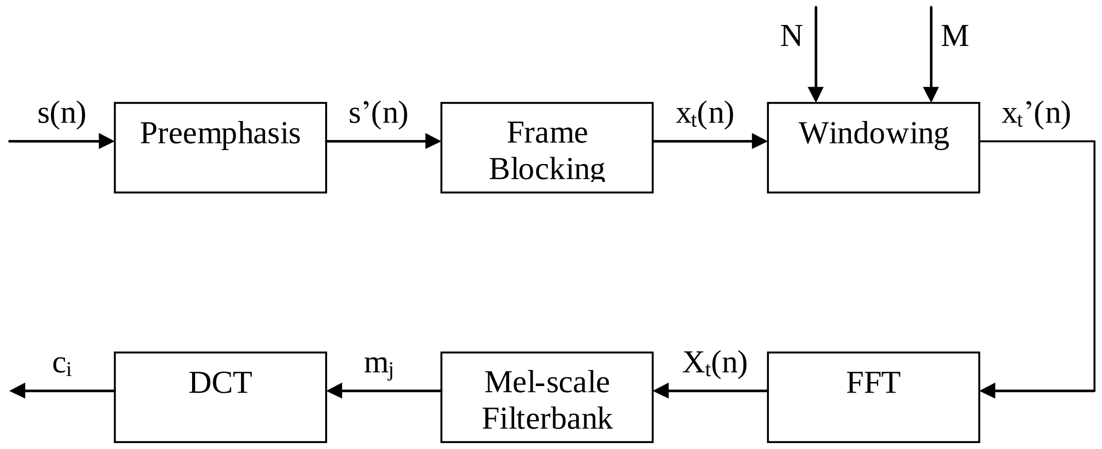


Figure 17 Block diagram of the DSR front-end

1. Signal Acquisition

Two parameters should be determined, sampling rate and A/D precision.

As we target PDAs, mobile phones and analog phones as the terminals, the best affordable parameters should be set. Mobile phones and PDAs have enough power to do the feature extraction at the sufficient high sampling rate and A/D precision. Usually, 16-bit A/D precision is used.

It shows some empirical relative word recognition error increase using a number of different sampling rates. If we take the 8 kHz sampling as our baseline, we can reduce the word recognition error with a comparable recognizer by about 10% if we increase the sampling rate to 11 kHz. If we further increase the sampling rate to 16 kHz, the word recognition error rate can be further reduced by additional 10%. Further increasing the sampling rate to 22 kHz does not have any additional impact on the word recognition errors, because most of the salient speech features are within 8 kHz bandwidth.

Relative error rate reduction with different sampling rates

Sampling Rate	Relative Error-Rate Reduction
8 kHz	Baseline
11 kHz	+10%
16 kHz	+10%
22 kHz	+0%

3. End-point detection

To lower the workload of the terminal and speech browser, we implement a end-point detection. Feature extraction starts only when the speech is detected.

4. Preemphasis

   The digitized speech signal, s(n), was put through a preemphasizer (typically a first-order FIR filter), to spectrally flatten the signal and to make it less susceptible to finite precision effects later in the signal processing. In this project, we chose 0.97 as the preemphasis coefficient.

$$H(z) = 1 - az^{-1}, \quad 0.9 \le a \le 1.0 \qquad \textbf{Equation 11}$$

5. Frame Blocking

In this step the preemphasized speech signal, s'(n), was blocked into frames of N samples, with adjacent frames being separated by M samples. We chose N = 320 and M = 160.

6. Windowing

   The next step in the preprocessing was to window each individual frame so as to minimize the signal discontinuities at the beginning and end of each frame. Hamming window is used, which has the form

$$w(n) = 0.54 - 0.46\cos(\frac{2pn}{N-1}), \quad 0 \le n \le N-1 \qquad \textbf{Equation 12}$$

7. Finally, 39 coefficients were generated for each frame. This number, 39, was computed from the length of parameterised static vectors (MFCC_0 = 13) plus the delta coefficients (+13) plus the acceleration coefficients (+13) after the standard FFT, Mel-scale filterbank and DCT, described in 2.1.2 above.

In general, time-domain features are much less accurate than frequency-domain features such as the mel-frequency cepstral coefficients (MFCC). This is because many features such as formants, useful in discriminating vowels, are better characterized in the frequency domain with a low-dimension feature vector. Temporal changes in the spectra play an important roles in human perception. One way to capture this information is to use delta coefficients that measure the change in coefficients over time.

When 16 kHz sampling rate is used, a typical state-of-the-art speech system can be built based on the following features

13th-order MFCC ck;

13th-order 40-msec 1st-order delta MFCC computed from $\Delta ck = ck+2 – ck-2$;

13th-order 2nd-order delta MFCC computed from $\Delta\Delta ck = \Delta ck+1 - \Delta ck-1$;

The 13th-order MFCC outperforms 13th-order LPC cepstrum coefficients, which indicates that perception-motivated mel-scale representation indeed helps recognition. In a similar manner, perception-based LPC features such as PLP can achieve similar improvement. The MFCC order has also been studied experimentally for speech recognition. The higher-order MFCC does not further reduce the error rate in comparison with the 13th-order MFCC, which indicates that the first 13 coefficients already contain most salient information needed for speech recognition. In addition to mel-scale representation, another percepion-motivated feature such as the first- and second-order delta features can significantly reduce the word recognition error, which the higher-order delta features provide no further information.

Relative error rate reduction with different features

Feature Set	Relative Error-Rate Reduction
13th-order LPC cepstrum coefficients	Baseline
13th-order MFCC	+10%

16th-order MFCC	+0%
+1st- and 2nd-order dynamic features	+20%
+3rd-order dynamic features	+0%

If we make units context dependent, we can significantly improve the recognition accuracy, provided there are enough training data to estimate these context-dependent parameters. Context-dependent phonemes have been widely used for large-vocabulary speech recognition, thanks to its significantly improved accuracy and trainability. A context usually refers to the immediately left and/or right neighboring phones.

A triphone model is a phonetic model that takes into consideration both the left and the right neighboring phones. If two phones have the same identity but different left or right contexts, they are considered different triphones.

The left and right contexts used in triphones, while important, are only two of many important contributing factors that affect the realization of a phone. Triphone models are powerful because they capture the most important coarticulatory effects. They are generally much more consistent than context-independent phone models. However, as context-dependent models generally have increased parameters, trainability becomes a challenging issue. We need to balance the trainability and accuracy with a number of parameter-sharing techniques.

If we use context-independent phone (monophone) as our baseline, we can reduce the word recognition error with a comparable recognizer by about 25% if we use context-dependent phone (triphone, for example). Clustered triphone and seone gives us much better the word recognition error rate than the others.

Relative error reductions for different modeling units.

Feature Set	Relative Error-Rate Reduction
Context-independent phone	Baseline
Context-dependent phone	+25%
Clustered triphone	+15%
Senone	+24%

4.6 Speech Browser

Speech browser is a cluster of powerful servers acting as a back-end that performs the core recognition and synthesis processes as well as the interpretation of VoiceXML files retrieved from the document server.

The implementation platform was based on the Sourceforge Project: Open VXI VoiceXML Interpreter V4.0 using SAXParser, HMM recognizer and FreeTTS synthesizer.

The speech browser was implemented in the client/server architecture. Different objects had different requirement on the platform at which they are located. Speech recognition is a computation-intensive process while TTS synthesis is memory-intensive.

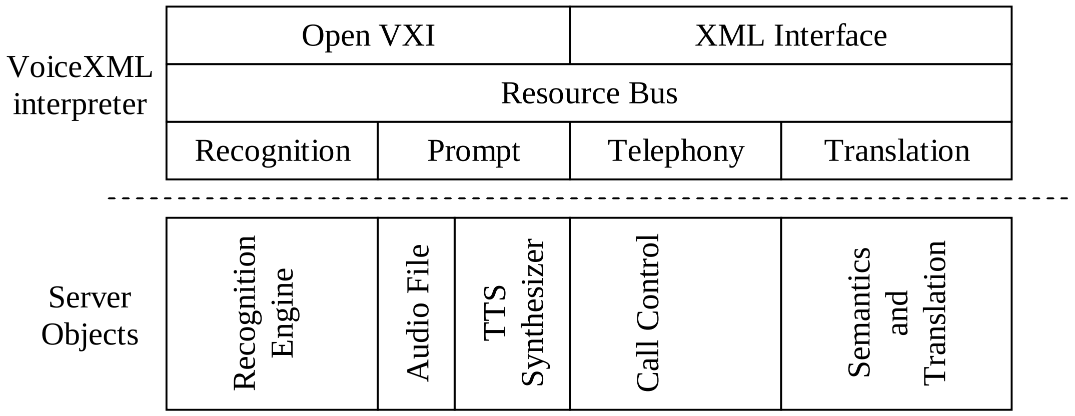


Figure 18 System architecture (Speech browser)

Chapter 5 IMPLEMENTATION

5.1 HMM Training by HTK

Here were the steps to train the HMM models by HTK.

1. The task grammar was expressed as a grammar definition language provided by HTK. For the credit card authorization application above, the suitable grammar should be

```
$name  = THOMAS CHENG | WENDY HOW | PETER SUN | MAY WONG;
$card = VISA | MASTER [CARD] | AMERICAN [EXPRESS];
$digit = ONE | TWO | THREE | FOUR | FIVE | SIX | SEVEN | EIGHT | NINE | OH |
ZERO;
$yesno = YES | NO;
( SENT-START ( $name | $card | <$digit> | $yesno ) SENT-END )
```

where the vertical bars denote alternatives, the square brackets denote optional items and the angle braces denote one or more repetitions. The complete grammar can be depicted as a network as shown in Figure 19.

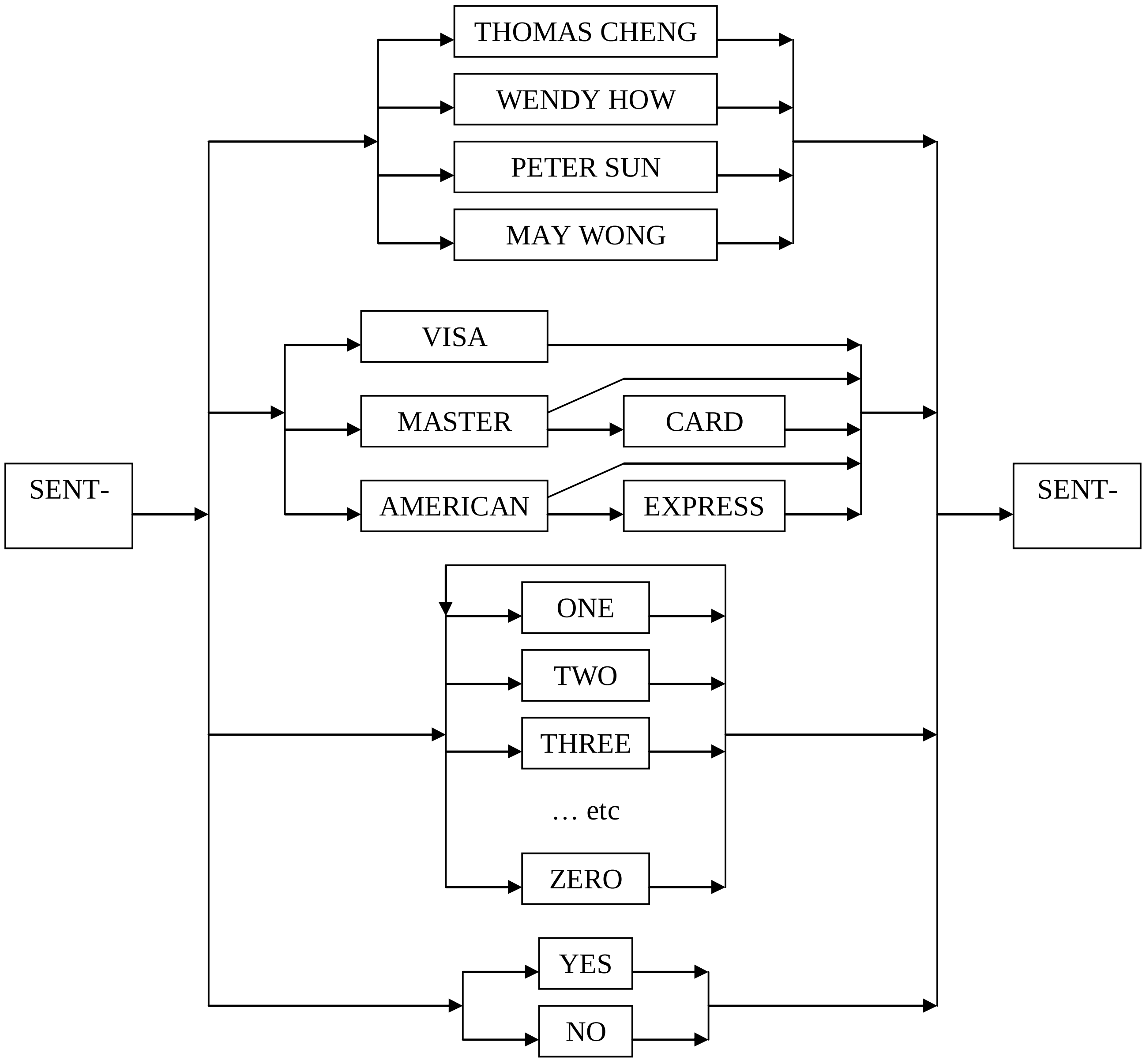


Figure 19 Grammar for Credit Card Authorization

The above high level representation of a task grammar was provided for user convenience. The HTK recognizer actually requires a word network to be defined using a low level notation called HTK Standard Lattice Format (SLF) in which each word instance and each word-to-word transition is listed explicitly.

In this SLF file, the total number of nodes and links were 32 and 63 respectively.

```
VERSION=1.0

N=32   L=63

I=0    W=SENT-END

I=1    W=NO

I=2    W=!NULL

I=3    W=YES

I=4    W=ZERO

I=5    W=!NULL

I=6    W=OH

I=7    W=NINE

I=8    W=EIGHT

I=9    W=SEVEN

I=10   W=SIX

I=11   W=FIVE

I=12   W=FOUR

I=13   W=THREE

I=14   W=TWO

I=15   W=ONE

I=16   W=EXPRESS

I=17   W=AMERICAN

I=18   W=CARD
```

```
I=19   W=MASTER

I=20   W=VISA

I=21   W=WONG

I=22   W=MAY

I=23   W=SUN

I=24   W=PETER

I=25   W=HOW

I=26   W=WENDY

I=27   W=CHENG

I=28   W=THOMAS

I=29 W=SENT-START

I=30   W=!NULL

I=31   W=!NULL

J=0     S=0    E=30

J=1     S=1    E=2

J=2     S=2    E=0

J=3     S=3    E=2

J=4     S=4    E=5

J=5     S=5    E=0

J=6     S=5    E=15

J=7     S=5    E=14

J=8     S=5    E=13

J=9     S=5    E=12
```

```
J=10    S=5    E=11

J=11    S=5    E=10

J=12    S=5    E=9

J=13    S=5    E=8

J=14    S=5    E=7

J=15    S=5    E=6

J=16    S=5    E=4

J=17    S=6    E=5

J=18    S=7    E=5

J=19    S=8    E=5

J=20    S=9    E=5

J=21    S=10   E=5

J=22    S=11   E=5

J=23    S=12   E=5

J=24    S=13   E=5

J=25    S=14   E=5

J=26    S=15   E=5

J=27    S=16   E=2

J=28    S=17   E=16

J=29    S=17   E=0

J=30    S=18   E=2

J=31    S=19   E=18

J=32    S=19   E=0
```

```
J=33   S=20   E=2

J=34   S=21   E=2

J=35   S=22   E=21

J=36   S=23   E=2

J=37   S=24   E=23

J=38   S=25   E=2

J=39   S=26   E=25

J=40   S=27   E=2

J=41   S=28   E=27

J=42   S=29   E=3

J=43   S=29   E=1

J=44   S=29   E=20

J=45   S=29   E=19

J=46   S=29   E=17

J=47   S=29   E=28

J=48   S=29   E=26

J=49   S=29   E=24

J=50   S=29   E=22

J=51   S=29   E=15

J=52   S=29   E=14

J=53   S=29   E=13

J=54   S=29   E=12

J=55   S=29   E=11
```

J=56	S=29	E=10
J=57	S=29	E=9
J=58	S=29	E=8
J=59	S=29	E=7
J=60	S=29	E=6
J=61	S=29	E=4
J=62	S=31	E=29

2. Build the dictionary containing a sorted list of the required words

A dictionary that contained the pronunciations for each word used in the application.

AMERICAN	ax m eh r ih k ax n sp
CARD	k aa d sp
CHENG	ch ae ng sp
EIGHT	ey t sp
EXPRESS	ih k s p r eh s sp
FIVE	f ay v sp
FOUR	f ao r sp
FOUR	f ao sp
HOW	hh aw sp
MASTER	m aa s t ax r sp
MASTER	m aa s t ax sp

```
MAY             m ey sp

NINE            n ay n sp

NO              n ow sp

OH              ow sp

ONE             w ah n sp

PETER           p iy t ax r sp

PETER           p iy t ax sp

SENT-END        sil sp

SENT-START      sil sp

SEVEN           s eh v n sp

SIX             s ih k s sp

SUN             s ah n sp

THOMAS          t oh m ax s sp

THREE           th r iy sp

TWO             t uw sp

VISA            v iy z ax sp

WENDY           w eh n d iy sp

WONG            k oh ng sp

YES             y eh s sp

ZERO            z ia r ow sp
```

A list of 34 phones was output.

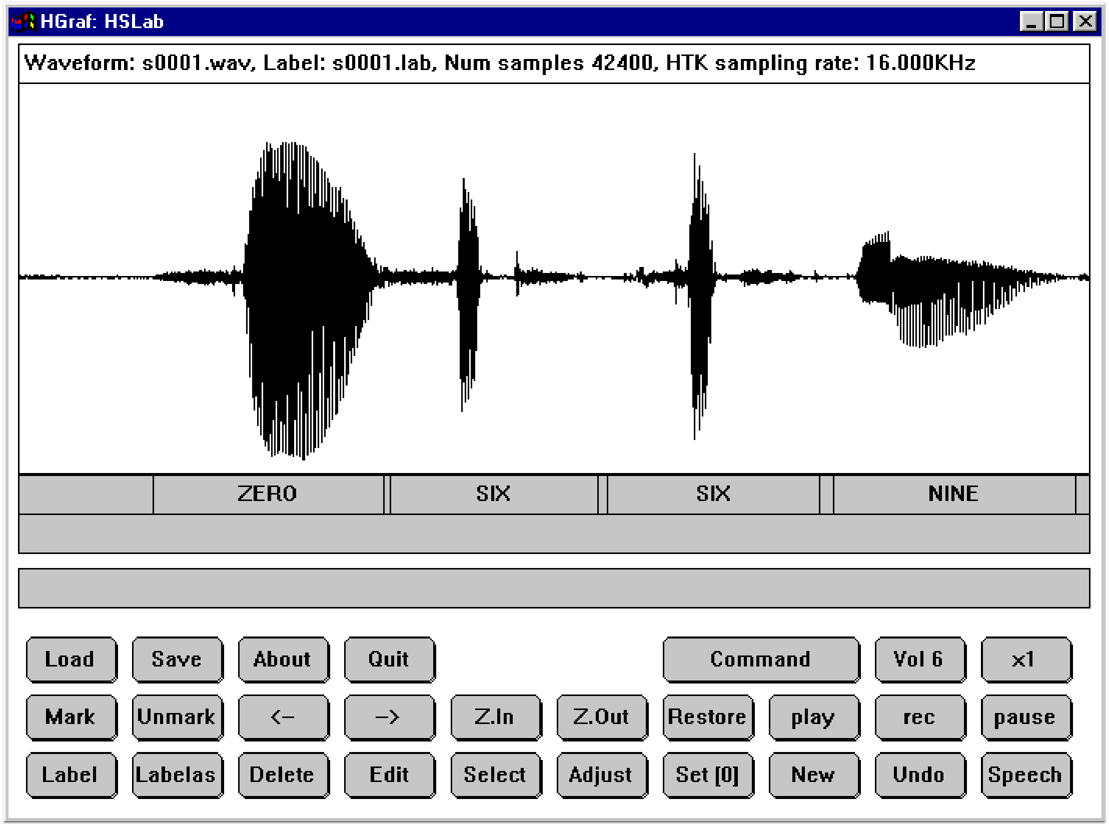
ax, m, eh, r, ih, k, n, sp, aa, d, ch, ae, ng, ey, t, s, p, f, ay, v, ao, hh, aw, ow, w, ah, iy, oh, th, uw, z, y, ia, sil

A list of 92 triphones was output. The HMMs was continuous density mixture Gaussian tied-state triphones with clustering performed using phonetic decision trees.

sil, z+ia, z-ia+r, ia-r+ow, r-ow, sp, s+ih, s-ih+k, ih-k+s, k-s, n+ay, n-ay+n, ay-n, th+r, th-r+iy, r-iy, s+eh, s-eh+v, eh-v+n, v-n, f+ay, f-ay+v, ay-v, m+aa, m-aa+s, aa-s+t, s-t+ax, t-ax, k+aa, k-aa+d, aa-d, w+ah, w-ah+n, ah-n, f+ao, f-ao, ey+t, ey-t, t+uw, t-uw, ow, n+ow, n-ow, m+ey, m-ey, k+oh, k-oh+ng, oh-ng, w+eh, w-eh+n, eh-n+d, n-d+iy, d-iy, hh+aw, hh-aw, f-ao+r, ao-r, ax+m, ax-m+eh, m-eh+r, eh-r+ih, r-ih+k, ih-k+ax, k-ax+n, ax-n, y+eh, y-eh+s, eh-s, p+iy, p-iy+t, iy-t+ax, s+ah, s-ah+n, t+oh, t-oh+m, oh-m+ax, m-ax+s, ax-s, ch+ae, ch-ae+ng, ae-ng, v+iy, v-iy+z, iy-z+ax, z-ax, t-ax+r, ax-r, ih+k, k-s+p, s-p+r, p-r+eh, r-eh+s

3. Record and label the 200 training data

There is a HTK tool for both waveform recording and labeling. 200 training utterances were recorded and labeled according to the defined grammar.



4. Create monophone HMMs

For the state-dependent left-to-right HMM, the most important parameter in determining the topology is the number of states. The choice of model topology depends on available training data and what the model is used for. In our case, each HMM is used to represent a phone, at least three to five output distributions are needed.

```
~o <VecSize> 39 <MFCC_0_D_A>

~h "proto"

<BeginHMM>
```

```
 <NumStates> 5
 <State> 2
   <Mean> 39
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   <Variance> 39
     1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <State> 3
   <Mean> 39
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   <Variance> 39
     1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <State> 4
   <Mean> 39
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   <Variance> 39
     1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <TransP> 5
```

```
  0.0 1.0 0.0 0.0 0.0

  0.0 0.6 0.4 0.0 0.0

  0.0 0.0 0.6 0.4 0.0

  0.0 0.0 0.0 0.7 0.3

  0.0 0.0 0.0 0.0 0.0

<EndHMM>
```

5. A series of training was then carried out including creating flat start monophones, fixing the silence models, realigning the training data, making tied-state triphones from monophones. The recognizer was complete and its performance can be evaluated.

   The line starting with SENT: indicates that of the 200 test utterances, 187 (93.5%) were correctly recognized. The following line starting with WORD: gives the word level statistics and indicates that of the 990 words in total, 990 (100%) were recognized correctly. There was no deletion error (D), no substitution error (S) and 14 insertion error (I). The accuracy figure (Acc) of 98.59% is lower than the percentage correct (Cor) because it takes amount of the insertion errors which the latter ignores.

```
==================== HTK Results Analysis =====================

  Date: Fri May 17 02:44:46 2002

  Ref : testref.mlf

  Rec : recout.mlf
```

```
----------------------- Overall Results -------------------------
SENT: %Correct=93.50 [H=187, S=13, N=200]
WORD: %Corr=100.00, Acc=98.59 [H=990, D=0, S=0, I=14, N=990]
===============================================================
```

5.2 Terminal

The terminal is mainly responsible for voice/text input, endpoint detection, feature extraction and parameters passing with speech browser.

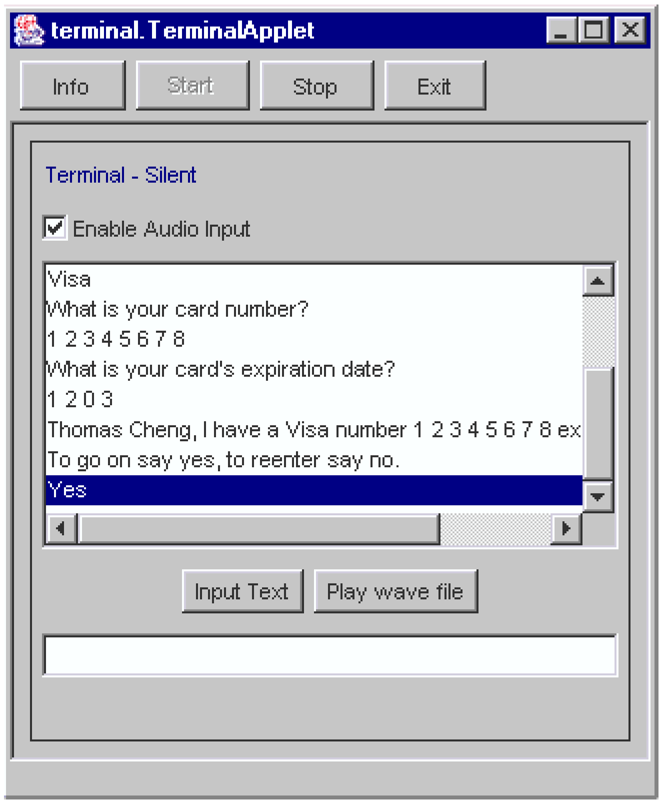
Class Name	Main Function
WaveDataInputStream	read the 16-bit sampled waveform from the wave file
FeatureExtraction	transform sampled waveform to parameter representation for recognition, i.e. MFCC
FeatureDecode	transform a parameter representation for synthesis to sampled waveform
FeatureOutputStream	send the parameter representation for recognition, i.e. MFCC to the speech browser
FeatureInputStream	receive the parameter representation for synthesis to the speech browser

5.3 Speech Browser

The speech browser is mainly responsible for parameters passing with terminal, VoiceXML file request and interpretation, speech recognition and synthesis.

Class Name	Main Function
FeatureOutputStream	send the parameter representation for synthesis to the terminal
FeatureInputStream	receive the parameter representation for recognition, i.e. MFCC from the terminal
VXMLInterpret	request and interpret the VoiceXML file
SpeechRecognize	transform the parameter representation for recognition, i.e. MFCC to the text
SpeechSynthesize	transform the text to the parameter representation for synthesis

Chapter 6 RESULT AND DISCUSSION



```
Visa
What is your card number?
1 2 3 4 5 6 7 8
What is your card's expiration date?
1 2 0 3
Thomas Cheng, I have a Visa number 1 2 3 4 5 6 7 8 ex
To go on say yes, to reenter say no.
Yes
```

The result was shown as below. When the terminal started, it automatically loaded a credit card authorization application from the speech browser. The user followed the dialog specified in the VoiceXML and the terminal finally retrieved the name and credit card information of the user via his/her voices.

Some improvements can be made in the future.

- Speaker adaptation

Since the speech browser knows who is calling from in the terminal side, it is possible to include the speaker adaptation algorithm in the current model in order to improve the accuracy.

- To make the speech browser become CORBA-based

The computation and memory requirements of speech recognition and speech synthesis tasks are different and it is more suitable to let them run in different hosts. In addition, the loading of the speech browser may vary very much from time to time since the number of terminals using the speech browser is unpredictable. In this case, a CORBA-based system should be introduced to make the system to be more efficient and robust where some loading balancing and fault tolerant features is added.

Chapter 7 CONCLUSION

In this project, we introduce a three-tier distributed VoiceXML-based speech system and demonstrate a credit card authorization application as one of the examples. Under this model, the service of a high performance speech system can be provided to enormous users via mobile devices in a unified way.

Chapter 8 REFERENCES

1 X.D. Huang, Y. Ariki and M.A. Jack, Hidden Markov models for speech recognition, Edinburgh University Press, c1990

2 Aurora Project: Distributed speech recognition, http://www.etsi.org/technicalactiv/dsr.htm

3 Voice eXtensible Markup Language Version 1.00, Voice Forum, c2000

4 Lawrence R. Rabiner, Ronald W. Schafer, Digital processing of speech signals, c1978

5 John R. Deller, Jr., John G. Proakis, John H.L. Hansen, Discrete-time processing of speech signals

6 Lawrence Rabiner, Biing-Hwang Juang, Fundamentals of speech recognition, Prentice Hall, c1997

7 Ganesh N. Ramaswamy, Ponani S. Gopalakrishnan, Compression of acoustic features for speech recognition in network environments, IBM Thomas J. Watson Research Center

8 Bell Labs Voice and Audio Technologies For Portable MP3 Players, http://www.lucentssg.com/speech/MP3.pdf

9 Java™ sound API programmer’s guide, Sun Microsystems, Inc., c2000

10 *FreeTTS 1.1* - A speech synthesizer written entirely in the Java^{TM} programming language, **http://freetts.sourceforge.net/**

11 Steve Young, Dan Kershaw, Julian Odell, Dave Ollason, Valtcho Valtchev, Phil Woodland, The HTK book (for HTK Version 3.0), Microsoft Corporation, c2000

12 Caludio Becchetti and Lucio Prina Ricotti, Speech recognition – theory and C++ implementation, John Wiley & Sons, c1999

13 L. Rabiner, Fundamental of speech recognition, Prentice Hall, c1993