FRAMEWORK AND IMPLEMENTATION FOR DIALOG BASED ARABIC SPEECH RECOGNITION

BY

MOHAMED MAHMOUD ALI

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

JANUARY 2012

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This dissertation, written by MOHAMMED MAHMOUD ALI under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE IN COMPUTER SCIENCE.

Thesis Committee

[illegible]

Thesis Advisor
Dr. Wasfi G. Al-Khatib

[illegible]

Co-advisor
Prof. Moustafa Elshafei

Sayed

Member
Dr. El-Sayed M. El-Alfy

[illegible]

Member
Dr. M. Salahadin

[illegible]

Member
Dr. Lahouari Ghouti

عادل

Department Chairman
Dr. Adel Ahmed

[illegible]

Dean of Graduate Studies
Dr. Salam Adel Zummo

16/1/12

Date

عمادة الدراسات العليا
DEANSHIP OF GRADUATE STUDIES

DEDICATION

I dedicate my thesis work to my family and friends who encouraged and supported me through the years.

I thank my parents for their love and support through all of my life.

I dedicate this work, with special gratitude, to my beloved wife for enduring all the hardships and giving me endless support to complete it.

I specially thank all my friends in Egypt and Saudi Arabia who paved the way for me to be able to reach my goals.

ACKNOWLEDGMENT

All praise is due to Allah, the Most Beneficent, and the Most Merciful.

Acknowledgment is due to the King Fahd University of Petroleum & Minerals for supporting this research.

I wish to express my heartfelt appreciation to my thesis advisor, Dr. Wasfi Al-Khatib for all his guidance and provision to make my research work a reality.

Special gratitude and sincere appreciations to my thesis co-advisor, Dr. Mustafa El-Shafei, for the endless hours he spent to mentor and guide me through my research work.

I would like to thank my committee members, Dr. El-Sayed El-Alfy, Dr. Salahadin Adam, and Dr. Lahouari Ghouti, who were more than generous with their expertise and precious time, and provided invaluable comments and criticism.

I would like to thank all faculty members who taught me throughout my graduate school.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT

NAME: MOHAMED MAHMOUD ALI

TITLE: FRAMEWORK AND IMPLEMENTATION FOR DIALOG BASED ARABIC SPEECH RECOGNITION

MAJOR: COMPUTER SCIENCE

DATE: JANUARY 2012

Spoken dialog systems have gained lots of interest for the past decades. Many applications have reached commercial level for English. However, the research on Arabic support is still scarce. VoiceXML provides a web based approach for building voice applications. It empowers the development process of Interactive Voice Response (IVR) applications with more flexible techniques and provides more opportunities for developing a unified framework. While VoiceXML was developed mainly for telephony applications, its similarity to modern web technologies allows for integration with modern web browsers.

In this thesis, we propose a framework for building and testing VoiceXML applications in a web-based environment. In addition, we investigate the requirements and possibilities to provide Arabic support to these applications. Finally, we develop a prototype application for accessing Saudi stock market using the proposed framework.

ملخص الرسالة

الاسم: محمد محمود عبد السميع محمد علي

عنوان الرسالة: إطار عمل للتعرف على الكلام العربي وتطبيقه في أنظمة التحاور الصوتية

التاريخ: يناير 2012

حققت أنظمة المحاورة الصوتية باللغة الإنجليزية نجاحا ملحوظا في السنوات الماضية. وعلى الرغم من ازدياد الاهتمام بتطوير تطبيقات باللغة العربية للتعرف الصوتي على الكلام، خصوصا في العقد المنصرم، إلا إن تطبيقاتها العملية مازالت قليلة. شهد معيار VoiceXML رواجا كبيرا في التطبيقات الأجنبية لأنظمة التحاور الصوتية. ويتميز هذا المعيار بسهولة توسيعه وتوافقه مع أنظمة الويب الحديثة مثل XML و HTML؛ مما يجعله منصة مثالية لتطوير إطار عمل للأنظمة المصممة لدعم اللغة العربية.

تقدم هذه الرسالة تفاصيل تطوير إطار عمل لدعم التعرف على الكلام العربي باستخدام معيار VoiceXML، بدءا من الخطوات اللازمة لتوفير نظام للتعرف على الكلام العربي ثم تطوير نظام يعمل بتقنية الويب للمحادثة الصوتية.

كما تقدم الرسالة مثالا لتطبيق متعلق بنظم المحادثة الصوتية للاستعلام عن أسعار الأسهم بطريقة آلية.

CHAPTER 1

INTRODUCTION

Arabic speech recognition has been one of the hot topics in the research community for the past decade. Researchers, motivated by several causes, have developed techniques and algorithms that tackle many aspects that differentiate Arabic from other, well studied, languages. However, end user applications that utilize these technologies are still scarce. It is necessary to increase the awareness of these technologies by providing end users with means to experiment them directly. It is also important to provide researchers with a unified framework where they can experiment and share their results freely. This thesis aims to shorten the gap to end users and provide researchers with a framework for Arabic speech recognition by utilizing the recent advancements in web technologies.

1.1 SPOKEN DIALOG SYSTEMS

One of the important applications of speech recognition is the development of spoken dialog systems. Spoken dialog systems, also known as Interactive Voice Response (IVR) systems, allow users to interact with the computer system to perform tasks or ask for information [1]. Using voice interface allows users to access these systems from telephones and handheld devices, providing high mobility and robustness in situations where a PC is not available.

While visual based dialog systems are dominant in day-to-day computer use, there are many situations where it is more preferred, if not mandatory, to use voice to interact with the system. Modern car navigation systems let the users ask for destinations and get directions in voice to keep their hands free and avoid any distractions that might cause accidents. Moreover, incorporating speech enhances the accessibility of computer systems for people with disabilities.

Dialogs have coherent structure and intend to achieve certain goals. It is possible, to some level, to predict the outcomes of a dialog. Dialogs can be in the form of yes/no questions or questions that expect the users to choose one of multiple answers. More complex dialogs allow the users to ask for multiple pieces of information or providing multiple constraints for a query. It is up to the dialog designer to specify the complexity of the dialog. A poorly designed dialog, however, will most probably cause user anxiety and frustration, pushing users away from benefiting from the application.

Nowadays, many call centers employ voice based inquiry systems, letting users retrieve information or perform tasks. Backed with powerful Automated Speech Recognition (ASR) systems and Text To Speech (TTS) generators, these applications achieve a rich user experience that’s both engaging and more efficient than regular desktop counterparts.

Early spoken dialog systems were designed to work solely with Public Switched Telephone Networks (PSTN), where the user input is either voice or dial tones. These applications can also be used with modern mobile phones or VoIP applications. However, modern smart phones have much better multimedia support and mostly come with fully featured web browsers and GPS navigation, allowing the development of a richer user interaction and more content delivery, such as audio, video, and maps, to the phone.

One example of a modern dialog system is Google's goog-411 service [2]. Users of this service can call the service phone number (+18004664411) and specify their location and the business they are looking for. They are then given a list of destinations to pick from. After picking a target, users are transferred to their number automatically. In addition, user can receive text and mapping information about that business if they are calling from mobile devices.

1.1.1 COMPONENTS OF A DIALOG SYSTEM

In the context of spoken dialog systems, a *dialog* is a sequence of system prompts and user responses [3]. A *prompt* is a message sent by the system to request the user to provide or confirm a set of information. Based on the user response, another subset of prompts might be issued until the dialog is terminated by either side. In IVR applications, a dialog typically starts by asking the users to provide an identifier like PIN number or location; users are then prompted to choose from the services that are available to them.

Since voice signals are likely to have high noise levels, robustness is an important factor in spoken dialog systems. It is common in IVR systems to ask users to repeat information that weren’t properly processed by the system. However, it is necessary for a successful system to reduce the number of re-prompts as much as possible to avoid user anxiety which might lead them to end the call. A common technique for re-prompting is to break down complex prompts into smaller, easily recognizable segments that have a higher chance of recognition. For instance, Google's 411 service starts by asking the users to say their location and the business they want at once. If the system is incapable of recognizing the user input, users are prompted to say the city and state only and then, after successful recognition of these tokens, they are asked to say the business they want.

Another form of noise is the utterance of filler words such as "ah" and "umm" by users; these are usually omitted automatically by the dialog system.

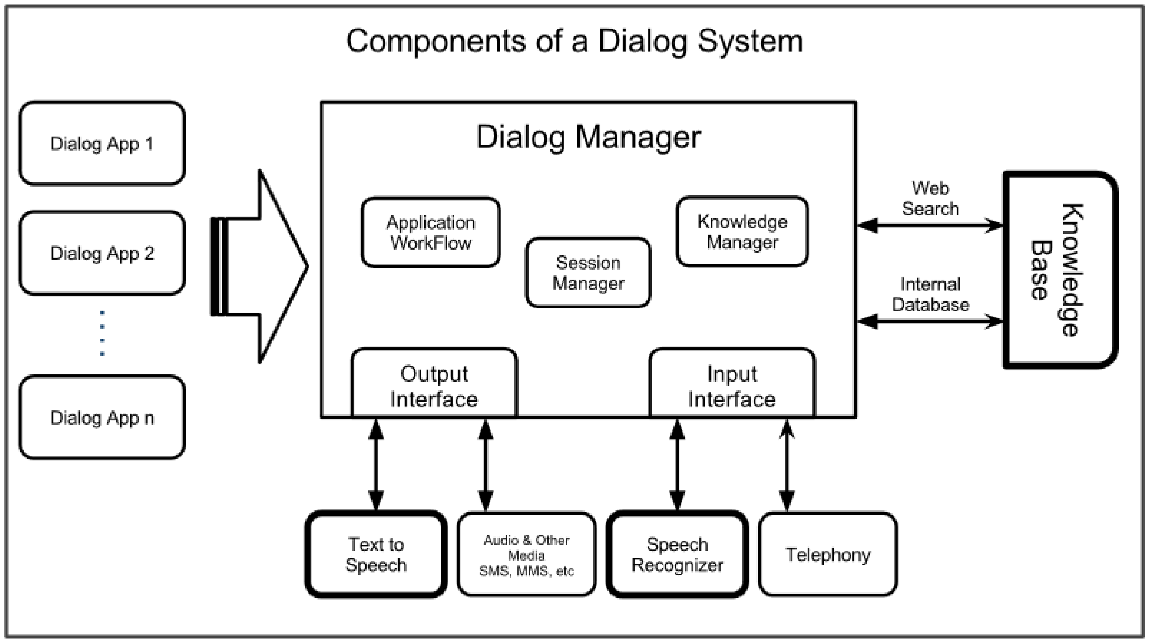


Figure 1 Components of a Dialog System

System response can be either synthesized speech or pre-recorded audio. While speech synthesis provides lots of flexibility to the system, recorded audio might be more feasible to generate a more humanly experience, especially for languages that still have poor text-to-speech support.

Figure 1 shows the main components of a dialog system, namely:

1. Dialog Manager
2. Speech Recognizer
3. Text to Speech Generator
4. Knowledge Base

At the core of the system is the *dialog manager*, which controls the process flow of the system. The dialog manager is responsible for receiving calls from users and maintaining call sessions. A user session holds information such as the current step the user is at and the user details that are either previously stored or collected from the user during the session. In addition, the dialog manager is responsible for interacting with the other modules of the system; the dialog manager would prompt the speech recognizer to start recording and try matching the user input against a defined grammar. It would also command the response generator to send pre-recorded messages or convert a certain output text into speech using Text-to-Speech. When needed, the dialog manager would request information from the knowledge base and process it to be presented to the user.

Spoken dialog systems naturally have client-server architecture, with clients having very limited computational capabilities. A client application might reside in the user’s machine to perform basic tasks such as audio recording and playback. A client might also submit a list of tasks to be executed by the dialog server, which is responsible for allocating resources to these tasks and finish them in a timely manner.

In most dialog applications, the user input is limited to a small set of vocabulary and sentences. A grammar can be used to define the set of allowed inputs. It is then the responsibility of the speech recognizer to inform the dialog manager whether the user input matches the defined grammar. The recognizer will also inform the dialog manager when it isn’t able to find any matches allowing the dialog manager to re-prompt for user input. Depending on the size of the provided grammar, the speech recognizer can be optimized to achieve better accuracy and speed than general purpose ASR systems.

Response is sent to the user as either generated speech using a speech synthesizer or as pre-recorded audio. In situations where the responses are a small set of sentences, it would be more preferable to record the audio responses beforehand to achieve better voice quality and to reduce computation overhead. However, in situations where the responses are generated in real time, it is necessary to utilize a good text-to-speech module to generate responses. It is also possible to have an intermediate solution where parts of the response are pre-recorded while the rest are synthesized in either word or phonetic level. For example, in a billing system where the users inquire for numeric responses, the system can generate responses by combining pre-recorded messages such as “Your balance is …” with a synthesized numeric values that are a combination of pre-recorded words.

Dialog systems can acquire information from various types of sources. Information can be stored in a local database or gathered from external sources. The growth of the World Wide Web and the movement towards service oriented architectures (SOA) and open APIs have created lots of deliverable content and information that can be presented to the user. Information about weather, stock market, real time news, social events, etc. are all accessible to end users nowadays.

1.1.2 DESIGN CHALLENGES

Lots of challenges face the developers of a spoken dialog system. Glass [1] breaks them down into three categories: Dialog design, Spoken language, and development issues.

Dialog design involves studying the complexity of human dialogs in order to be able to build dialogs that are humanly acceptable. Studies in this field show that many of human queries use a small subset of words and relatively short sentences [1]. It is also important to be able to handle dialog phenomena such as pauses, partial words, and filler words. One important feature is to allow users to barge-in, i.e., to interrupt the system output to provide their input. Systems that allow users to barge-in are said to provide a *mixed initiative* approach, in contrary to *system driven* approach where the user is forced to follow the system instruction. It is also important to be able to recover gracefully from errors, such as misinterpreted input. Users should be able to correct their input in case of wrong interpretation. This can be achieved by confirming user input before performing important transactions, such as checking out, and allow them to correct their input. Error correction should be non-linear; a user who wants to correct their credit card info shouldn't be forced to repeat the whole purchasing process.

Spoken language issues involve issues of speech recognition, speech synthesis and language understanding.

A speech recognizer for a dialog system is subject to the input channel conditions. Recognition over telephony is affected by various sources of noise that affect the quality of the recognized utterances. The recognizer has to be capable of adapting to individual users; knowing the user identity beforehand can help increase the user adaptation. Otherwise, short term adaptation techniques are needed to quickly adapt to the speaker accent and talking speed. Finally, speech recognizers for a spoken dialog system would benefit from techniques for detecting and adding out of vocabulary words dynamically to the system.

A speech synthesizer must be capable of producing human like sounds. In situation where this is difficult, pre-recorded audio can be used. Prosody can be used to improve the quality of generated output.

Language understanding involves issues related to natural language processing. Context-free grammar can be used to define sets of acceptable inputs. It is also possible to use keyword spotting to parse the user input. It might be necessary to perform a full linguistic analysis to understand the user input, but the real time constraint of a spoken dialog system restricts such approach.

Development issues involve the ability to collect the necessary data for the spoken dialog system, as well as issues for portability and integration with other system components.

1.2 VOICEXML

The success of voice based applications has inspired industry leaders to collaborate into building a unified standard for building voice based applications. The result of these efforts was the introduction of VoiceXML [4]. VoiceXML is an XML based standard for building voice enabled applications. VoiceXML is inspired by HTML standard used for web based applications; an application capable of processing VoiceXML documents is called a voice browser, similar to a web browser. VoiceXML allows for client side scripting using JavaScript, giving developers a simple and fast way to add front-end programming logic to their applications. At the backend, a document server is responsible for handling user requests and providing the required data in a comprehensible format.

The HTML standard has advanced a lot since the time the first version of VoiceXML was introduced. The upcoming HTML 5 [5] standard adds lots of features to HTML that were only achievable using 3rd party plug-ins. For instance, HTML5 supports Scalable Vector Graphics (SVG) [6] objects, which were developed around the time VoiceXML was introduced. SVG is an XML-based standard that allow vector shapes to be embedded into an HTML document. JavaScript can then be used to manipulate these objects freely. In addition, two new tags will be added to HTML to support audio and video files, enabling HTML to natively provide features that were only doable using plug-ins such as Flash and Silverlight.

The movement toward HTML5 support is backed by technology giants such as Microsoft, Google, and Apple. Support for HTML5 is also a main feature of modern mobile phones such as the iPhone and Android based phones, unifying the development efforts for both PCs and mobile phones to a large extent.

In addition to SVG, HTML5 provides extra features that shift the web paradigm from "web pages" and more into "web applications". This shift, along with the commonalities between VoiceXML and HTML, serve as motivations to seek opportunities to merge these technologies and have the next generation web browsers capable of natively handling speech in a way similar to what's being done to images, audio, and video files.

Research-wise, there are efforts to study the implications of voice based applications for health information, especially for developing countries [7]. Studies for merging VoiceXML with web based applications are shown in [8] and [9]. Efforts for localizing dialog systems to other languages, such as Slavic languages [10] and Chinese [11], are also currently active. A recent research [12] is investigating the ability of adding user adaptivity and semantic analysis as a higher layer in VoiceXML based dialog systems.

Specifications of VoiceXML started in 1999 by a joint effort of Motorola, AT&T, IBM and Lucent under the name of VoiceXML forum. Version 1.0 of VoiceXML was release in March 2000. [13] After releasing the first version of the standard, it was transferred to W3C to start working on a second version of the standard, which was then released in 2004. Many of the standard implementations have added extra features that weren't defined in the standard. In response to that, an update of the standard was released as version 2.1, which has become a recommended version in 2007. Work is now in progress on the third version of the standard.

VoiceXML is Unicode based. However, most of the implementations focus on English language only. From the surveyed products, only one product provided support to some European languages such as German, French, and Spanish.

VoiceXML applications follow the 3-tier architecture as illustrated in Figure 2. At the top tier, a Document Server accepts requests made from a VoiceXML interpreter. The document server processes these requests and generates responses as VoiceXML documents that can be processed by the interpreter. The interpreter resides in its own context. A context complements the interpreter by monitoring the application flow and providing extra controls to the user that are not be explicitly defined in the VoiceXML application. For example, the context might give the users the ability to change the voice volume if they are not able to hear the messages clearly. The implementation platform is the layer where tasks such as speech recognition and synthesis are performed. The interpreter and its context control the implementation platform by capturing events generated by it and issuing commands to its components to perform certain tasks.

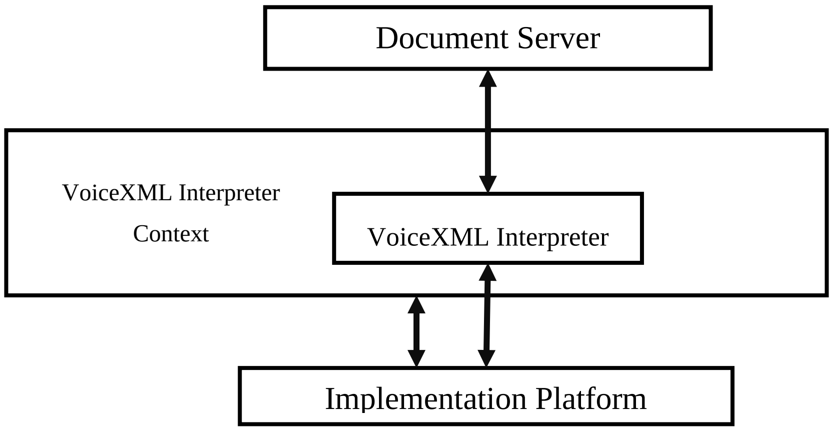


Figure 2 VoiceXML Architecture

A voice interpreter along with its context is also called a *voice browser* [14] [15]. Depending on the system design, a voice browser might exist on the same machine where the implementation platform is provided or every tier can be running on a separate machine.

W3C defines a set of standards that accompany VoiceXML to interface between the system components, namely:

1. Speech Recognition Grammar Specification (SRGS) [16], which is used to define the input grammar.
2. Semantic Interpretation for Speech Recognition (SISR) [17], allowing grammar tokens to be assigned to semantic interpretations.
3. Speech Synthesis Markup Language (SSML) [18], which provides a global interface to interact with speech synthesis applications.
4. Pronunciation Lexicon Specification (PLS) [19], which allows for specifying phonetic dictionaries that are both helpful for speech synthesis and recognition tasks.
5. Call Control eXtensible Markup Language (CCXML) [20], which is used to define the flow of how calls are received, processed, transferred, and so on.

This set of XML based standards provides a unified approach for a complete framework for handling automated speech conversation.

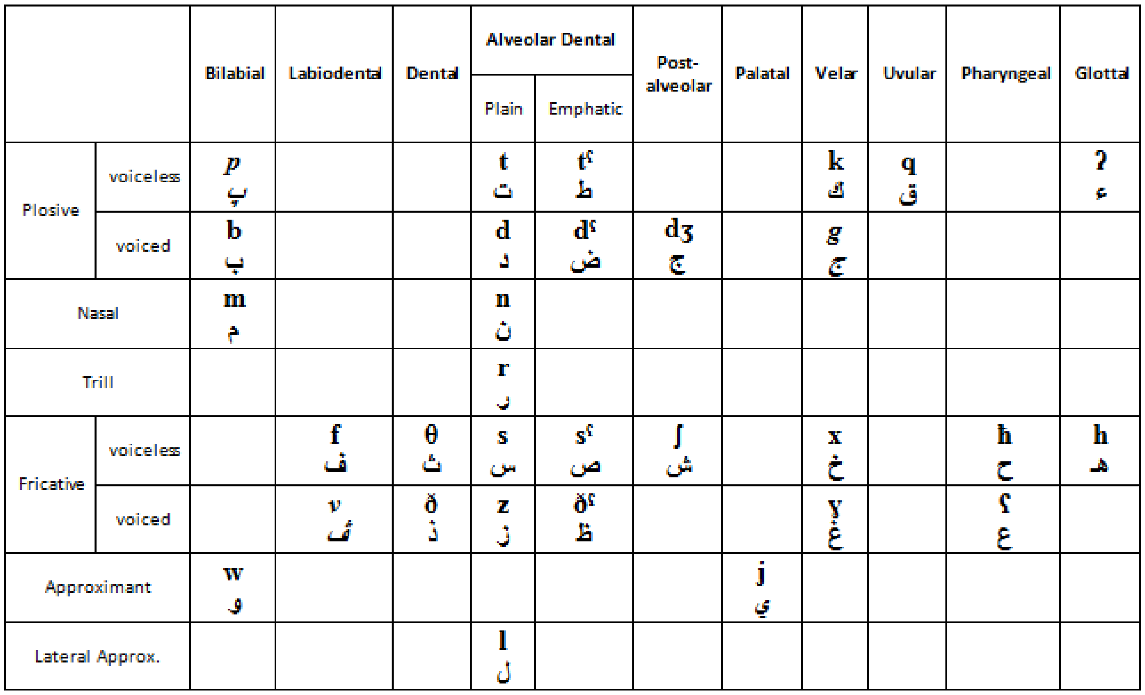
Further details of the VoiceXML standard will be discussed in chapter 3.

1.3 ARABIC LANGUAGE FEATURES

Arabic has a unique set of features that are not available in other languages. These features require special handling for the task of Arabic speech recognition. It is necessary for developers of Arabic speech recognition systems to be aware of the techniques and models necessary to successfully process spoken Arabic Input.

Figure 3 shows the Arabic consonants. Standard Arabic has a total of 28 consonants. Compared to English, Arabic has a set of extra phonemes: the emphatic and pharyngeal consonants, in addition to /ɣ/, /x/ and /q/ .The phonemes /p/ and /v/ are generally absent from standard Arabic, however, they might be used in pronouncing foreign names. In Egyptian dialect, the phoneme /dʒ/ is replaced with a /g/; and /q/ is replaced with a /ʔ/ in most of the cases. Some Gulf dialects replace /q/ with /g/. The Emphatic /l/ only occurs in the word (Allah).

Arabic has 6 vowels: three short vowels (/a/, /i/, and /o/) and three long vowels (/a:/, /i:/, and /o:/). In addition, Arabic has two semi-vowels: /w/ and /y/, that can generate two diphthongs: /ay/ and /aw/, if they follow the phoneme /a/, as found in the words "صيف" and "خوف". The emphatic and pharyngeal consonants affect the surrounding vowels and it is suggested to mark those vowels with a separate phoneme in text-to-speech as well as in speech recognition systems [21].



		Bilabial	Labiodental	Dental	Alveolar Dental		Post-alveolar	Palatal	Velar	Uvular	Pharyngeal	Glottal
					Plain	Emphatic						
Plosive	voiceless	p پ			t ت	tˤ ط			k ك	q ق		ʔ ء
	voiced	b ب			d د	dˤ ض	dʒ ج		g ج			
Nasal		m م			n ن							
Trill					r ر							
Fricative	voiceless		f ف	θ ث	s س	sˤ ص	ʃ ش		x خ		ħ ح	h هـ
	voiced		v ڤ	ð ذ	z ز	ðˤ ظ			ɣ غ		ʕ ع	
Approximant		w و						j ي				
Lateral Approx.					l ل							

Figure 3 Arabic Consonants

Nasalized vowels (known in Arabic as Tanween "تنوين") occur at the end of words only, and are usually omitted if the speaker stops on that word. Nasalization is performed by combining the /n/ sound to the standard vowels, resulting in: /an/, /in/ and /on/ respectively.

Vowels are represented in text as diacritic marks, and are usually omitted from most of the written text except for important and sacred texts like the Quran, the Arabic Bible, and international documents. Partial diacritization might be used to reduce ambiguity of written text. The lack of vowels in Arabic text is one of the main problems in Arabic language processing. Both text-to-speech and speech recognition systems require some sort of diacritization for undiacritized text.

Another major issue in Arabic is the huge number of affixes that can be combined with words. Prefixes such as (فَ, بَ, and وَ) are too short and are harder to recognize. The lack of diacritizaion plus the various possible affixations increase the ambiguity for AASR systems. Detailed morphological analysis of Arabic text is needed to overcome these issues. [22] [23]

In this research we focus on **Modern Standard Arabic (MSA)** only. Similar approaches can be developed for individual dialects. In order to reduce ambiguity, all Arabic text must be **fully diacritized**. This is true for both speech recognition and text-to-speech processing.

While research and industrial development for English speech applications has been going for a long while now, resulting in very high accuracy rates for speech recognition and articulate text to speech engines, research for Arabic speech is still a widely open domain with lots of areas to investigate. In this research work, we try to attract more attention to the importance of this field, and open up more opportunities for development by providing an open web-based framework that other researchers and developers can use and extend.

1.4 THESIS OBJECTIVES

The main objective of this thesis is to develop a framework for building web-based Arabic Dialog systems. In order to achieve this goal, the following steps have to be performed:

1. A study of VoiceXML standard, architecture, implementations and accompanying standard. This includes:
    1. A thorough study of W3C specifications for VoiceXML and related standards.
    2. A survey of both industrial and academic implementations of VoiceXML.
    3. Requirement Specification for Arabic support in VoiceXML.
2. Development of Models, Grammars and Dictionaries for Arabic speech processing, which includes:
    1. Training and maintenance of Arabic acoustic models for various problem sizes.
    2. Development of context-free Grammar for Arabic common dialogs.
    3. Generation of Phonetic Dictionaries for most common Arabic words.
3. Development of A web-framework for building and testing VoiceXML applications, which includes:
    1. A framework to create and Store VoiceXML files.
    2. A web-based front end for testing the applications.
    3. Development of a test application for Tadawul service, which includes:
        a. Defining user-interaction scenarios and use-cases.
        b. Development of the required VoiceXML documents.
        C. Expanding the front-end to support charting features.

1.5 RESEARCH CONTRIBUTIONS

The main contributions of this work can be summarized as:

1. A framework Design for web based spoken dialog systems.

2. A study of adding Arabic support to VoiceXML, the requirements and methods of providing them.
3. Demonstration of the capabilities of the framework by implementing a stock market navigation application.

CHAPTER 2

RELATED WORK

2.1 ARABIC SPEECH RECOGNITION

There has been lots of recent research and development for Arabic Speech Recognition. Efforts have been made to develop language and acoustic models, as well as techniques to build phonetic dictionaries. Al-Otaibi [24] proposed in 2001 an early Arabic speech dataset to be used for processing Modern Standard Arabic. He proposed a new technique for labeling Arabic Speech and reported a recognition rate of 93.78% for continuous Arabic speech.

In 2002, a workshop was held in John-Hopkins University [25] to address challenges in developing speech recognition systems for conversational speech over the telephone in Egyptian dialect. They proposed Romanization of Arabic text to increase readability by non-Arab researchers and reduce encoding issues. Billa et al. [26] addressed indexing of Arabic news broadcast along with a number of research issues regarding AASR. El-Ramly et al. [27] studied recognition of isolated Arabic phonemes using Time Delay Neural Networks and reported an accuracy of 91% in the best case.

Al-Otaibi [28] reported achieving 94.5% accuracy for speaker independent Arabic digit recognition using Recurrent Networks. A. M. Ahmad et al. [29] tested an isolated alphabet recognition system using RNN with various parameter settings and two feature extraction methods (LPC and MFCC). They reported that the best performance was achieved by using LPC and 50 fully connected nodes in the hidden layer.

Bahi and Sellami [30] investigated using vector quantization and Hidden Markov Models for recognition of Arabic isolated digits, and reported an accuracy of 93% in the best case. They have also investigated hybrid Neural Networks and Hidden Markov models for speech recognition [31]. M. Shoaib et al. [32] used a hybrid Power Spectral Densities (PSD) and Concurrent Self Organizing Maps (CSOM) system and tested on 28 phonemes with 100 samples for each phoneme. They achieved an overall accuracy of 90%.

Efforts were also made to develop hardware that is capable of Arabic speech recognition. Elmisery et al. [33] developed a hardware/software co-designed Arabic speech recognition system that aims to increase recognition speed of HMMs. They reported an accuracy of 98% on isolated Arabic digits and a 6 times increase in recognition speed. Zhuo et al. [34] described IBM’s hand-held speech to speech recognition system, aimed at achieving real time results on systems with limited resources.

M. Nofal et al. [35] studied the effect of automatic segmentation of the training data on enhancing the recognition quality. The problem of automatic segmentation was also studied in [36].

Further research in Arabic morphology was performed by Krichhoff et al. [23]. They represented four different approaches for Arabic language modeling and introduced a novel technique called factored language models. Xiang et al. [37] investigated algorithms to separate words and affixes in language models. Affify et al. [22] proposed a word decomposition morphological language model to improve recognition rates for Iraqi dialect.

Messaoudi et al. [38] investigated the problem of generating phonetic dictionaries and the effect of using morphological rules to generate pronunciations for huge databases of more than 1 million words. Gales et al. [39] have also studied the problem of generating phonetic dictionaries, while focusing on the effect of multiple pronunciations on recognition quality. Their research emphasizes on the inclusion of unsupervised training data as a way to improve the overall system accuracy. An enhancement to that effort was done in [40] where a multi-phase pronunciation generation is performed, with expert rules that cover cases that can't be captured with morphological analyzers.

Due to the significant increase in available Arabic speech data, recent research on developing complete AASR systems has become significant, with efforts from IBM [41] and CMU/Interact group [42]. Both projects are parts of the GALE program [43] supported by DARPA. Both papers highlight the importance of speaker adaptation in improving recognition quality.

Recently, Kosayba et al. [44] investigated the ways to implement Arabic support in VoiceXML applications. However, their basic assumption that VoiceXML doesn't support Arabic text has been found to be invalid. It is completely possible to use Arabic Unicode text in xml documents and therefore there is no need to Romanize Arabic text before processing it.

Alternative ways to build language models for Arabic where also investigated. Emami et al. [45], studied building continuous language models using Neural Networks. While Afify et al. [46] suggested using Gaussian mixtures to solve two problems with language models: the ability to generalize a model to account for unseen structures; and the ability to quickly adapt to newly added input to the model.

Research in Arabic Text to Speech was pioneered by El-Shafei et al. [21], where they developed a full set of rules to convert a fully diacritized Arabic text into synthesized speech. Basic text-to-speech generation depends on the morphological information in terms of letters, syllables and punctuation marks. In order to achieve higher quality of text-to-speech generation, further prosodic and semantic information need to be incorporated to the generator. Work for prosodic analysis for Arabic text-to-speech can be found in [47] and [40].

2.2 VOICEXML

2.2.1 APPLICATIONS OF VOICEXML

The commercial success of VoiceXML based IVR systems has lead researchers to investigate applying VoiceXML to emerging technologies such as e-learning and e-commerce. Researchers focus on design issues of such systems, as well as methods to make them cost-effective and affordable.

Kolias et al. [48] proposed a framework to add voice support to an existing e-learning system. They analyze the various modules of such system: lecture notes, wiki, forum and chat, and studied the scenarios where students can access these modules from telephony. Special attention was paid to the personalization of the student phone interface. Once authenticated over the phone, students can have quick access to material they recently accessed from the web interface. Personalization is needed to solve the problem of accessibility. An e-learning system contains huge datasets; however, individual students are most likely interested in material related to their classes only. It is also most likely that a student wants to access recently accessed material rather than older ones.

Azeta et al. [49] focuses on the problem of providing e-learning to the visually impaired. Students can access the system via phone or VoIP to ask specific questions. These questions are matched against a knowledge base. If a match is found, then an audio reply will be played for the student. Otherwise, the system goes into a process of tokenization and stemming to try to find a matching answer. If now answer could be found, the question will be forwarded to teachers where they can answer the question via email.

Long et al. [50] studied the various strategies to apply caching to a VoiceXML based e-learning system. Caching is required to reduce the bandwidth, and hence cost, of development of voice based applications, allowing economically constrained countries to benefit from such technology.

Applications to the e-commerce have been also investigated. Wenyun [51] discussed basic dialog design for a price inquiry system. Tsai [52] discussed the development of a multimodal web based application for food ordering for Mandarin language. The system data is stored in XML, which can be transformed into either HTML or VXML using proper XSL stylesheets, allowing faster development of these applications. It is then the task of the system designer to be able to do proper conversion using well developed stylesheets.

Sharp et al. [53] discussed the details of a framework for a recommendation system that can be used for applications such as restaurant search. In their research they investigated two techniques for building the recommended list: information based, where the recommendation is built on user profiles that contain info about their location, likes and interests, and collaboration based, where the recommendation is based on the wisdom of the crowd. The structure of the framework goes into 4 tiers; one client tier and three server tiers: voice processing and interpretation, application tier, and data management tier. The voice application is capable of accessing the recommendation data by interacting with a query manager that is capable of converting requests initiated by the user into search commands and return the results as a list of recommendations.

Eccher et al. [54] discussed the details of a multimodal website for filling patients' medical forms. Patients were able to use a web browser or a telephone to provide their information. In order to keep data in sync, a TCP/IP channel was used to update the VoiceXML session with info provided using the other input method. Armaroli et al. [55] discuss a framework and applications of similar natures, while Shao et al. [56] proposed the use of an intermediate language to automatically convert HTML documents to VoiceXML.

Reusch et al. [57] discussed the ability to convert user guides into voice enabled applications. Guides are usually organized into sections and sub-sections, indexed by tables of contents. In their framework, the indexes are converted to VoiceXML menus that can be navigated by the users, while section contents are read out by the system. To reduce the system overhead, each guide chapter is considered as a separate VXML document with the menu items being the input grammar.

Other applications were also developed based on VoiceXML. Kurkovsky et al. [58] [9] developed a framework for providing voice access to social networks using modern web services as a source of data access. Lian et al. [59] proposed an approach to convert geographic data, stored in an XML format, into VoiceXML documents, allowing users to search for destinations by phone. Kwon et al. [60] proposed an application to allow users to navigate RSS feeds using generic commands. In their application, a user is able to list the most recent updates from a news source, and then choose a certain story that gets read to the user using synthesized audio. Tsai et al. [61] proposed a mobile based application for tourists to be able to search for information and get responses via SMS messages.

The proposed systems point to many possibilities for interactive voice applications in many vital sectors: health, commerce, education. As part of the effort to develop a framework for Arabic voice applications, it would be necessary to investigate the requirements to build such applications.

2.2.2 FRAMEWORK DESIGN & EVALUATION

In addition to investigating particular domains and figuring out possible scenarios of using VoiceXML technologies, researchers investigated issues that affect the design and development of voice enabled application frameworks. Lin et al. [62] proposed a framework for web accessibility, where the users are able to retrieve parts of a web page using voice commands. In order to achieve accessibility, the HTML pages of a web site must follow a certain template that makes clear distinction between site block such as menus, banners, audio, headers and footers. While the requirement of following a certain HTML template is a big limitation for this framework, the recently expanding HTML5 [5] standard provides a set of new tags that make it easier to distinguish the prominent page blocks, which increases the applicability of Lin's approach.

Maes [63] addressed the issue of reusability in VoiceXML applications. Reusability is a highly required feature by developers to rapidly develop their applications. W3C has issued a technical report [64] detailing formatting and organization requirements to achieve reusability for voice applications. Maes's work expands on that specification to provide a "Java beans" like structure to define server side reusable dialogs. Two limitations are also addressed: the need to be able to dynamically define the grammar instead of it being statically embedded in current voice applications; and the ability to pass audio as a variable, allowing for customization to the developer needs.

Pietquin [65] introduced machine learning techniques to improve the dialog strategy of a VoiceXML application. According to Pietquin, Dialog can be expressed as Markov Decision Processes (MDPs), which extend Markov Chains to add a set of finite actions and a reward function that maps the transition between two states using a certain action to an immediate gain. The goal of MPDs is to find a decision strategy that maximizes the cumulative gain. In VoiceXML domain, states can represent the various call session states, which reflect the amount of information collected so far and the current operation that the user is at. Actions represent the various actions that can be performed by dialog manager, such as greeting the user, asking questions, confirming requests, presenting data and closing the conversation. The reward function can measure the number of steps needed to reach a final state. The faster a final state is reached, the faster the users were able to achieve their goal. However, this estimate is not an accurate measure of user satisfaction in a real life application, where other metrics, such as actual surveying of user feedback, can be used to evaluate the dialog.

Oria [66] discusses extending Mobile Application Programming Interface (MAPI), to support voice based interfaces. MAPI is a subset of HTML that is designed to work on mobile phones, inducing limitations to user input and display area. By adding voice support to the API, it becomes easier to extend voice support to business level applications originally designed for mobile phones.

Kong [67] proposed a framework to automatically convert HTML forms into VoiceXML applications. This is achieved by matching HTML elements to two main elements of a VXML document: *form* and *menu*. HTML elements that ask for user to provide textual input can be translated into VXML form prompts, while elements that provide users with choice (Menus, Radio groups, and checklists) can be converted to VXML menus.

Ruiz et al. [68] discussed the issue of the cost of developing a commercial quality VoiceXML gateway. In a commercial setting, special voice enabled hardware is used to receive client calls and process their requests. However, this can prove costly for lower economies. Ruiz then proposes a compilation of components and API that can be used to develop a low cost VoiceXML solution. It is however important to note that the recent improvements in cloud based computing and the development of high quality, open source solutions, as discussed later in this thesis, could be much more cost-effective alternative.

Various efforts to add multi-lingual support to VoiceXML applications are noted. Brkić et al. [10] discussed the efforts needed to provide support for Slavic languages, especially Croatian. Meng et al. [69] discussed the use of a bilingual Chinese-English voice interface that utilizies the “lang” tag in XML. Users are first prompted to choose their input language. Based on the chosen language, different tools and models are loaded at the back end.

2.2.3 COMMERCIAL FRAMEWORKS

VoiceXML separates dialog design from low level implementation, allowing companies to provide their speech technologies as a service for developers. This approach is being applied in many commercial products such as Nuance’s BeVocal Café[1], Microsoft’s TellMe Studio[2] and Voxeo’s Evolution[3]. These products share the same framework: users are provided with a web interface to develop and verify their VoiceXML documents. Users can test their application by calling a dedicated phone number or, in some cases, use VoIP applications such as Skype. Web based tools are provided to the users to allow them to build grammar, record audio, and design the dialog flow.

[1] http://cafe.bevocal.com
[2] http://studio.tellme.com
[3] https://evolution.voxeo.com

Most commercial products target English primarily. Multi-lingual support is provided by Voxeo for many European languages and partially for Mandarin. None of the surveyed products provided any Arabic support.

IBM released a tool set for the design and development of VoiceXML applications as an Eclipse plugin. These tools allow developers to visually design dialogs and grammars. The toolset al.lows developers to create their own voices that can be incorporated into VoiceXML applications. An API for data access via web services is also provided. Developers can then publish their applications to an OpenVXML based server.

CHAPTER 3

BACKGROUND

3.1 VOICEXML

VoiceXML is an XML based standard for defining audio dialogs. XML is widely used for data exchange, with a flexible structure to support arbitrary data. XML documents can be easily verified against a specified schema to detect corruption and avoid misinterpretation of shared data. XML documents are easily readable by humans, contrary to other formats such as binary files that aren't readable by humans and delimiter separated text files that are harder to interpret and can't effectively represent complex data structures. XML is also platform independent; it doesn't suffer from line break and character encoding issues that usually affect cross platform document sharing, making it easier to exchange the data between different systems and different applications. XML interpreters are now common to all major programming languages, simplifying the task of developing XML applications. In some languages, XML documents are treated as first class citizens.

XML supports Unicode by nature, which simplifies the task for sharing and interpreting non-Latin documents. It is also possible to include multi lingual data in the same document by specifying the *lang* attribute of the elements, allowing developers to design their applications towards localization. XML goes an extra step by allowing the developers to define a locale for their elements. In the scope of speech processing, the language/locale pair can roughly refer to a dialect of a language.

The multi-lingual aspect of XML is beneficial for the design of a Spoken Dialog System. The system designer can focus on the overall application flow and have it applied to different languages or dialects. In traditional systems, every language might have its own sub system that follows a different workflow and uses a different set of tools, which increases the cost of maintaining and evolving the system.

3.1.1 THE VOICEXML STANDARD

VoiceXML builds upon the advantages of XML to provide a standard for dialog definition. The main goal is to utilize modern web architecture to build voice based web applications. Modern web applications follow the standard 3-tier architecture, separating data, logic and representation. VoiceXML allows developers to focus on designing the dialog flow without getting involved in the lower level implementation and resource management.

VoiceXML applications are organized as a set of VoiceXML **documents**. Each document consists of a set of dialog constructs, including **forms**, **menus** and other control elements. Application **state** is defined as the current active dialog. An application can have one **root** document and multiple child documents, each containing one or more dialogs.

VoiceXML constructs describe elements of the interaction between the user and the system in a dialog application, which include:

User Input: In telephony applications, users can either provide spoken input or use the dial tone (DTMF input) to enter information. The sets of acceptable user inputs are defined using **Grammar** definitions. A grammar defines the set of acceptable inputs for a certain dialog construct. A grammar is **active** when the system is listening for input that matches it. If the user input matches an active grammar, a set of associated actions will be executed. In case of speech input, a **speech recognizer** is used to analyze the user input and match it to the active grammars. Filler phrases such as “uh” and “um” are usually omitted from the recognition results.

Audio Recording: In many applications, such as voice mail, users can record voice messages that can be played back. It is not necessary in this case to use a speech recognizer to interpret the message. However, advanced voicemail applications can use speech recognition to provide a transcript of the message. Recorded audio can also be used to customize user applications by providing custom recorded prompts.

System Output: Depending on the application target and available resources, audio output can be provided as either pre-recorded voices or synthesized speech using text-to-speech (TTS) applications. While pre-recorded audio can produce higher quality output, it won’t be feasible for dynamic applications that have non-determined outcomes. The output of TTS applications can be customizable; users might want to increase the volume level or change the speed of the spoken words. It is also helpful for users when the TTS engine is able to spell out words and numbers, for example, be able to pronounce 2010 as both “two thousand and ten” and “two oh one oh”.

Dialog Flow: A **session** starts when the user begins interacting with the application and ends when the application terminates or the user disconnects. The interaction can be represented as a state machine. At each state, the application has one active dialog; each dialog can have multiple actions that can lead to different states. Elements from a root document are active during the application execution and can interrupt the actions of the active dialog. Dialogs can be either forms or menus. Forms are used to collect user input and store it into **form variables**. A form can contain multiple **fields**. Fields can be of different types and are used to fill the form variables. Fields are filled from user input using a **Form Interpretation Algorithm** (FIA), which iteratively visits unfilled form fields and fill it with user input. Menus present users with options to choose from. Each option can lead the user to a new dialog in the current document or in a different document. During the execution of an application, many exceptions can occur, interrupting the execution of the application. Exceptions include events such as not receiving input from the user or not being able to recognize user input. User **disconnect** is one of the important events to handle, allowing the server to handle user input after the call session has ended, which includes final cleanup and submitting user data for processing.

<u>Telephony:</u> VoiceXML provides telephony features for call management, such as call transfer and disconnect handling. These features are important when integrating with a call center manager. It is important for a web based setup to be able to handle connects and disconnects properly, however, call transfers might not fit in with that model.

A **sub-dialog** is a form of dialog that can be invoked from other dialogs. It is similar to a function call in procedural programming. After executing the sub-dialog, control is returned back to the caller dialog. Sub-dialogs can be used to build reusable components and libraries that can be shared between different applications.

While dialogs are mostly system driven, VoiceXML supports **mixed initiative** dialogs using **links**. A Link specifies a grammar that is active within the link's scope. If the user input matches the grammar, control is transferred to the link's destination. This is helpful in situations where the user wants, for example, to move back to the main application menu from any application state. Links can also be used for custom event handling such as help requests.

Client side scripting can be performed using **ECMA** script blocks, with **JavaScript** being the most common implementation. Client side scripting involves setting variable values. Event handling might also involve executing client side scripts to perform additional computations. Client script can access the document's object model (DOM) to retrieve application related information such as the last recognized result. VoiceXML 2.1 introduced the **data** element which allows fetching external XML data without leaving the XML document, client scripting can be used to access the DOM model of the data to assign it to document variables.

An **Implementation Platform,** a. k. a. an interpreter or voice browser, is a system that implements the specifications of the VoiceXML standard. Implementations vary in their support for the standard features; some would only support a subset of the features and/or add extra features that are thought to be missing. The additional features might find their way back into the standard if they prove to be useful. It is, however, necessary for any implementation platform to support the following requirements:

1- **Document acquisition:** The voice browser must be able to fetch documents addressed using HTTP URIs. The browser must provide a unique User-Agent that differentiates it from other browsers. The identifier must be in the format of name/version.

2- **Audio Output:** The browser must support audio playback and TTS. The browser should also be able to sequence TTS output. Certain audio formats, specified in the standard, must be supported by the platform at minimum. If the requested audio resource is not found, the implementation must throw a noresource exception.

3- **Audio Input:** The implementation must be able to recognize user input and record audio and be able to control the time interval within which the input can be accepted. If no user input is available then a *noresource* exception must be raised. The recognizer must be able to receive the recognition grammar dynamically. Two grammar formats must be supported by the recognizer, XML and Augmented-BNF formats of the Speech Recognition Grammar Specification (SRGS). Additional formats such as JSGF are optional.

4- **Transfer:** The implementation platform must be able to accept calls and make connections with other communication networks.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/vxml
http://www.w3.org/TR/voicexml20/vxml.xsd" version="2.0">
<form>
    <field name="meal">
        <prompt>Would do you like to eat?</prompt>
        <grammar src="meals.grxml" type="application/srgs+xml"/>
    </field>
    <block>
        <submit next="http://www.resturant.example.com/meal.asp"/>
    </block>
</form>
</vxml>
```

Figure 4 Sample VoiceXML document

3.1.2 DIALOG ELEMENTS

3.1.2.1 FORMS

The main components of VoiceXML documents are forms. A form consists of multiple input items in addition to control items and event handlers. All input items have three main attributes:

1- **Name:** The name of the variable associated with the input item.
2- **Expr:** An initial value for the item if needed.
3- **Cond:** A Boolean pre-condition for processing the input item by the FIA.

VoiceXML defines the following types of form input items:

1- **Field:** Used to acquire user input from DTMF or speech recognition.
2- **Record:** Used to record user audio message. An optional beeping sound can be issued to the user to start the recording process. Recording can be terminated using key presses or user disconnect. A maximum timer can also be specified.
3- **Object:** Used to call platform specific modules, similar to <object> tag in HTML. Parameter passing to the called object is performed using the <param> tag.
4- **Subdialog:** Used to call other dialogs. This is similar to a function call; control is transferred to the called dialog until it finishes execution and then it would return to the calling dialog afterwards.

In addition to input items, forms define two control items:

1- **Block:** Define a set of execution items that don't require user input. A block has an implicit variable that has its value set to true before the execution of the block.

2- **Initial:** Used to define elements that are executed in the first iteration of the FIA algorithm only. These elements are usually used to help users providing combined queries. For instance, in a weather forecasting application, an initial block could try to collect information about location and time of the forecast. If any of these variables get filled within the initial block, next iterations of the FIA algorithm will then try to ask the more specific questions specified in the remaining fields. Initial fields, combined with form level grammar, are used to provide mixed initiative dialogs.

Figure 4 [4] shows a sample VoiceXML form containing one field "drink" and an execution block to submit the user's choice. The field contains a **prompt** which generates TTS output and a **grammar** that is active in the scope of the field. It is possible to use **option** list instead of grammar to specify the possible field values. An internal prompt counter is used to count the number of times a certain prompt is issued. It is possible to provide multiple prompts based on the counter to try to provide a more concise message to the user. In addition to prompts and grammars, fields can have the following items:

1- **Filled:** Event handler for the filling of the field with an acceptable value.

2- **Catch handlers:** Exception handlers for thrown events. Shorthand tags are used for common exceptions such as <noinput>, <nomatch> and <help>.

FIA algorithm is performed on the active form element and goes through an initialization step followed by looping through form items to fill them. The loop stops when the form fields are filled or the control is transferred to another dialog.

1- **Initialization:** Forms are initialized every time they are entered. During initialization phase, all prompt counters are reset to 1 while field variables are set to undefined if no default value is explicitly set.
2- **Main Loop:** The main loop consists of three phases:
    a. Select phase: Sequentially select the next unfilled form item for processing. Pre-conditions are checked before selecting an item for processing. Various control items such as <goto> can be used to change the sequential behavior of the algorithm.
    b. Collect phase: Prompt users for input and collect their responses.
    c. Process phase: Process the user input and handle events and exceptions if necessary.

3.1.2.2 MENUS

A Menu is a form that contains one, anonymous field and has its grammar defined as a set of choices. Menu input is generally controlled using both DTMF input and speech. Menu choices are automatically assigned the DTMF inputs from 1 to 9.

An **enumerate** element can be used to loop through menu choices.

Figure 5 shows an example of such block. When processing the menu choices, the enumeration will generate a message similar to “For choice1 press 1; for choice2 press 2.” and so on.

```
<enumerate>
  For <value expr="_prompt"/>, press <value
  expr="_dtmf"/>.
</enumerate>
```

Figure 5 Enumerate Block

3.1.2.3 LINKS

Links are used to change the execution flow of a document. Links can exist at document level, form level, or field level. Links are associated with their own grammar and can be activated any time within their scope. E.g., a link at the document level can activate if any of the document dialogs are active. Links at the root document can activate at any stage of application execution. When activated, a link will transfer to a new dialog or throw an event.

3.1.2.4 EVENT HANDLING

Through the execution of the FIA algorithm, a *filled* event is activated whenever an input item is assigned a value. The <*filled*> element can be specified at form or field level to perform actions when that event occurs. Possible actions include checking the integrity of input data or performing additional computations. For example, the filled event can be used to check if the user has specified a valid date range, i.e. the start date is less than the end date. If the data is invalid, input fields can be reset so they are picked up again with next iterations of the FIA algorithm.

In addition to the “filled” event, the voice browser throws additional events that can be handled using a catch element:

1. **Error:** Thrown when an error occurs during the execution of the application. Among the possible errors are inaccessible resources and malformed VoiceXML documents.
2. **Help:** Implementation specific request for help. This can simply be the user saying the word “help” or dialing special DTMF sequence.
3. **Noinput:** Thrown if the system doesn’t receive an input from the user during a specified time frame.
4. **Nomatch:** Thrown if the user input doesn’t match any of the active grammars.

It is also possible to explicitly invoke events using a *<throw>* element. Thrown events can be system pre-defined events or application defined events, both can be handled by a matching <catch> element.

Shorthand elements can be used instead of the *<catch>* element to capture system events, namely: *<error>, <help>, <noinput>,* and *<nomatch>*.

3.1.3 INPUT GRAMMAR

Grammar is the key element for handling user input; it specifies the acceptable input from the user as well as the **semantic interpretation** for that input. In case of speech input, grammar specifies the set of utterances that the speech recognizer can accept, while in case of DTMF, grammar specifies the set of key presses to be matched by the interpreter.

The VoiceXML standard requires implementation platforms to support at least the grammar specifications detailed in the **Speech Recognition Grammar Specification (SRGS)**. SRGS grammar is **Context Free (CFG)**; a **rule** in the left hand side consists of a single non-terminal symbol while the right hand side consists of a sequence of terminal and non-terminal symbols. Other formats might also be supported by the implementation platform.

Grammar is usually passed to the speech recognizer to match it against the user input. The result of this process is called **raw text transcription** or **literal text**. Raw text is returned to the interpreter to generate the semantic interpretation, which could be equivalent to the user input or a simple key-pair matching, such as choosing the nth item of a menu.

In order to support a wide range of speech recognizers, SRGS comes with two forms for grammar definition:

1- **Augmented Backus-Naur Form (ABNF):** A non-XML, BNF based format derived from **JSpeech Grammar Format (JSGF),** which is supported by Java-based speech recognizers such as Sphinx4.

2- **XML Form:** An equivalent, XML based format derived from similar XML formats such as PibeBeach and TalkML [70].

Both forms are equivalent and provide the same semantic interpretation.

The *<grammar>* element is used to specify the input grammar. Grammar can be either **inline,** defined within the body of the grammar element, or **external,** referenced by a URI. Grammar **mode** can be either **DTMF** or **voice.** Grammar **type** specifies the mime-type of the grammar definition.

```
<grammar mode="voice" type="application/srgs">
#ABNF 1.0;
language en-US;
mode voice;
root $command;
  public $command = $action $object;
  $action = show | hide | close;
  $object = [the | a] (window | tab | menu);
</grammar>
```

Figure 6 Sample ABNF Grammar

```
<grammar mode="voice" xml:lang="en-US" version="1.0" root="command">
  <rule id="command" scope="public">
    <ruleref uri="#action"/> <ruleref uri="#object"/>
  </rule>
  <rule id="action">
    <one-of>       <item> show </item>       <item> hide </item>       <item> close </item>
    </one-of>
  </rule>
  <rule id="object">
   <item repeat="0-1">
      <one-of> <item> the </item> <item> a </item> </one-of>
    </item>
    <one-of>       <item> window </item>       <item> tab </item>       <item> menu </item>
    </one-of>
  </rule>
</grammar>
```

Figure 7 Equivalent XML Grammar

Figure 6 shows a sample ABNF grammar for constructing a command. A public rule for *command* is defined as public and is assigned as the root rule. A command is then defined to consist of an action and an object. An action can be either to show, hide, or close while the object can be either a window, tab or a menu, preceded by an optional article (a or the).

Figure 7 shows the same grammar defined using XML.

Version 2.1 of VoiceXML allows external grammars to be referenced **dynamically**, enabling dynamic loading of different input sets based on the user input. For example, when a user looking for weather information specifies their country, a grammar for the available cities can be dynamically loaded in order to fill the city field.

Grammar can be defined at different levels of the VoiceXML document. The scope of the grammar defines regions of the document where the grammar is active. Grammar can have one of the following scopes:

1- **Input item scope:** Grammars defined at the input item level, such as fields, are active when the input item is visited by the form interpretation algorithm. Input grammar cannot be scoped to outside of their input items.

2- **Dialog scope:** Grammars assigned to forms and menus have dialog scope by default; they are active when the user is in the form. Link grammar takes the scope of the link container by default. If the link is inside a form or menu then it will have dialog scope.

3- **Document scope:** Form and menu grammars can be assigned document scope; they will be active during the execution of the document. If the grammar is specified in the root document, then it will be active during the overall execution of the application.

Multiple grammars can be active at the same time. The set of active grammars is determined by the current active dialog and the scopes of the application dialogs. User input may match more than one active grammar, causing a grammar overlap. If an overlap occurs, precedence rules are applied to match the input to the highest priority grammar.

Weights can be assigned to grammars and grammar rules. Weights are used by speech recognizers to introduce bias to recognition results. The default value for weight is 1.0. The interpretation of the weight values is the responsibility of the speech recognizer. Therefore, it is preferred not to change the weight values unless there is a need for run-time tweaking of the recognition results.

3.1.4 SYSTEM OUTPUT

Audio output of VoiceXML applications is defined using **prompts**. The basic prompt element contains text to be passed to the TTS engine. Additional output features can be specified according to the **Speech Synthesis Markup Language (SSML)** specifications, which allow specifying output features such as phonetics, prosody, grouping, interpretation and audio source.

Phonetic pronunciation of the output can be specified using the <phoneme> tag, allowing developers to choose specific pronunciations to words that have multiple pronunciations or specify the correct pronunciation for words out of the scope of the TTS engine.

International Phonetic Alphabet (IPA) is the standard format for specifying the phonetic pronunciation. Engine specific formats can also be used, in which case the ***alphabet*** attribute of the phoneme tag must be set to the engine format.

Prosodic features of the spoken text can be set using a set of tags: <*voice*>, <*emphasis*>, <*break*>, and <*prosody*>. Voice tag allows choosing a speaker by name, gender, or age. Emphasis allows stressing on certain parts of the text. Break allows specifying pause points in text utterance. Prosody allows controlling features such as pitch, rate, duration and volume.

Text can be organized into paragraphs and sentences using the <*p*> and <*s*> tags respectively. The <*sub*> tag can be used to provide aliases to the contained text, such as abbreviation expansions.

Text interpretation allows specifying the correct way to pronounce the text. Dates, currencies and phone numbers are examples of text that can be pronounced in different ways. It is possible to use the <*say-as*> tag to specify the type of the enclosed text.

Audio playback within prompts is allowed using the <*audio*> tag. Audio source can be either an external URI or an audio stored in a variable using the <*record*> tag.

The <*value*> tag can be used to reference user interpreted input. This is useful especially for confirmation step of user transactions. The attribute can also be used for any type of processing of user input using ECMA scripts.

One important feature of prompts is to allow users to **barge-in**, i.e., interrupt the system and start providing input. This feature is controlled using the *bargein* attribute of prompts.

Finer details for the barge-in feature can be specified, such as allowed interrupt sources (audio, key press) and interrupt keywords.

Multiple prompts can be specified for a form item. A prompt **counter** is used to choose the appropriate prompt during each attempt to fill the item. Prompts can also be controlled using conditions. A **conditional prompt** will be executed only if its condition is matched.

3.2 SPEECH PROCESSING

In order to provide Arabic support, an implementation platform must provide Arabic processing capabilities for two main features:

1- **Speech Recognition:** The implementation must provide a speech recognizer capable of reading Arabic input grammars and matching them with user speech. The recognizer must support the grammar formats determined by the standard. The recognizer must be also capable of receiving input grammars on run time and processing user input accordingly.

2- **Text to Speech Engine:** The implementation must provide a text-to-speech engine capable of converting prompts to Arabic spoken audio. One or more Arabic voices are to be provided by the engine. Advanced features from the SSML standard are recommended.

Having these features supported, Arabic VoiceXML applications can be developed to contain the following:

1- **Arabic Grammar:** Either ABNF or XML based grammars that use Arabic literals for rule symbols.
2- **Arabic Prompts:** Arabic text that is converted by the TTS engine to Arabic speech.
3- **Arabic Menus:** Menus consisting of multiple choices that are written and spoken in Arabic.

3.2.1 SPEECH RECOGNITION

A speech recognizer is required that is capable of matching user input to the application active grammars.

Due to the interactive nature of VoiceXML applications, a **live decoder** is required to perform the recognition process. The recognizer should also be capable of receiving grammar dynamically during the application execution. Recognition model must be **speaker independent,** i.e., capable of recognizing input from male and female speakers from different ages.

Speech recognition is a process that consists of various steps:

1- **Audio Recording:** Audio is fetched from the audio source. In case of live recording from microphones, an **end-pointer** processor is needed to detect the start and end points of the audio segment.
2- **Pre-processing:** Various operations are performed on the recorded audio to reduce noise and split the audio input into smaller segments.

3- **Feature Extraction:** The audio signal is converted into a set of feature vectors. The most commonly used feature sets for speech recognition are **Mel-frequency cepstral coefficients** (**MFCC**) and **LPC** [71]**.**

4- **Decoding:** A search algorithm is used to match the input feature vectors against an **acoustic model** and a **language model**. Various search techniques are applied to generate best matching word sequences.

3.2.1.1 THE ACOUSTIC MODEL (HIDDEN MARKOV MODELS)

The acoustic model is a statistical representation of sound units. These units can be words, syllables, or phonemes, depending on the vocabulary size of the model. For limited vocabulary models, such as Arabic digits, it is possible to use the numeral utterances as the basis for the acoustic model. For very large vocabulary models, however, it is best to use phonemes as the basis for the acoustic model.

The most common technique for building acoustic models relies on **Hidden Markov Models (HMMs)** [71]**,** which estimate the likelihood of each phoneme at contiguous, small frames of the speech signal.

Each phoneme is modeled as a sequence of HMM states. In standard HMM-based systems, the likelihood (also known as the emission probability) of a certain frame observation being produced by a state is estimated using traditional **Gaussian Mixture Models** (GMMs). The use of HMM with Gaussian mixtures has several notable advantages such as having a rich mathematical framework, efficient learning and decoding algorithms, and an easy integration of multiple knowledge sources.

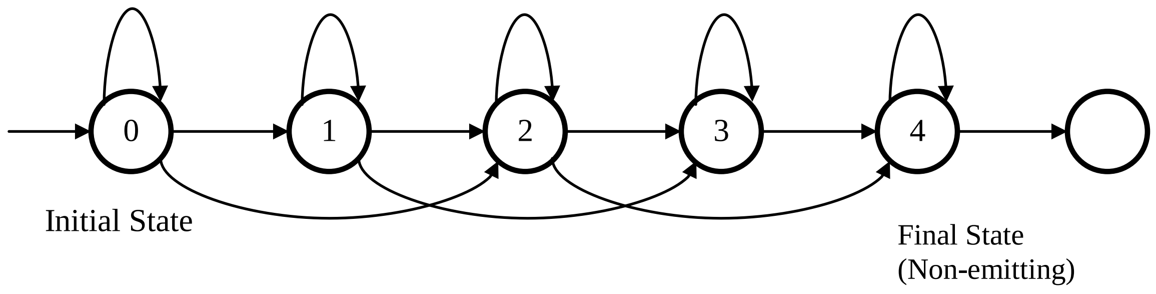


Figure 8 HMM Architecture

An HMM model, **λ**, is completely specified by the following parameters:

- The number of states *N*.
- The state transition probabilities, A, $a_{ij} = P(s_{t+1} = j \mid s_t = i)$ where s_t is the state at time *t*.
- The observation symbol probability, B, $b_j(x_t) = P(x_t \mid s_t = j)$ where x_t is the observation at time *t*.
- The initial state probabilities, Π, $\pi_i = P(s_1 = i)$.

It has been recognized that each phoneme is influenced in different degrees by its neighbors. HMM-based systems that are developed for continuous speech usually use *triphones*. A *triphone* still models only a single phoneme; yet the same phone surrounded by different left and right phones is captured using distinct models. For example, in a context where the phoneme /l/ is preceded by an /a/ and followed by an /x/, a separate model for the triphone /l(a,x)/ will be created. Triphones increase the complexity of the recognition task. Instead of building models for 30-40 phonemes, the number of triphones in a specific task could range from 7000 to 15000. Each HMM model uses different distributions for each HMM state, which means in a context with 7000 triphones and 5-state HMMs there will be around 35000 different distributions. In case of Gaussian mixtures, each distribution is modeled with up to 32 Gaussian models. So, overall, there are lots of computations needed to process each speech frame. Techniques that reduce the complexity of the model and improve processing speed are essential for the success of HMM based models.

In the case where Gaussian mixtures are used for computing the observation probabilities B, the probability of generating the observation xt given the transition state j, $P(x_t \mid j)$ becomes

$$b_j(x_t) = p(x_t \mid q_t = j) = \sum_{k=1}^{M} w_{j,k} N_{j,k}(x_t)$$

where $N_{j,k}$ is the k-th Gaussian distribution, w$_{j,k}$ are the mixture weights, and $\sum_k w_{j,k} = 1$.

The main drawback of Gaussian mixtures is the extremely large number of parameters needed to describe the Gaussian distributions.

In situations of less complexity, a simpler discrete model is used rather than Gaussian mixtures, namely Vector Quantization. A set of discrete vectors in the feature space are used to model feature distribution. Each chosen vector is called a codebook. The codebooks are used to map a feature vector x_t to one or more code vectors v_k. Then, the emission probability is approximated by

$$b_j(x_t) = \sum_{k=1}^{L} f_j(v_k) d(v_k, x_t)$$

where f is the distribution associated with the state j, and d is the distance between the code vector k and the data vector x_t. An example of distance function would be the Mahalanobis distance which is given by

$$d(v_k, x_t) = \frac{1}{(2\pi)^{n/2}} \exp(-0.5 * (x_t - v_k)^T \Pi_k^{-1} (x_t - v_k))$$

where Π is the covariance vector.

L is the number of nearest codebooks used to calculate the distance, which is user defined. The values of $f_j(v_k)$ can be pre-computed and stored in a lookup table before performing the recognition task.

There are two main algorithms that are frequently used white training and using HMM models: the Viterbi (Forward) algorithm, and the Baum-Welch algorithm (Forward Backward) algorithm. The following sub-sections explain the steps and computations included in both.

3.2.1.2 VITERBI ALGORITHM

The Viterbi algorithm is a dynamic programming algorithm that tries to find the highest scoring state sequence S = ($s_1, s_2, \ldots, s_T$) for a given observation sequence X = ($x_1, x_2, \ldots, x_T$)

in a given time frame t =1 to T. In other words, find S_{best} = $\arg\{max_S\ P(S|X)\}$, which is equal to $\arg\{\max_S \prod_{i=1,\ldots k} P(x_i|s_i, s_{i-1})\ \ P(s_i|s_{i-1})\}$. [24]

We define $\phi(t, j)$ to be the probability of the most likely partial state sequence or path until time frame t, ending at the ith state. We also define $U(t, i)$ to be the reference to the previous state that lead to a maximum score. The probabilities $a_{i,j}$ and b_j are the transition and observation probabilities per each HMM, respectively. The algorithm proceeds in the following steps:

<u>Step 1: Initialization:</u> For every state, s_j, $\phi(1, j)$ is computed as follows:

$$\phi(1, j) = a_{1,j} b_j (x_1)$$

<u>Step 2: Induction:</u> Starting from time frame t = 2, $\phi(t, j)$ and $U(t, i)$ are calculated as follows:

$$\phi(t, j) = \max_i \{\phi(t-1, i) a_{i,j}\} b_j (x_t)$$
$$\text{i} = 1,2,\ldots, N; \text{and } t = 2,3,..T$$

$$U(t, i) = \arg\{\max_j \{\phi(t-1, j) a_{i,j}\} b_j (x_t)\},$$
$$j = 1,2,\ldots, N; \text{and } t = 2,3,..M$$

<u>Step 3: Best Path:</u> The maximum likelihood of the best path is then given by:

$$P(X / Model) = \varphi(N, T) = \{\max_j \{\phi(N, j)\} \quad j = 1,2,\ldots, n_v (M)\}\}$$

$$U(M, i_{best}) = \arg\{\max_j \{\phi(M, j)\} \quad j = 1,2,\ldots, n_v (M)\}\}$$

<u>Step4: Backtracking:</u>

$$\begin{aligned} i_M &= i_{best} \\ i_{t-1} &= U(t, i_t); \text{ for } t = M, M-1, \ldots\ldots 2 \\ S &= s_{i1} s_{i2} \ldots\ldots\ldots s_{iM} \end{aligned}$$

Viterbi algorithm belongs to dynamic programming paradigm and is used for decoding the input vectors into phonemes and states. However, the dynamic programming assumption about best paths doesn't cover the more complicated task of splitting phonemes into words. For that purpose, more advanced algorithms are used. The most well-known algorithms to replace the Viterbi algorithm are the multiple-pass decoding, where the Viterbi algorithm returns the n-best sequences instead of only the highest scoring sequence, the n-best sequence is then fed to the language model to recalculate the score using techniques such as n-grams; and the A* (stack) algorithm. [71]

3.2.1.3 BAUM-WELCH ALGORITHM

Baum-Welch algorithm is used for training the model, i.e., computing transitional probabilities (A) and observational probabilities (B). The algorithm is iterative; it starts with an initial model λ and keeps improving it over time. [72] [73]

Let $L_j(t)$ denotes the probability of being in state j at time t, and let $\alpha_j(t)$ be the forward probability for some model λ with N states be defined as

$$\alpha_j(t) = P(x_1, x_2, \ldots x_t; s(t) = j \mid \lambda)$$

That is, $\alpha_j(t)$ is the joint probability of observing the first t speech vectors and being in state j at time t. This forward probability can be efficiently calculated by the following forward recursion

$$\alpha_1(1) = 1; \qquad \alpha_{\mathrm{j}}(1) = a_{1j} b_j(x_1)$$
$$\alpha_j(t) = \left[\sum_{i=2}^{N} \alpha_i(t-1) a_{ij} \right] b_j(x_t)$$

For $1 < j < N$, and the final condition is given by,

$$P(X \mid \lambda) = \alpha_N(T) = \sum_{i=2}^{N-1} \alpha_i(T) a_{iN}$$

where $P(X \mid \lambda)$ is the probability of the observation with the given HMM model.

The backward probability $\beta_j(t)$ is defined as

$$\beta_j(t) = P(x_T, x_{T-1}, \ldots x_t; s(t) = j \mid \lambda)$$

As in the forward case, this backward probability can be computed efficiently using the following backward recursion

$$\beta_i(T) = a_{iN};$$
$$\beta_i(t) = \sum_{j=2}^{N-1} \beta_j(t+1) a_{ij} b_j(x_{t+1})$$

And the final condition is given by

$$\beta_1(1) = \sum_{j=2}^{N-1} a_{1j} b_j(x_1) \beta_j(1)$$

From these definitions, we can compute the joint probability:

$$P(X, s(t) = j \mid \lambda) = \alpha_j(t)\beta_j(t)$$

hence,

$$L_j(t) = P(s(t) = j \mid X, \lambda) = \frac{P(X, s(t) = j \mid \lambda)}{P(X \mid \lambda)} = \frac{\alpha_j(t)\beta_j(t)}{P}$$

We also define $\zeta_t(i, j) = P(s(t) = i, s(t+1) = j \mid X, \lambda)$ is the probability of being in state *i* at time *t*, and state *j* at time *t*+1, given the model and the observations.

$$\zeta_t(i, j) = \frac{\alpha_j(t) a_{ij} b_j(x_{t+1}) \beta_j(t+1)}{P(X \mid \lambda)}$$

We now have all the variables needed to execute the Baum-Welch algorithm.

The algorithm proceeds iteratively, starting from an initial model λ. The steps in this algorithm may be summarized as follows:

Step 1: Calculate the forward and backward probabilities for all states *j* and times t.

Step 2: Update the parameters of the new model as follows

$$\bar{\pi}_j = \text{expected frequency (number of times) in state j at time t} = 1 = L_j(1)$$

$$\bar{a}_{ij} = \frac{\text{expected number of transition from state i to state j}}{\text{expected number of transitions from state i}} = \frac{\sum_{t=1}^{T-1} \zeta_t(i, j)}{\sum_{t=1}^{T-1} L_i(t)}$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in state j and observation symbole } v_k}{\text{expected number of times in state j}}$$

If the state the output distribution is a single component Gaussian, the parameters of the distribution can be found by

$$\bar{\mu}_j = \frac{\sum_{t=1}^{T} L_j(t) x_t}{\sum_{t=1}^{T} L_j(t)}; \qquad \bar{\Sigma}_{\mathrm{j}} = \frac{\sum_{t=1}^{T} L_j(t)(x_t - \bar{\mu}_j)(x_t - \bar{\mu}_j)'}{\sum_{t=1}^{T} L_j(t)}$$

Step 3: If the value of $P(X \mid \lambda)$ for this iteration is not higher than the value at the previous iteration then stop, otherwise repeat the above steps using the new re-estimated parameter values.

3.2.1.2 THE LANGUAGE MODEL

VoiceXML uses Context-Free Grammar to define the language context. Given a set of grammar rules, it is possible to build a search graph that defines all possible input sequences. The graph is used by the decoder to determine the possible outcomes. If the decoder is not able to reach a final state, then it must return to the interpreter with the proper error message (no input or no match).

Figure 9 shows the search graph for the sample grammar from Chapter 3. Silence nodes are added as start and end nodes. The grammar is matched if any of the paths from the start node to the end node is fully traversed by the spoken input. Each node is then assigned a value corresponding to the matched word, which is then returned to the interpreter.

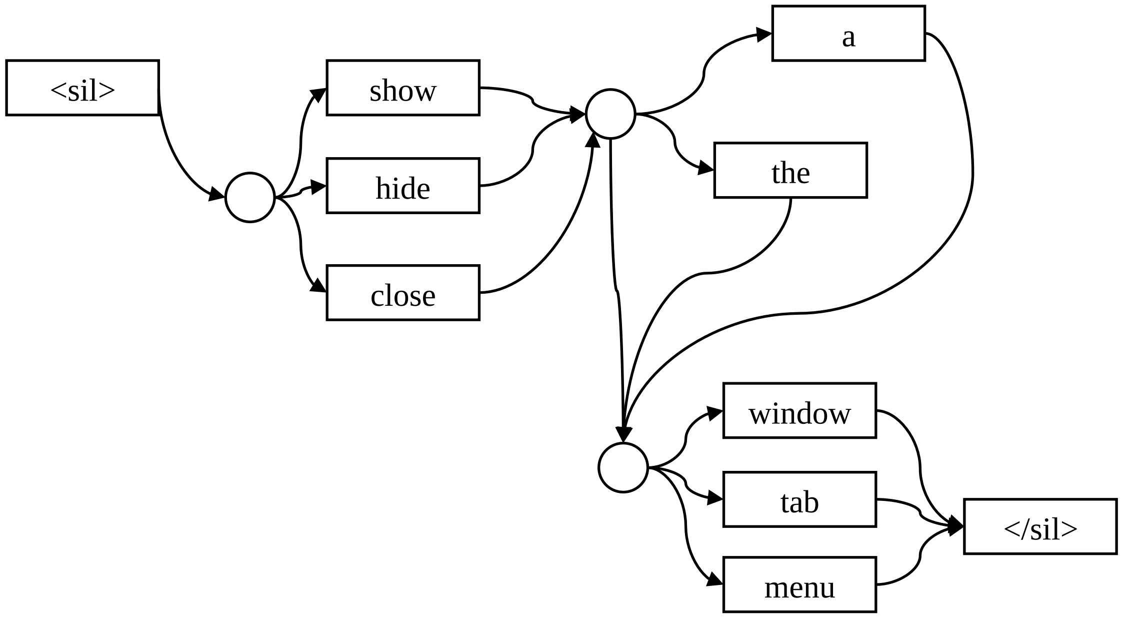


Figure 9 Grammar Search Graph

Weights provided in the grammar can be applied to search paths to increase the likelihood of certain paths against other paths. If no weights are provided, each branch will be considered equally likely.

Grammars are application specific; it is up to the application developers to build and specify the grammars they want to use. It is necessary, however, to update the phonetic dictionary to accommodate for all words used in the grammar.

3.2.2 TEXT TO SPEECH

TTS generation is a process based on **Natural Language Processing (NLP)** and **Digital Signal Processing (DSP)**. The text to speech module is responsible for receiving plain text from the interpreter and generating the audio output. Figure 10 shows the general framework for an Arabic TTS system [21] [44].

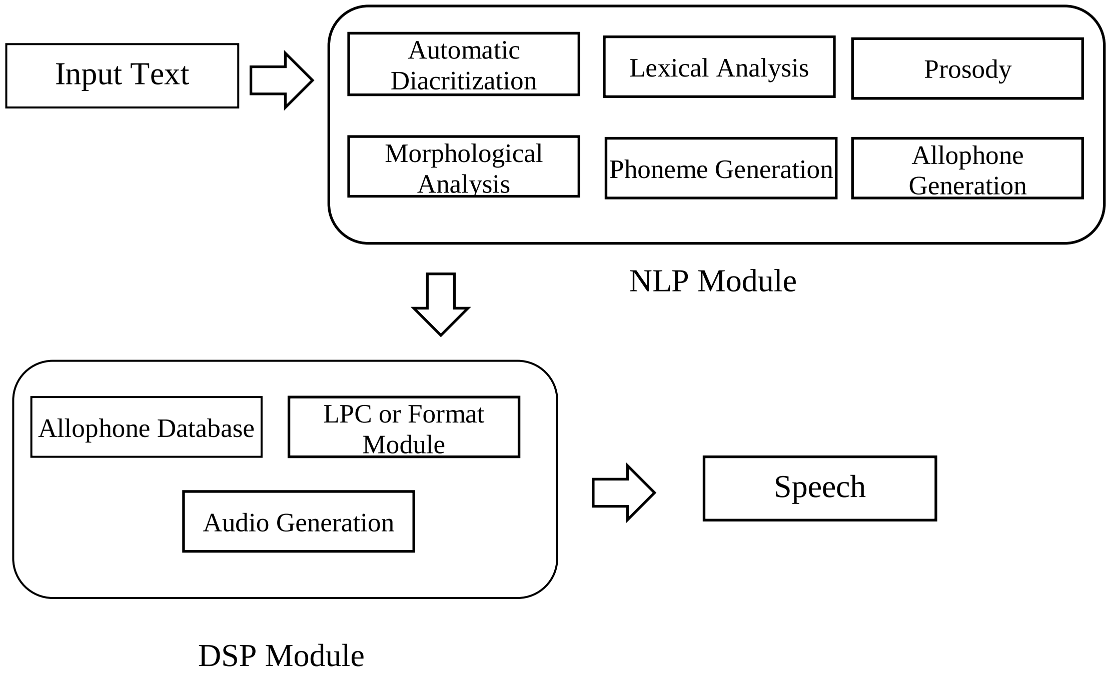


Figure 10 Text-To-Speech Process

3.2.2.1 NATURAL LANGUAGE PROCESSING

The NLP module is responsible for converting input text into phonetic pronunciation. The goal is to be able to generate as much information about the input text as possible, in order to produce proper pronunciation. Annotations provided by VoiceXML can be used to aid the process of speech generation.

NLP processing of the input text involves the following steps:

1- Text Pre-processing: Numbers, dates, currencies, and other symbols must be converted to their corresponding textual interpretation. VoiceXML tag <say-as> and <interpret-as> can be used to aid this process.

2- **Full Diacritization**: If the input text is non-diacritized or partially diacritized, an automatic diacritization module is used to fill the missing diacritic marks. Automatic diacritization is a process that involves different levels of text analysis, including morphology, syntax and semantics analysis. In addition, the <phoneme> tag can be used to aid the selection of the correct pronunciation for ambiguous words.

3- **Pronunciation Generation**: An algorithm analyzes the text to generate proper pronunciation. Phonetic analysis of Arabic is necessary to develop the correct conversion rules. This involves phonetic classification of Arabic sounds, syntax analysis of the input text, and application of special rules for exceptional cases in Arabic pronunciation.

4- Prosody: Prosody affects features of the spoken text such as pitch, duration, volume, and emphasis. Semantic analysis of the input text helps detecting questions and emphasized segments. Prosody attributes passed from the VoiceXML interpreter can also help determining the nature of the generated speech.

The output of the NLP module is a phonetic transcription of the text, which includes any necessary pronunciation marks such as stops and emphasis. The pronunciation is then transferred to the DSP module for audio generation.

3.2.2.2 DIGITAL SIGNAL PROCESSING

Digital signal processing module is responsible for synthesizing the audio output. Diphone concatenation is one of the common techniques for Arabic TTS [21]. A database of pre-recorded phoneme-pair audio files is used to generate the final audio. The TTS system might have more than one database, each resembling a voice. Application developers can switch between voices if needed using the proper VoiceXML tagging.

Various signal processing techniques are used to combine the diphones, including LPC formatting [21]. Signal processing is required to remove anomalies that occur in-between diphones, such as rapid change of volume or frequency that cause the generated output to have metallic sounds.

CHAPTER 4

ARABIC SPEECH RECOGNITION

As discussed earlier, providing support Arabic support involves development of Arabic acoustic and language models. And while VoiceXML simplifies the language models to domain specific grammars, it is still necessary to develop complete acoustic models to facilitate recognition of phonemes from different user groups.

The development of the acoustic model undergoes various steps:

1- Phoneme Selection

2- Word to Phoneme Mapping (Phonetic Dictionary)

3- Data collection and segmentation

4- Training of Hidden Markov Models

4.1 PHONEME LIST

The first step in developing the Arabic acoustic model is to define the phoneme set to be used. In this research we utilize a phoneme set which is based on text-to-speech work from [21]. The phoneme set contains 45 base phonemes. The combination of phonemes generated a set of 52,065 tri-phones. The complete list is shown in Table 1.

.

Table 1 List of Arabic Phonemes

Phoneme	Arabic Letter	Exampl e	Description
/AE/	ـَ	بَ	Short Vowel **FATHA**
/AE:/	ـَا	بَاب	Long Version of /AE/
/AA/	ـَ	خَ	Pharyngeal Version of /AE/
/AA:/	ـَ	خَاب	Long Version of /AA/
/AH/	ـَ	قَ	Emphatic Version of /AE/
/AH:/	ـَ	قَال	Long Version of /AH/
/UH/	ـُ	بُ	Short Vowel **DAMMA**
/UW/	ـُو	دُون	Long Version of /UH/
/UX/	ـُ	غُصن	Pharyngeal Version of /UH/
/IH/	ـِ	بِنت	Short Vowel **KASRA**
/IY/	ـِي	فِيل	Long Version of /IH/
/IX/	ـِ	صِنف	Pharyngeal Version of /IX/
/AW/	ـَو	لَوم	A Diphthong of both /AE/ and /UH/
/AY/	ـَي	صَيف	A Diphthong of both /AE/ and /IH/
/E/	ء		Arabic Voiceless Glottal Stop **HAMZA**, and a variation for **QAF** in some dialects.
/B/	ب		Arabic Voiced Bilabial Stop Consonant **BEH**
/T/	ت		Arabic Voiceless Dental Stop Consonant **TEH**
/TH/	ث		Arabic Voiceless Inter-dental Fricative Consonant **THEH**
/ZH/	ج		Standard Arabic Voiced Palatal Stop Consonant **JEEM**, similar to English /ZH/.
/G/	ج		Egyptian Dialect (and others) for **JEEM.** Also used in foreign names. A Velar version of /G/
/JH/	ج		A Voiced Fricative Version of Jeem, similar to the English /JH/
/HH/	ح		Arabic Voiceless Pharyngeal Fricative Consonant **HAH**
/KH/	خ		Arabic Voiceless Pharyngeal Velar Consonant **KHAH**
/D/	د		Arabic Voiced Dental Stop Consonant **DAL**
/DH/	ذ		Arabic Voiced Inter-dental Fricative Consonant **THAL**
/R/	ر		Arabic Dental Trill Consonant **REH**
/Z/	ز		Arabic Voiced Dental Fricative Consonant **ZAIN**, and a variation of **THAL** in many dialects.
/S/	س		Arabic Voiceless Dental Fricative Consonant **SEEN**
/SH/	ش		Arabic Voiceless Palatal Fricative Consonant **SHEEN**
/SS/	ص		Arabic Emphatic Voiceless Dental Fricative Consonant **SAD**
/DD/	ض		Arabic Emphatic Voiced Dental Stop Consonant **DAD**
/TT/	ط		Arabic Emphatic Voiceless Dental Stop Consonant **TAH**
/DH2/	ظ		Arabic Emphatic Voiced Dental Fricative Consonant **THAH**
/AI/	ع		Arabic Voiced Pharyngeal Fricative Consonant **AIN**
/GH/	غ		Arabic Voiced Velar Fricative Consonant **GHAIN**, also a variation of **QAF** in many dialects.
/F/	ف		Arabic Voiceless Labial Fricative Consonant **FEH**
/V/	-		Voiced Version of **FEH**. Exists in Foreign Names Only.
/Q/	ق		Arabic Voiceless Uvular Stop Consonant **QAF**
/K/	ك		Arabic Voiceless Velar Stop Consonant **KAF**
/L/	ل		Arabic Approximant Dental Consonant **LAM**
/M/	م		Arabic Nasal Labial Consonant **MEEM**
/N/	ن		Arabic Nasal Dental Consonant **NOON**
/H/	هـ		Arabic Voiceless Glottal Fricative Consonant **HEH**
/W/	و		Arabic Velar Approximant Semi-vowel **WAW**
/Y/	ي		Arabic Palatal Approximant Semi-vowel **Yeh**

The regular Arabic short vowels /AE/, /IH/, and /UH/ correspond to the Arabic diacritical marks Fatha, Damma, and Kasra respectively. The /AA/ is the pharyngealized allophone of /AE/, which appears after an emphatic letter. Similarly, the /IX/ and /UX/ are the pharyngealized allophones of /IH/ and /UH/ respectively. When /AE/ appears before an emphatic letter, its allophone /AH/ is used instead. When a short vowel is located between two nasal letters in the same syllable it is likely to be nasalized. The allophones /AN/, /IN/, and /UN/ are the nasalized versions of /AE/, /IH/, and /UH/ respectively, however, they were not considered in this reported work.

The regular Arabic long vowel allophones are /AE:/ /IY/ and /UW/ respectively. The length of a long vowel is normally equal to two short vowels. The allophones /AY/ and /AW/ are actually two vowel sounds in which the articulators move from one post to another. These vowels are called Diphthongs. The allophone /AY/ appears when a Fatha comes before an undiacritized Yeh. Similarly, /AW/ appears when a Fatha comes before an undiacritized Waw.

The Arabic voiced stops phonemes /B/ and /D/ are similar to their English counter parts. /DD/ corresponds to the sound of the Arabic Dhad letter.

The Arabic voiceless stops /T/ and /K/ are basically similar to their English counter parts.

The sound of the Arabic emphatic letter Qaf is represented by the phone /Q/. The Hamza plosive sound is represented by the phone /E/.

The normal allophone of Jeem is /JH/. The Arabic affricative sound /JH/ is similar to the corresponding one in English, while /ZH/ is a concatenation of a voiced stop followed by a fricative sound. The /ZH/ allophone is more common in the unvoweled positions, but could also replace /JH/ in some dialects. The third version of Jeem is the /G/ allophone which is commonly used in the Egyptian dialect. The /G/ allophone replaces the Qaf letter in many Arabic dialects.

The voiceless fricatives are produced with no vibration of the voice cords. The sound is produced by the turbulence flow of air through a constriction. The Arabic voiceless fricatives /F/, /S/, /TH/, /SH/, and /H/ are basically similar to their English twins. In addition, the Arabic phones /SS/, /HH/, and /KH/ are the sounds of the Arabic letters Sad, Hah, and Khah respectively.

Voiced Fricatives are generated with simultaneous vibration of the vocal cords. The Arabic voiced fricative phones are /AI/, /GH/, /Z/, and /DH/ corresponding to the sound of the Arabic letters: Ain, Ghain, Zain, and Thal.

The Arabic resonants are similar to the English resonant phones. These are /Y/ for Yeh, /W/ for Waw, /L/ for Lam, and /R/ for Reh.

4.2 PHONETIC DICTIONARY

In order to train the acoustic model using the data set, an intermediate **phonetic dictionary** is used to generate the phonetic transcription of the input text. The phonetic dictionary is also used in the decoding step to map recognized phoneme sequences to words.

The data set also contained instances of noise input. A filler dictionary is used to define noise events. Noise filtering is necessary for VoiceXML implementation platforms, filler words such as “uh” and “umm” are also added to the filler dictionary.

Arabic has a very formal set of pronunciation rules, especially for fully diacritized text. Hence it was possible to use a rule based approach to generate the phonetic dictionary automatically from the transcription text.

Using the selected phoneme set, we developed a set of rules that are used to automatically generate the phonetic pronunciations for Arabic words. We also created a set of tools that process the given Arabic text and generate all possible pronunciations for every word in the text.

Rules are provided for each Arabic letter available in the Unicode listing (45 letters). Each rule tries to match certain conditions on the context of the letter and provide a replacement from the phoneme list. Replacements can be one or more phonemes. Some letters don’t have an effect on pronunciation or, depending on context, they might not be pronounced; in this case, the replacement will be empty.

The rules used to generate the phonetic dictionary are in the form of context-dependent grammars that have the following format:

(pre_condition) . *(post_condition)* -> *replacement*

pre_condition and *post_condition* are rules that assert the content of the Arabic letter under investigation, which its position is denoted by the dot(.). Replacement can be one or more phonemes from the phoneme list or an empty string, denoted by an asterisk (*).

Multiple classes are defined to simplify the rules syntax. Each class is referenced by its symbol (L, D, S, etc.), surrounded by angle brackets (< >).

The classes are:

- <L>: All Arabic consonants.
- <D>: Diacritic marks (FATHATAN, DAMMATAN, KASRATAN FATHA, DAMMA, KASRA, SHADDA, and SUKUN).
- <S>: Word Start.
- <T>: Word End.
- <SH>: Shamsi Letters (TEH, THEH, DAL, THAL, REH, ZAIN, SEEN SHEEN, SAD, DAD, TAH, ZAH, LAM, and NOON).
- <V>: Vowels (FATHA, DAMMA, KASRA, and SHADDA).
- <VA>: Vowels without SHADDA (FATHA, DAMMA, AND KASRA).
- <P>: Prefix letters (WAW, BEH, FEH, KAF, and LAM).
- <E>: Emphatic letters (TAH, SAD, DAD, and ZAH).
- <PH>: Pharyngeal letters (QAF, GHAIN, KHAH, and REH).

The pre-condition has one of the following formats:

- *(?<=pattern)*: context before the current position matches the pattern.
- *(?<!pattern)*: context before the current position does not match the pattern.

In the same way, the post-condition has one of the following formats:

- *(?=pattern)*: context after the current position matches the pattern.
- *(?!pattern)*: context after the current position does not match the pattern.

Patterns use the following operators to define expressions:

Alternation: A vertical bar (|) is used to separate alternatives.

Grouping: Parentheses ( ) are used to define groups that determine scope and precedence of the operators and build complex expressions.

Optional Matching: A question mark (?) is used to mark parts of the expression that may or may not exist.

The right hand side of the rule defines the replacement, which can either be a phoneme or a sequence of phonemes from the phoneme list, or the letter might not have a matching phoneme and will be omitted from pronunciation. This case is marked with an asterisk (*) on the right hand side.

We define a rule set that covers all possible Arabic letters that are used in typing. Many of the rules are straight forward; they match the Arabic letters to their corresponding phonemes as explained in Table 1. Vowels require more elaborate rules to cover all possibilities. Special attention is required for nasalized consonants (Meem and Noon) and a few more exceptions that will be explained in the following sections.

The complete list of the rules is provided in Appendix 1.

The following is a sample from the generated phonetic dictionary:

Table 2 Sample Definitions from the Phonetic Dictionary

آبَارِ E AE: B AE: R IX N
آخَر E AE: KH AA R
آخَرَ E AE: KH AA R AA
آخَرُوْنَ E AE: KH AA R UW N AE
آخَرِيْنَ E AE: KH AA R IX: N AE
آخَرِيْنْ E AE: KH AA R IX: N
آخَرْ E AE: KH AA R
آخِذَةٌ E AE: KH IX DH AE T UH N
آخِرَ E AE: KH IX R AA
آخِرْ E AE: KH IX R

4.3 TRAINING DATA

A data set of 7 hours of business and sports news broadcast was used to build the base acoustic model. The database consists of diverse news broadcasts from different Arabic TV channels, including male and female broadcasters. Audio was recorded as 16 bit, signed PCM, *wav* files at a sampling rate of 16kHz. The audio files were cleaned from noise such as background music and then were split into smaller audio segments of length between 5 – 15 seconds, resulting in 6146 audio segments.

Each audio segment was manually transcribed using fully diacritized Arabic text. A set of clean up tools are then used to correct errors in transcription. The generation rules are then applied on the transcribed text to generate the phonetic dictionary.

4.4 THE ACOUSTIC MODEL

For feature extraction, audio is segmented into an analysis window of 25.6 msec (410 samples) with consecutive frames overlap by 10 msec. Each window is pre-emphasized and is multiplied by a Hamming window [74].

The basic feature vector uses the Mel Frequency Cepstrum Coefficients (MFCC). The basic feature vector consists of 13 values: 12 cepstral coefficients and energy. The basic feature vector is highly localized. To account for the temporal properties, delta and double-delta values are added, resulting in a 36-value feature vector per window.

The adopted HMM model is the Bakis Model, which has a fixed topology consisting of 3 emitting sates and one output state. Output probability distributions in HMM states are modeled with mixtures of 8 diagonal covariance Gaussians. First, context-independent HMMs of one Gaussian were trained. Then these HMMs were successively extended to one more Gaussian and re-estimated to up to 8 Gaussians. Baum-Welch re-estimation was used with a minimum of 7 iterations until the estimation converges. An estimation convergence ratio of 0.04 was used.

The generated model consisted of 8348 triphones with more than 25000 HMM states. Senons are used to reduce the number of generated states by tying together highly similar states. The states are tied based on the classification of phonemes. The tied model consists of 1638 senons.

4.5 EVALUATION OF THE ACOUSTIC MODEL

To test and validate the proposed phoneme set and the conversion rules we split the audio recordings into training and testing sets. The training set contained around 6 hours of audio while the testing set contained the remaining 1 hour. We used the CMU language toolkit to build a statistical language model from the transcription of the full 7 hours of audio.

Next, several test cases were built to validate our choice of the phoneme set. The main focus was on vowels since they impose most of the complexity and variety to the rules. For each of these cases the entire Arabic speech recognition engine is re-trained with the modified dictionary, and performance of the speech recognition is evaluated on the test set of the speech utterances (1144 voice files).

First, we tested the AASR system using the phone set and the rules outlines in the previous sections as the base system.

We then developed four test cases:

1- The first test case studies the effect of removing the emphatic and pharyngeal vowels. The test case was built in which we removed these vowels from the phoneme set and omitted the rules for emphatic and pharyngeal vowels.

2- In the second test case we examined the effect of merging the long and short versions of vowels into one vowel (for example /AE/ and /AE:/) to test whether the tri-phone models in the speech recognition engine would be capable of handling these vowels without the need to introduce additional phonemes.

3- In the third case we examined alternative rules for handling the vowels preceding the definite article AL ALTA'REEF (الـ). And for certain cases of the short vowel /AE/ and the long vowel /AE:/.

4- In the fourth case we examined alternative rules for dealing with gemination Shaddah, and for co-articulation effects of the emphatic consonants on the preceding vowels.

Results for these test cases are shown in Table 3.

Further analysis indicates that many of the word substitution errors are due to slight differences (deletion/substitution) of diacritical marks, especially the end cases. Since MSA text is written without diacritical marks, the error analysis was carried out once more after removing all the diacritical marks. The percentage of the correctly recognized words was 92.84%. The WER dropped to 9.0%.

Table 3 Evaluation of the Acoustic Models

	Test case	Accuracy	Ins	Del	Sub	WER
	Base system	90.1	168	82	838	11.71
1-	Emphatic and pharyngeal versions of vowels were removed	89.8	168	89	858	12
2-	Long versions of Vowels were removed	87.96	179	99	872	12.33
3-	Alternative rules for FATHA and for /AE/ and /AE:/ in the definite articles	88.47	214	80	991	13.84
4-	Alternative rules for co-articulation effects of the emphatic consonants	89.81	157	77	869	11.88

The results of these tests lead us to conclude that it is necessary to include both emphatic and pharyngeal vowels and maintain separate phonemes for short and long vowels. The tests clearly validate the proposed phoneme set and the proposed rules for automatic generation of the Arabic pronunciation dictionaries for Arabic speech recognition applications. However, we believe also that more research work is still needed to achieve better accuracy results.

The rules we proposed were based on the assumption that the triphone model used in HMMs will be able to capture gemination cases (double consonants). To validate that, we built an additional test case where we replace geminated letters with double phonemes, for example, a Beh followed by Shaddah will be replaced by (/B/ /B/). However, this didn’t improve the accuracy of the model.

CHAPTER 5

SYSTEM ARCHITECTURE

As discussed earlier, VoiceXML brings the power of web development to Spoken Dialog Systems, providing it with levels of abstractions that streamline the process of voice application development. There are, however, inherited limitations in VoiceXML framework that reduces it usability, especially in research communities:

1- The integration with the telephony system introduces high deployment costs, including prices for specialized hardware for call management and transfer. The cost barrier also prevents developing countries from fully utilizing IVR services as discussed in [7].

2- Traditional VoiceXML applications are highly localized; In order to deploy the system, a local phone number must be reserved for receiving calls. While this might be viable for corporations that can afford local or toll-free numbers, research teams, especially small ones, might not be able to provide a dedicated line or the hardware needed to support it. Moreover, International users might face many limitations trying to use the application.

3- VoiceXML is based on the traditional telephony system, where voice is the only possible interface. Nowadays most telephonic communications are accomplished using mobile devices that have a lot more capabilities than older models. Efforts for multi-modal interfaces try to overcome this issue; however, the interfaces cannot communicate directly and have to use 3rd party solutions or socket communications.

4- VoiceXML is highly implementation dependent. Similar problem occurred for HTML for a long time. Many vendors might choose to alter certain specifications or add their own tags and attributes. While this problem is hard to overcome, it is necessary to promote open solutions that might gain wide acceptance and provide a unified implementation framework.

Our proposed framework attempts to provide solutions to these issues while also being:

1- **Accessible:** Any client using a modern, flash supporting, browser should be able to access the framework from any location. No extra hardware or software is needed.

2- **Supporting Rapid Development:** Developers can quickly build their own applications by uploading their documents and their associated data to an application container.

3- **Flexible:** The framework front end can be used as a pluggable component in web interface development frameworks. The backend is also capable of supporting any web development language.

4- **Extensible:** Developers with deeper knowledge can extend the framework and add new features to both the backend and the front end.

5- **Scalable:** While not relying on specific hardware, the system is built utilizing Cloud Computing solutions to be able to scale as the computing, memory or space requirements increase.

6- **Open:** The framework is built solely on available open source solutions and SDKs, providing researchers with extensive resources to examine and evolve.

7- **Multi-Lingual:** While the framework was specifically developed for Arabic support, it is easily usable for any other languages provided that proper data and models are available.

The following sections discuss the framework details in depth. The roles of each component as well as the communication protocols are explained. The libraries or frameworks used for each component are briefly introduced.

5.1 FRAMEWORK OVERVIEW

Figure 11 shows the main components of the frameworks. Each component can work as an independent sub-system. Communication between the components is performed using networking protocols such as HTTP, RTMP, and sockets.

1- **Front End:** The application's front end consists of ordinary HTML based web pages that can be static or dynamically generated. Since HTML doesn't have streaming capabilities, a plug-in component is used to stream audio between the front end and the back end.

2- **Streaming Server:** This server is responsible for accepting connections from clients. The streaming server acts as a middle man between the client, the VoiceXML interpreter, the speech recognizer and the text-to-speech engine, coordinating stream flow between these components and handling the session initiation and termination.

3- **Web Server:** This server acts as the container for both static and dynamic web applications. The server can also be used as a document server for VoiceXML documents. Depending on application requirements, there can be one or more web servers deployed in the framework.

4- **Data Store:** A database server responsible for storing local application data. The data store might also run background processes to extract external data, such as weather information, and provide it for the local applications.

5- **VoiceXML Browser:** A VoiceXML interpreter responsible for processing and executing VoiceXML applications. The interpreter is activated when the streaming server initiates a connection. The interpreter manages the application flow and initiates requests to send or receive audio.

6- **Speech Recognizer:** A speech decoder module responsible for handling recognition tasks initiated by the interpreter. The recognizer receives input streams from the streaming server and returns the recognized input to the interpreter.

7- **TTS Engine:** A speech synthesis module responsible for handling prompt requests initiated by the interpreter. The generated audio is passed through the streaming server to the user client.

The framework backend is built on Cloud Computing architecture to allow the framework to scale based on the user demand. Scaling can be performed in three dimensions:

1- **Processing power:** Mirrors of the streaming and web server can be created on the fly to cover for user traffic. The data store architecture provides transparent scaling without affecting other components. Application pools are used to manage instances of the interpreter, speech recognizer, and TTS engine; additional instances of each application can be deployed to fulfill user requests.

2- **Memory allocation:** Each component starts with a minimum allocated address space. The server can be configured to double or triple the memory allocation if the traffic on the site surpasses a certain limit.

3- **Disk storage:** Storage space can be increased, almost indefinitely, as the site grows. No data reallocation is needed and disk storage management is transparent to the framework.

In addition to scalability features, many Cloud-based solutions provide a flexible pay per use model, which provides significant reduction in deployment costs.

5.2 COMPONENT DETAILS

Each component uses a different set of technologies, protocols and relies on its own sub-components and libraries. Combining the components together involves lots of integration requirements that affect design decisions of these components. The integration requirements might also introduce limitations on components capabilities. The following sections discuss the design details for each component and integration interfaces to other components.

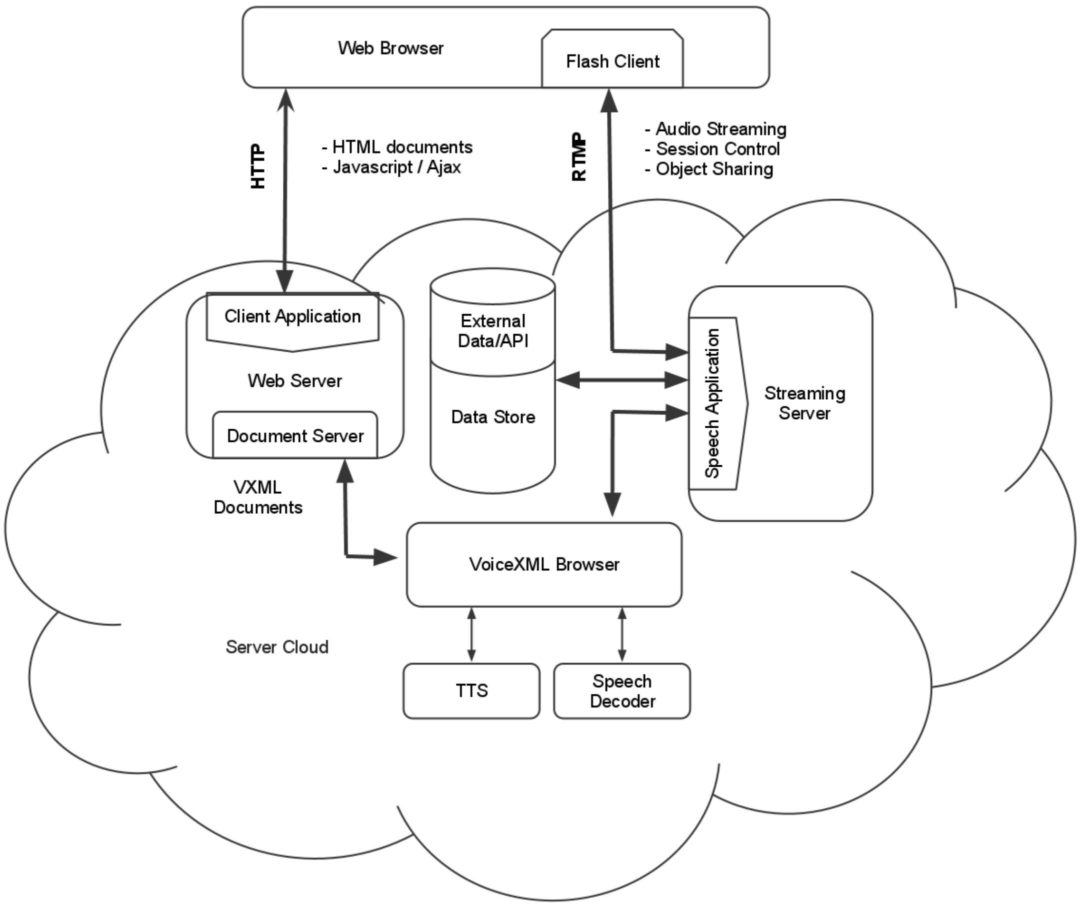


Figure 11 Framework Overview

5.2.1 WEB CLIENT

The web client can be any modern web browser. The client interface is majorly based on standard HTML/CSS/JavaScript coding. Unfortunately, current HTML (4.1) standard doesn't provide a direct way to stream multimedia. HTML5 provides an initial support for streaming using the <audio>, <video>, and <devices> tags. However, HTML5 leaves the protocol and encoding support to be native to the implementation, reducing the portability of HTML5 based applications. HTML5, however, can be used to build platform specific applications such as applications for the iPhone.

Due to the limitations of current HTML specifications, an external plug-in is needed to handle the streaming part of the web interface. Flash is used to develop the streaming component due to the following reasons:

1- Flash is widely supported by major browsers on almost all operating systems.
2- It is very likely that the client machine has the Flash plugin already installed, eliminating the need to install extra add-ons to be able to use the system.
3- Flash provide out of the box streaming solutions for audio, video, and message exchange with the server.
4- Flash is capable of capturing and streaming microphone input directly. In addition, Level controls can be used to reduce background noise when streaming.
5- Flash is not completely isolated from the HTML container. Flash can access and manipulate the document's DOM tree and JavaScript can be used to interact with the flash component.

The developed component is responsible for performing the following tasks:

1- Connecting to the server, initiating a new application session.
2- Receiving audio prompts from the backend and playing them.
3- Capturing microphone input when prompted by the server and streaming it back using the proper encoding attributes.
4- Showing any debugging, informative messages sent by the server.
5- Disconnecting from the server, closing the current session.

Flash clients support the **Real-Time Messaging Protocol (RTMP)** [75] which can be used to stream audio, video, and data between the flash client and the streaming server.

RTMP is a TCP based protocol that allows for real-time communication over a persistent connection. The transmitted data may be split into smaller packets to facilitate smoother communication. Data is transmitted on different channels, including audio, video, remote procedure calls and packet control channels. Each packet sent can belong to one of these channels only.

RTMP can be used for streaming live and recorded media and is being used by large scale streaming services such as YouTube and Google Video.

RTMP works on port 1935 by default. It is possible, however, to tunnel the stream through HTTP protocol to avoid firewall limitations. The HTTP encapsulated version of RTMP is called RTMPT.

Flash clients support two audio formats that can be streamed through RTMP:

1- **Nellymoser Codec**: A proprietary codec owned by Nellymoser Inc. Flash is capable of streaming audio using that codec at different rates (5kHz, 8kHz,11kHz, 22kHz, 44kHz). Until recently, there were no available free decoders for this coded. However, FFmpeg project [76] now provides encoding and decoding support for this codec.

2- **Speex Codec**: A codec designed specifically for encoding speech. Flash supports this codec at 16kHz rate only. The codec supports advanced features such as Voice Activity Detection (VAD) and can help reduce the bandwidth usage when no speech is detected. Choosing Speex decoder allows specifying quality levels for the transmitted speech.

The choice of the proper codec depends mainly on the codecs supported by the speech recognizer. If the speech recognizer doesn't support any of these formats then there will be a need for an intermediate encoder/decoder to convert the audio streams. The choice of the codec might also affect the quality of speech recognition, especially when the stream is re-encoded in the middle.

While Flash is a proprietary technology, Adobe provides Flex SDK as an open source platform for developing Flash applications, which contributes to the openness of the framework.

5.2.2 STREAMING SERVER

The streaming server is responsible for transferring audio streams between the web client and other server components. It is also responsible for initiating and ending application sessions. The streaming server is responsible for handling encoding/decoding tasks as required.

Specifically, the streaming server is responsible for the following tasks:

1- Accepting client connections. As per VoiceXML standard, the connection defines a new application session.

2- Invoking a VoiceXML interpreter to handle the new user session.

3- Receiving prompt requests from the interpreter and passing them to the TTS engine.

4- Encoding and forwarding audio and TTS streams to the web client.

5- Listening for input stream from the web client.

6- Encoding and forwarding the input stream to the speech recognizer.

7- Forwarding the recognition results to the interpreter.

8- Exchanging other information needed by the web client.

9- Detecting disconnects and informing the interpreter accordingly.

10- Closing the session when the application ends.

An RTMP streaming server is required to perform these tasks. In our implementation we use Red5 [77] streaming server for this task. Red5 is a Java based open source streaming server that implements most of the RTMP features. Red5 follows the web application container framework used by other web containers such as Tomcat and Jetty; Red5 provides Red5 applications with baseline streaming capabilities, while allowing application developers to develop custom handlers that manipulate the streams.

Red5 is capable of recording and streaming audio using Nellymoser codec only. A stream encoder is required to change the audio codec and rate into a format acceptable by the speech recognizer. Fortunately, Red5 can be easily integrated with Xuggler [78], a Java based media encoder that is built upon the popular FFmpeg library.

We utilize Red5 by developing a streaming application responsible for the following:

1- Interacting with VoiceXML for the various application events.

2- Encoding and decoding audio to the proper media formats.

3- Record audio messages to the data store when needed.

5.2.3 VOICEXML INTERPRETER

As discussed earlier, the interpreter is responsible for the following tasks:

1- Initiating the application session.

2- Acquiring and parsing VoiceXML documents from the document server.

3- Performing the Form Interpretation Algorithm and processing the Prompt Queue.

4- Closing the session.

The interpreter is the central component in the framework, interacting with the streaming application, the document server, the speech recognizer and the TTS engine.

We use JVoiceXML [79], a Java based VoiceXML interpreter, as the Voice browser in our framework. JVoiceXML supports a wide range of the standard features while continuously improving the supported feature set.

JVoiceXML is built for extensibility; it provides the base interpreter as well as a set of common implementations for platforms such as JSAPI1&2, CMUSphinx, FreeTTS, Open Mary, and others. The platform can be easily extended to support other implementations, allowing it to be easily integrated into the platform.

JVoiceXML browser is deployed into an individual Java Virtual Machine (JVM) that can be accessed using one of two methods:

1- Remote Method Invocation (RMI).

2- Socket Connections.

Both approaches can be used when communicating from the streaming application.

5.2.4 SPEECH RECOGNITION

The speech recognizer is responsible for processing user input and generating the semantic interpretation required by the interpreter.

We utilize Sphinx4 [80] to perform speech recognition tasks within our framework. Sphinx4 is a Java based framework for speech recognition developed by the Speech group at CMU. Based upon the success of Sphinx-2 and Sphinx-3, both were developed in C++, Sphinx4 served as a port of the algorithms and techniques of the previous implementations as well as an integrable solution that complies with JSAPI specifications.

Sphinx4 has a modular architecture that allows developers to extend or swap modules without affecting the remaining parts of the system.

Figure 12 demonstrates the sub-systems available in Sphinx4 and the communication means between them. The following are the main sub-systems in Sphinx4:

- **The Front End:** This sub-system is responsible for feature extraction. MFCC feature set is used in this system.
- **The Linguist:** The Linguist uses the provided data models to produce a search graph that would be used for the recognition task.
- **The Decoder:** This sub-system does the actual recognition job. When speech is entered into the system, the Front End converts it into features as described earlier.

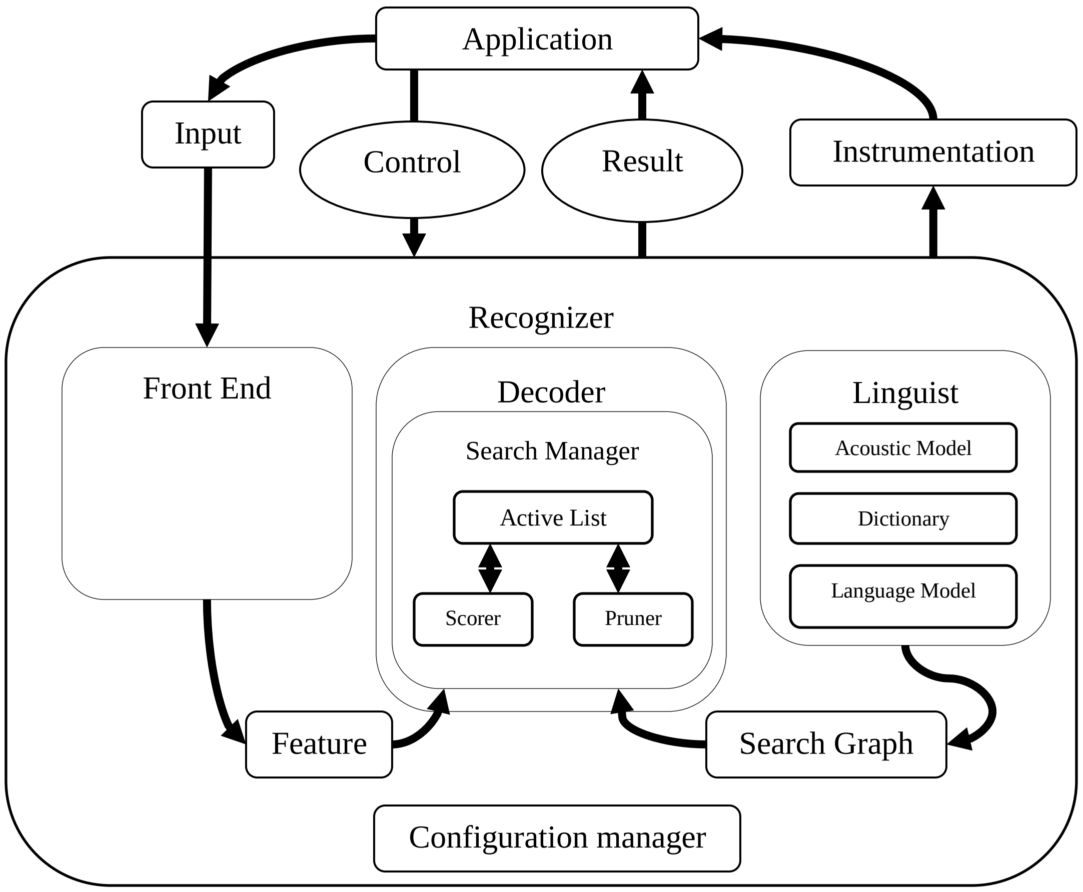


Figure 12 Sphinx4 Framework

The Decoder takes these features, in addition to the search graph provided by the Linguist, and tries to recognize the speech in the features.

- **The Configuration Manager:** This sub-system is designed in order to make Sphinx4 pluggable and flexible. Configuration information is provided in XML files, and the actual code uses the Configuration Manager to look up the different modules and sub-systems.

Sphinx4 is an HMM based speech recognizer that can be used to train and recognize Arabic input. The required models and data for the dataset mentioned in chapter 4 were developed and tested on Sphinx4 decoder.

JVoiceXML is already equipped with an implementation platform for Sphinx4. One limitation in Sphinx4, however, is that it is unable of receiving new grammar on runtime; Grammar must be provided as one grammar file at the beginning of execution of the application.

5.2.5 TEXT TO SPEECH ENGINE

While JVoiceXML provides support for many TTS solutions such as FreeTTS and OpenMary, none of them provide Arabic TTS at this stage. The only available Arabic TTS solution as of the writing of this chapter is the MBROLA project [81], which provides two Arabic voices. However, MBROLA lacks the algorithm to convert Arabic text to phonetic pronunciation. A commercial replacement, Acapela TTS [82], which is a spin-off of the MBROLA project, is used in this framework.

5.2.6 DATA STORE

The data store is used by two components in the framework:

1- It is used by the streaming server to store recorded messages.

2- It is used by the Application Server to read and store application data.

In order to be able to scale, cloud based solutions such as Amazon’s SimpleDB [83] are used for the data store.

CHAPTER 6

CASE STUDY AND EVALUATION: TADAWUL APPLICATION

As discussed in previous chapters, VoiceXML provides the basis for web development of voice applications. The proposed framework adds the missing components that enable building web based voice applications. The Cloud backend provides scalable processing power that can extend to fit the application needs. It is therefore the task of application developers to utilize the framework to build rich content interactive applications. In addition, the framework can be extended to support additional features such as multi-modal interfaces.

Developers can extend the framework functionality in two ways:

1- **Web Front End:** The front end component can be extended to provide other functionalities such as: page navigation, graphs and visualizations, and other interactive elements. For instance, an e-learning system for kids can use cartoon characters that interact with students and increase their engagement with the system. Various visualization tools can be easily integrated inside Flash environment.

2- **Streaming Handler:** The streaming application is responsible for transcoding streams and communicating with the interpreter. Additional functionalities can be added for further manipulation of input and output speech. For example, a module can be used to collect prosodic information from the users to help detect frustration events and gain more insight about how users interact with the system.

In the following section we present a sample application for interacting with the Saudi stock market (Tadawul) and the various interactions with the system components involved in this application.

6.1 SAUDI STOCK MARKET (TADAWUL)

The Saudi stock market is one of the most active markets in the region. As of January 2011, the market lists 146 publically traded companies in 15 sectors. The market is legally organized and supervised by the Capital Market Authority (CMA).

Investors have access to the market through its official portal <http:// www.tadawul.com.sa> and various licensed providers. Online banking and ATM machines can also be used for stock exchange. Many tools, both web based and desktop, are available for users to organize and plan their investments. For instance, by accessing the market's official website, users can perform the following actions:

1- View a summary of the market performance, including the value traded, trading volume, trade count, number of traded symbols (companies) and the number of symbols with increased or decreased values.
2- Gain information about the market index, including the global market index and individual sector performance.
3- View a summary of the highest and lowest performing stocks.
4- Gain detailed information about each stock symbol. Including minute by minute valuation, the number of trades, the opening and closing prices, and the company performance over various time periods.

5- Registered users can save their favorite stocks into a watch list for quicker access.

The market is accessible in both Arabic and English. Each registered company can be identified using one of the following keys: a numeric 4 digit symbol, a short name and an acronym. For example, Al Rajhi Bank has the numeric symbol as 1120, the short name "Al Rajhi" and the acronym "RJHI". The Arabic interpretation of the short name and acronym is different; usually the acronym is a shorter version of the name reducing it to one word or incorporating the initials of some of the words.

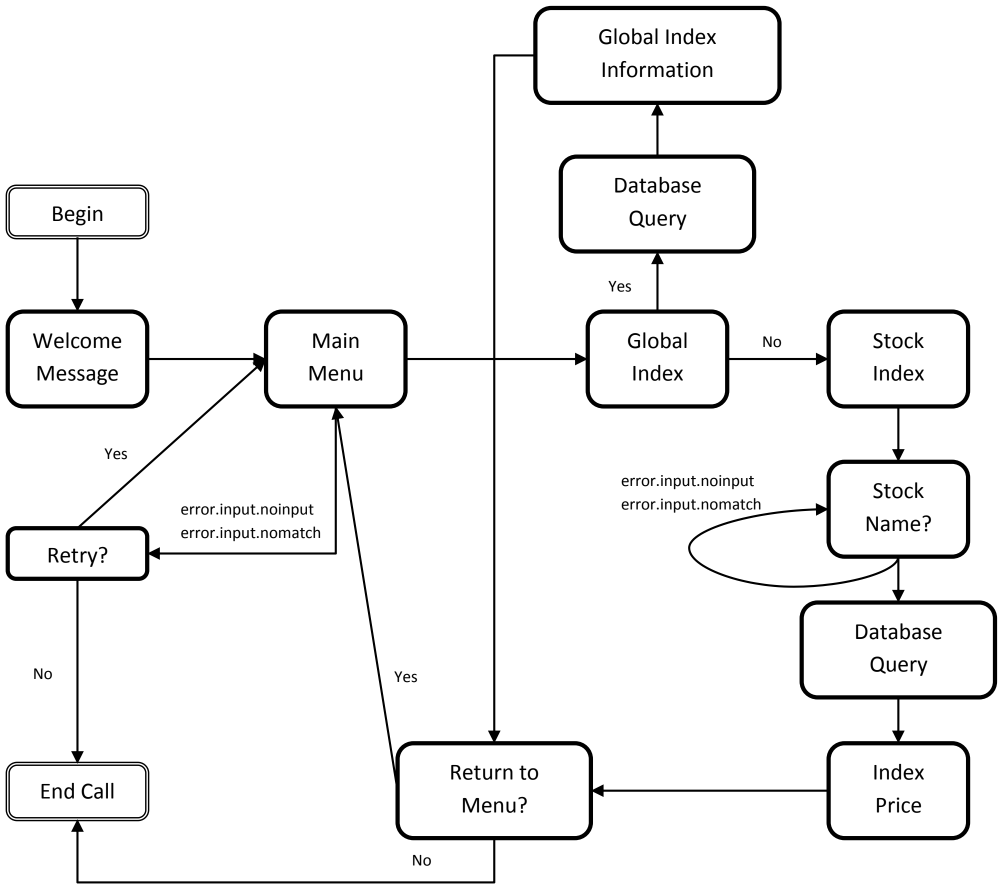


Figure 13 Dialog flow

Figure 13 shows sample dialog flow for a voice based Tadawul navigation application. When a user is connected to the system, he is presented with a short welcome message followed by the application's main menu. The main menu prompts the user to choose one of the following options:

1- **Market Global Index:** Using this option the user is presented with global information about the market performance at the time of the query.
2- **Specific Symbol Information:** Using this option the user will say the symbols name, short name, or acronym to get information about the the company. DTMF input can be also used to enter the numeric identifiers.

Various grammar sets are used to define the acceptable inputs, including:

1- Main Menu Grammar: Two possible inputs can be provided by the user: (المؤشر الرئيسي) "Global Index" and (سعر السهم) "Symbol Price". The choices can be abbreviated to (المؤشر) "Index" and (السهم) "symbol" for the user convenience.
2- Yes/No Grammar: This grammar is crucial for decision making nodes. Possible values are (نعم) "yes" and (لا) "no".
3- Companies Grammar: A grammar combining all possible company names as well as their short names and abbreviations. Names that belong to the same company are given the same tag.
4- Symbols DTMF Grammar: Each numeric symbol consists of exactly 4 digits. A DTMF grammar that matches exactly four consecutive digits is sufficient for this task.

Figure 14 shows a sample tagged grammar for the Saudi British Bank (SAB).

```
#JSGF V1.0;
grammar companies;
public <companies> = (.. |{سَاب} بِرِيطَانِي | سَاب | البَاحَة | عَسِير | ..);
```

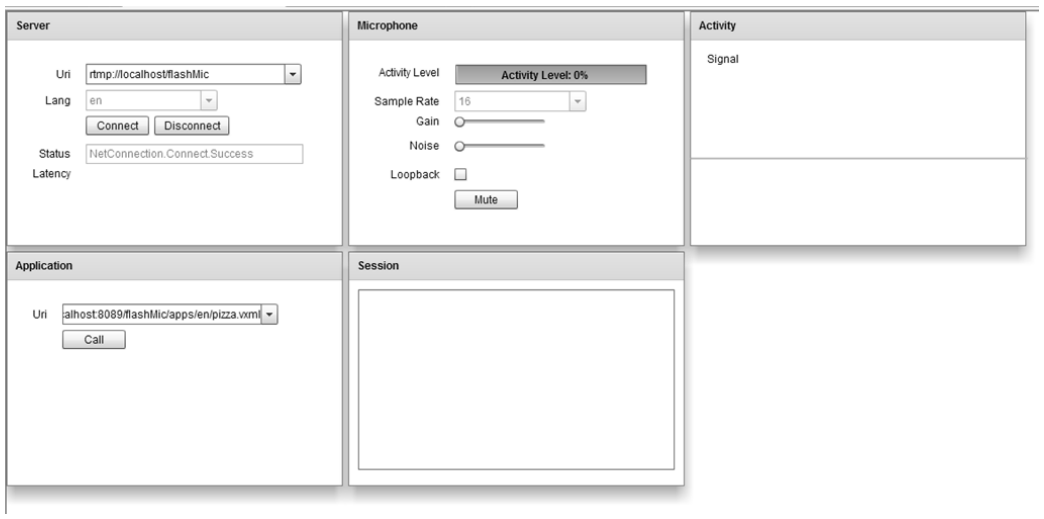
Figure 14 Sample Tagged Grammar

An application specific phonetic dictionary is built using all possible text inputs in the grammar. As mentioned earlier, the text tokens have to be fully diacritized to get the correct phonetic transcription. A special case that requires attention is companies with foreign names, which might have different phonetic rules and therefore their names have to be transcribed manually.

Database queries are performed by submitting the user input to a web page that dynamically generates a response in the form of a new VoiceXML document. The generated documents provide the users with the required information then transfer the control back to the root document.

Event handling nodes are used to handle the “noinput” and “nomatch” events. Basically, users are re-prompted to provide their input. The prompts can be refined if the user fails to provide the proper input for a multiple times.

6.2 USER INTERFACE



Uri: rtmp://localhost/flashMic
Lang: en
Status: NetConnection.Connect.Success
Activity Level: Activity Level: 0%
Sample Rate: 16
Uri: alhost:8089/flashMic/apps/en/pizza.vxml

Figure 15 Application User Interface

Figure 15 shows the user interface for the application. It consists of 5 sub modules:

Server Module: This module is responsible for connecting to the streaming server. Users can specify the target server URI as well as the language (Arabic or English). The server status is shown as well as the latency when recording audio.

Microphone Module: This module monitors the microphone activity and allows the user to control various aspects about the microphone: Sampling Rate, Microphone Gain, Noise Reduction. It also allows the user to make the microphone loopback to the user speakers or mute the microphone.

Activity Module: This module provides a visual representation of the audio signal, which helps the user identify their noise level and the responsiveness of the microphone.

Application Module: This module lists all the available applications on the server.

Session: This module provides a text transcription of the recognized user input as well as the system output.

6.3 FRAMEWORK EVALUATION

The propose framework lays the foundation of a new generation web based voice applications. Design decisions involved in the framework development affect the framwork's flexibility and acceptance ratios.

The Front end is designed to be lightweight; a few minimal components are deployed to the user machine. Using Flash for the user client is suitable for most platforms. An equivalent, RTMP based, client application can be developed for platforms that don't support Flash such as the iPad, iPhone, and other mobile platforms.

Communication between the front-end and the streaming server is trivialized by modern high bandwidth connections. A microphone stream that uses 16kHz sampling rate requires around 32Kbits/sec bandwidth [84]. Latency, however, could be an issue especially for shared and wireless connections. In situations of high latency, users are advised to switch to wired connections and try to avoid network traffic peak times.

The system backend consists of interdependent components. The components are loosely coupled; each component runs on its own environment, various networking protocols are used for communication. While being loosely coupled allows scaling each component independently, communication and encoding overhead might affect the framework performance. Moreover, each component is totally oblivious about the internal operations of other components, which limits the options for optimization of the back end.

Load balancing and optimization features are solely based on traffic usage. While this is typically enough for overall performance, computation intensive processes such as the speech recognizer might need extra processing power than other components.

Caching is one of the issues not addressed by the current framework. Various components of the system generate output that can be reused later. For instance, common speech prompts that are identical in multiple user requests might better be replayed from a recorded audio. Web caching techniques can also be applied to reduce the bandwidth on requesting VoiceXML documents. Front end content can be compressed and minified using common web compression techniques such as the gzip algorithm.

Being an interactive system, user satisfaction is a very important indicator. While it might not be possible to directly measure satisfaction, the following attributes will affect the level of engagement with the system:

1- **Responsiveness:** The lesser the delay between actions the better.

2- **Accuracy:** Correctly detecting user intent reduces the hassle for the user.

3- **Clarity:** Using uniform, clear messages reduce user anxiety and help the users reach their target faster.

6.3.1 RESPONSIVENESS

Major delays affect the system performance badly. The current framework has the following bottlenecks that may affect interaction speed:

1- **Network Delay:** While network delays are inevitable, various attributes can be adjusted to reduce network overhead:

   a. Amount of Data Transfer: Reducing microphone and audio bitrate helps reducing the network overhead. However, very low bitrates will reduce the system accuracy and clarity.

   b. Physical Location: Having the server cloud deployed nearby the target audience reduces the service latency. Unfortunately, no cost-effective solutions exist right now in the Middle East.

   c. Number of Streams per Client: RTMP uses at least two media streams between two end-points: downstream and upstream. RPC calls, however, can be transferred in both directions using one network connection. It is recommended to try to reduce the number of network connections and streams to the minimum.

2- **Streaming Delay:** While RTMP streaming is intended to be real time, streaming servers attribute to the delay due to computation intensive processes such as media transcoding. In order to reduce streaming delay, the following actions might be performed:

   a. **Adjust Packet Size:** Bigger packet sizes reduce the server overhead but might reduce the response time of the system. RTMP streaming can dynamically adjust the packet size to conform to Quality of Service requirements.

   b. **Reduce Transcoding Overhead:** Avoid inefficient transcoding algorithms. User lower bitrates if sufficient.

3- **Processing Delay:** Speech recognition and TTS synthesis are computation intensive processes. Caching can be used reduce TTS overhead. Fast decoding techniques can be used to improve responsiveness while sacrificing an accuracy margin. In general, reducing the search space improves recognition speed. The following actions can be used to improve recognition speed:

   a. **Use tied state acoustic models:** Tied state “Senone” models have lesser number of nodes and therefore are faster to search.

   b. **Reduce the size of the Phonetic Dictionary:** Eliminate unnecessary and uncommon pronunciations. User lesser number of phonemes.

   c. **Increase the pruning threshold:** Various parameters can be used to fine tune the speech recognizer. Pruning eliminates search paths with lesser likelihood. By increasing the pruning threshold, fewer paths are visited while decoding.

6.3.2 ACCURACY

There is a tradeoff between responsiveness and accuracy. The faster the recognizer responds the more error prone it will be. However, other factors affect the recognition accuracy:

1- **Noise Level:** High levels of noise or the use of poor quality microphones will reduce the recognition accuracy. Noise filtering techniques are needed to reduce the noise effects. Front end filtering also helps removing noise between utterances.

2- **Task complexity:** The more complex the recognition task is, the higher chances for errors will be. It is important to design the input grammar to be granular as much as possible.

3- **User behavior:** Users vary in tendency when speaking. It might be helpful to design the system to be smart enough to instruct the users to raise/lower their voices or speak faster/slower when applicable. Maintaining user profiles also help developing an adaptive recognition system.

6.3.3 CLARITY

Designing uniform and clear dialogs helps improving the acceptance rate of the system. Providing clear and instructive prompts helps the users find their way around the system. Being able to recover from errors gracefully reduces user frustration with the system. Various design techniques from the field of Human Computer Interaction (HCI) can be applied enhance the user experience with the framework.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this thesis, we proposed a framework for web based dialog system for the Arabic language. The framework benefits from three main factors:

1- The recent advancements in Arabic speech processing techniques that allow for high quality output.
2- The increasing adoption of VoiceXML as a framework for web development for voice applications.
3- The affordable and powerful Cloud computing infrastructure that is becoming the basis for modern web development.

We first introduced the concepts behind spoken dialog systems and VoiceXML. We then discussed the unique features of the Arabic language that affect speech processing. Chapter 2 provided an analysis of various approaches and techniques developed to overcome issues for Arabic ASR as well as the various approaches to deploy VoiceXML applications. In order to determine the effort needed to provide Arabic support to VoiceXML, chapter 3 provided the necessary background required to understand VoiceXML and Speech Recognition. Chapter 4 discussed in details the steps needed to provide an Arabic implementation platform and the possible limitations. Chapter 5 explained the main framework components, their interactions, and the design decisions involved in each component, while chapter 6 described a sample application built upon the framework and the various evaluation issues involved.

The work in this thesis lays the foundation of the next generation of Arabic speech recognition. The long term vision is to have voice as a trivial interface to computing devices, in ways similar, if not superior, to what’s being applied to touch and body gesture systems. Web based development is gaining grounds from desktop applications and modern web applications are more sophisticated than they were 10 years ago. It would be only a matter of time to have voice as part of web standards. However, several milestones are yet to be achieved. And being able to provide these features for a language as widespread as Arabic is a key the development of the whole region.

Further enhancements can be applied to the framework, including the following:

1- Improving the web-front end by adding navigational and other interactive features in order to be able to build multi-modal interactive systems.
2- Arabic TTS support is still minimal. VoiceXML provides a rich set of TTS features that can enhance the user experience significantly.
3- Development of reusable dialog components for Arabic as a way to help build rapid voice applications.
4- Incorporation of more NLP units such as automatic diacritization to reduce the development overhead for Arabic grammars.
5- Exposing the framework as a public API by utilizing Software as a Service (SaaS) techniques.

APPENDIX

I. PHONETIC DICTIONARY PRONUNCIATION RULES

Pronunciation Rules	**Description**
`HAMZA:` `.-> E` `ALEF_WITH_MADDA_ABOVE:` `.(?!FATHA)-> E AE:` `.(?=FATHA)-> E` `ALEF_WITH_HAMZA_ABOVE:` `.-> E` `WAW_WITH_HAMZA_ABOVE:` `.-> E` `ALEF_WITH_HAMZA_BELOW:` `.-> E` `.(?!KASRA|KASRATAN)-> E IH` `YEH_WITH_HAMZA_ABOVE:` `.-> E`	A HAMZA or other letters that are combined with a HAMZA will always be replaced by the phoneme /E/. The rules take in consideration cases where the user might have forgotten to put the proper dicritization over the (آ) and (إ). Both symbols have implicit vowels that must be included even if the user misses them. This is not the case, however, for (أ), (ئ) and (ؤ). User must explicitly put the vowels when appropriate.
`ALEF:` `(?<=<S>).-> E` `.-> *`	The ALEF (ا) is always omitted in pronunciation. The only exception is when the letter comes at the beginning of the word, in which case it might be pronounced as glottal stop /E/. We cannot be certain which pronunciation the user will use. Words that have an ALEF at the beginning will at least have two definitions that alternate between the two possibilities.
`BEH:` `.-> B`	The letter BEH (ب) is always matched with the phoneme /B/.

TEH_MARBUTA: .(?=<T>)-> H .-> T	The letter TEH_MARBUTA is formally pronounced as an /H/ if the speaker stops on it. However, some dialects will pronounce it as a /T/ regardless of stopping or not.
TEH: .-> T	The letter TEH (ت) is always matched with the phoneme /T/.
THEH: .-> TH .-> S	The standard pronunciation for the letter THEH is /TH/. However, in many dialects, such as the Egyptian dialect, it is pronounced as /S/.
JEEM: .-> J .-> JH .-> G .-> ZH	JEEM is another letter that has multiple pronunciations depending on dialectal differences. Also, foreign names (in the Egyptian standard) that have either G or J are always transliterated to a JEEM.
HAH: .-> HH KHAH: .-> KH	These letters are always matched to their corresponding phonemes.
DAL: .(?=TEH<V>)-> * .(?!TEH<V>)-> D	If the letter DAL is followed by a vowelled TEH then it is omitted in pronunciation. For example, the word "أردتُم".
THAL: .-> DH .-> Z	Multiple pronunciations are caused by different dialects.
REH: .-> R ZAIN: .-> Z SEEN:	These letters are always matched to their corresponding phonemes.

`.-> S`
`SHEEN:`
`.-> SH`
`SAD:`
`.-> SS`	

`DAD:`
`.(?=(TEH|TAH)<V>)-> *`
`.(?!(TEH|TAH)<V>)-> DD`
	If the letter DAD [(ض) is followed by a vowelled TEH or TAH then it is omitted in pronunciation. For example, the words “أفضـتُم” and “اضطُررتم”. (germination taken care of by the acoustic model)

`TAH:`
`.-> TT`
`ZAH:`
`.-> DH2`
`.-> Z`
`AIN:`
`.-> AI`
`GHAIN:`
`.-> GH`
`TATWEEL:`
`.-> *`
`FEH:`
`.-> F`
`QAF:`
`.-> Q`
`.-> G`
`.-> GH`
`.-> E`
`KAF:`
`.-> K`
	These rules match the letters to their corresponding phonemes while taking account of dialectal differences.

`LAM:`
 	
`(?<=(<P><V>)?ALEF FATHA?).(?=<SH>)-> *`
 	
`.-> L`
	The first rule takes care of the case of the Shamsi Lam. If the LAM is part of the (الـ) followed by a letter from the Shamsi group then it is possibly omitted from pronunciation. However, not every speaker will abide to this rule all the time and exceptions might happen. That's why another rule will always create an alternative pronunciation that has the /L/ phoneme included.

`MEEM:`
 	
`.-> M`
 	
`NOON:`
 	
`.(?=BEH)-> M`
 	
`.-> N`
	The letter MEEM (م) is always pronounced as an /M/. The letter NOON (ن) might also be pronounced as an /M/ if it was followed by a BEH. But again the speaker might not abide to that rule and pronounce it as an /N/ always.

`HEH:`
 	
`.-> H`
	HEH is always replaced by /H/.

`WAW:`
 	
`(?<=(FATHA|DAMMA)).(?!<V>)-> *`
 	
`(?<=(FATHA|DAMMA)).(?=<V>)-> W`
 	
`(?<!(FATHA|DAMMA)).-> W`
	The letter WAW (و) is sometimes treated as semi-consonants /W/ or /AW/ and other times it is treated as a long vowel, depending on its context. If the letter WAW is not vowelled and is preceded by a DAMMA then it is considered to be a long vowel. Examples include "نُجُوم".
	
	The case of the semi-vowel /AW/ is similar to the long vowel, except it is then preceded by a FATHA. In this case the WAW is omitted. The insertion of the /AW/ phoneme is handled by the FATHA rules as it will follow shortly.
	
	In the rest of the cases the WAW is

	converted to the semi-vowel /W/.

`ALEF_MAKSURA:`
 	
`.-> *`
	The ALEF MAKSURA (ى) is always omitted.

`YEH:`
 	
`(?<=(FATHA|KASRA)).(?!<V>)-> *`
 	
`(?<=(FATHA|KASRA)).(?=<V>)-> Y`
 	
`(?<!(FATHA|KASRA)).-> Y`
	As with the WAW, YEH (ي) is sometimes treated as the semi-consonants /Y/ and /AY/ and the other times treated as a long vowel. Rules here follow the same logic for the WAW.

`FATHATAN:`
 	
`(?<!<E>|<PH>).-> AE N`
 	
`(?<=<E>).-> AH N`
 	
`(?<=<PH>).-> AA N`
 	
`DAMMATAN:`
 	
`(?<!<PH>).-> UH N`
 	
`(?<=<PH>).-> UX N`
 	
`KASRATAN:`
 	
`(?<!<PH>).-> IH N`
 	
`(?<=<PH>).-> IX N`
	These rules differentiate between the emphatic and or pharyngeal versions of the vowels. Each rule appends an /N/ sound to the pronunciation. In order to reduce the complexity of the dictionary, we omit the case where the speaker stops at the end of the word and doesn't proceed to pronounce the /N/ sound. We assume that whenever the symbols for Tanween are present in the corpus that the user has actually pronounced them.

`FATHA:`
 	
`(?<!<E>|<PH>).(?!ALEF|((WAW|YEH)(<L>|<T>)))-> AE`
 	
`(?<!<E>|<PH>).(?=ALEF)-> AE:`
 	
`(?<=ALEF_WITH_MADDA_ABOVE).-> AE:`
 	
`(?<=<E>).(?!ALEF|((WAW|YEH)(<L>|<T>)))-> AH`
 	
`(?<=<E>).(?=ALEF)-> AH:`
 	
`(?<=<PH>).(?!ALEF|((WAW|YEH)(<L>|<T>)))-> AA`
 	
`(?<=<PH>).(?=ALEF)-> AA:`
 	
`.(?=WAW (<L>|<T>))-> AW`
 	
`.(?=YEH (<L>|<T>))-> AY`
	First two rules are responsible for the long and short versions of the normal vowels. Third rule is also for the long vowel where the FATHA is followed by an ALEF_WITH_MADDA_ABOVE.
	
	Rules 4-7 are for the emphatic and pharyngeal versions of the vowel.
	
	Rules 8-9 takes care of the semi-vowels AW and AY as mentioned in the rules for the WAW and YEH.

`DAMMA:`
`(?<!<PH>).(?!WAW)-> UH`
`(?<=<PH>).(?!WAW)-> UX`
`(?<!<PH>).(?=WAW<V>)-> UH`
`(?<=<PH>).(?=WAW<V>)-> UX`
`.(?=WAW(<L>|<T>))-> UW`
`KASRA:`
`(?<!<PH>).(?!YEH)-> IH`
`(?<=<PH>).(?!YEH)-> IX`
`(?<!<PH>).(?=YEH<V>)-> IH`
`(?<=<PH>).(?=YEH<V>)-> IX`
`(?<!<PH>).(?=YEH (<L>|<T>))-> IY`
`(?<=<PH>).(?=YEH (<L>|<T>))-> IX:`
	Rules for DAMMA and KASRA follow the same logic for the FATHA.

`SHADDA, SUKUN:`
`.-> *`
	Shadda and Sukun symbols have no matching phonemes.

II. VOICEXML ELEMENTS

Element	**Purpose**
<assign>	Assign a variable a value
<audio>	Play an audio clip within a prompt
<block>	A container of (non-interactive) executable code
<catch>	Catch an event
<choice>	Define a menu item
<clear>	Clear one or more form item variables
<disconnect>	Disconnect a session
<else>	Used in <if> elements
<elseif>	Used in <if> elements
<enumerate>	Shorthand for enumerating the choices in a menu
<error>	Catch an error event
<exit>	Exit a session
<field>	Declares an input field in a form
<filled>	An action executed when fields are filled
<form>	A dialog for presenting information and collecting data
<goto>	Go to another dialog in the same or different document
<grammar>	Specify a speech recognition or DTMF grammar
<help>	Catch a help event
<if>	Simple conditional logic
<initial>	Declares initial logic upon entry into a (mixed initiative) form
<link>	Specify a transition common to all dialogs in the link's scope
<log>	Generate a debug message

<menu>	A dialog for choosing amongst alternative destinations
<meta>	Define a metadata item as a name/value pair
<metadata>	Define metadata information using a metadata schema
<noinput>	Catch a noinput event
<nomatch>	Catch a nomatch event
<object>	Interact with a custom extension
<option>	Specify an option in a <field>
<param>	Parameter in <object> or <subdialog>
<prompt>	Queue speech synthesis and audio output to the user
<property>	Control implementation platform settings.
<record>	Record an audio sample
<reprompt>	Play a field prompt when a field is re-visited after an event
<return>	Return from a subdialog.
<script>	Specify a block of ECMAScript client-side scripting logic
<subdialog>	Invoke another dialog as a subdialog of the current one
<submit>	Submit values to a document server
<throw>	Throw an event.
<transfer>	Transfer the caller to another destination
<value>	Insert the value of an expression in a prompt
<var>	Declare a variable
<vxml>	Top-level element in each VoiceXML document

II. PUBLICATIONS FROM THIS THESIS

1. Mohamed, Moustafa Elshafei, Mansour Alghamdi, Husni Almuhtaseb and Atef Alnajjar Ali, "Arabic Phonetic Dictionaries for Speech Recognition," *Journal of Information Technology Research*, vol. 2, no. 4, pp. 57-80, 2009.
2. Ali, Mohamed, Mustafa Elshafei, Mansour Alghamdi, Husni Al-Muhtaseb and Atef Al-Najjar, “Generation of Arabic Phonetic Dictionaries for Speech Recognition”. *The 5th International Conference on Innovations in Information Technology*. Alain, UAE, 16-18 December 2008.

NOMECLATURE

IVR	Interactive Voice Response: A computer system that interacts with users using voice commands and DTMF input.
DTMF	Dial Tone Multi-Frequency Signaling: Used to send Dialing tones (numbers) through the analog signal.
VoiceXML	Voice eXtensible Markup Langage: A standard for developing IVR applications that separates dialog logic design from other issues related to system development.

REFERENCES

[1] James R. Glass, "Challenges for spoken dialogue systems," in *IEEE Workshop on Automatic Speech Recognition & Understanding, ASRU*, 1999.

[2] Google goog-411. [Online]. http://www.google.com/goog411/

[3] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Dialog_system

[4] VoiceXML 2.0 Specifications. [Online]. http://www.w3.org/TR/voicexml20/

[5] HTML 5 Technical Report. [Online]. http://www.w3.org/TR/html5/

[6] SVG Standard. [Online]. http://www.w3.org/Graphics/SVG/

[7] Jahanzeb Sherwani and Roni Rosenfeld, "The Case for Speech and Language Technologies for Developing Regions," in *Proc. Human-Computer Interaction for Community and International Development workshop*, 2008.

[8] J. Rouillard, "Web services and speech-based applications," in *ACS/IEEE International Conference on Pervasive Services*, 2006, pp. 341 - 344.

[9] S. Kurkovsky, D. Strimple, E. Nuzzi, and K. Verdecchia, "Mobile Voice Access in Social Networking Systems," in *Fifth International Conference on Information Technology: New Generations (itng 2008)*, 2008, pp. 982 - 987.

[10] M. Brkic and M. Matetic, "VoiceXML for Slavic languages application

development," in *Conference on Human System Interactions*, 2008, pp. 147 - 151.

[11] Zhiyong Wu, H.M. Meng, Hongwu Yang, and Lianhong Cai, "Modeling the Expressivity of Input Text Semantics for Chinese Text-to-Speech Synthesis in a Spoken Dialog System," in *IEEE Transactions on Audio, Speech, and Language Processing*, 2009, pp. 1567 - 1576.

[12] Tobias Heinroth, Dan Denich, and Alexander Schmitt, "OwlSpeak - adaptive spoken dialogue within Intelligent Environments," in *8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2010, pp. 666 - 671.

[13] VoiceXML Forum. [Online]. http://www.voicexml.org/

[14] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Voice_browser

[15] Voice Browser Group. [Online]. http://www.w3.org/Voice/

[16] Speech Recognition Grammar Specification Version 1.0. [Online]. http://www.w3.org/TR/speech-grammar/

[17] Semantic Interpretation for Speech Recognition (SISR) Version 1.0. [Online]. http://www.w3.org/TR/semantic-interpretation/

[18] Speech Synthesis Markup Language (SSML) Version 1.0. [Online]. http://www.w3.org/TR/speech-synthesis/

[19] Pronunciation Lexicon Specification (PLS) Version 1.0. [Online]. http://www.w3.org/TR/pronunciation-lexicon/

[20] Voice Browser Call Control: CCXML Version 1.0. [Online]. http://www.w3.org/TR/ccxml/

[21] M. Elshafei, "Toward an Arabic text-to-speech system," *The Arabian J. Science and Engineering*, vol. 4, no. 16, pp. 565-583, 1991.

[22] M. Afify, R. Sarikaya, H-K J. Kuo, L. Besacier, and Y. Gao, "On the Use of Morphological Analysis for Dialectal Arabic Speech Recognition," in *Interspeech-2006*, Pittsburg PA, September 2006.

[23] Katrin Kirchhoff, Dimitra Vergyri, Jeff Bilmes, Kevin Duh, and Andreas Stolcke, "Morphology-based language modeling for conversational Arabic speech recognition," *Computer Speech & Language*, vol. 20, pp. 589-608, 2006.

[24] Fahad A.H. Al-Otaibi, *Speaker-Dependant Continuous Arabic Speech Recognition*, King Saud University, 2001.

[25] K. Kirchhoff et al., "Novel Approaches to Arabic Speech Recognition: Report from the 2002 John-Hopkins Summer Workshop," in *ICASSP*, 2003, pp. I-344-I347.

[26] J. Billa et al., "Audio indexing of Arabic broadcast news," in *ICASSP '02*, vol. 1, 2002, pp. 5-8.

[27] S.H. El-Ramly, N.S. Abdel-Kader, and R. El-Adawi, "Neural networks used for

speech recognition," in *Proceedings of the Nineteenth National Radio Science Conference, NRSC*, 2002, pp. 200-207.

[28] Y.A Al-Otaibi, "High performance Arabic digits recognizer using neural networks," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 1, 2003, pp. 670- 674.

[29] A.M. Ahmad, S. Ismail, and D.F. Samaon, "Recurrent neural network with backpropagation through time for speech recognition," in *International Symposium on Communications and Information Technology, ISCIT 2004*, vol. 1, 2004, pp. 98-102 vol.1.

[30] H. Bahi and M. Sellami, "Combination of vector quantization and hidden Markov models for Arabic speech recognition," in *ACS/IEEE International Conference on Computer Systems and Applications*, 2001, pp. 96-100.

[31] H. Bahi and M. Sellami, "A hybrid approach for Arabic speech recognition," in *ACS/IEEE International Conference on Computer Systems and Applications*, 2003, pp. 14-18.

[32] M. Shoaib et al., "A novel approach to increase the robustness of speaker independent Arabic speech recognition," in *7th International Multi Topic Conference, INMIC*, 2003, pp. 371-376.

[33] F.A. Elmisery, A.H. Khalil, A.E. Salama, and H.F. Hammed, "A FPGA-based HMM for a discrete Arabic speech recognition system," in *Proceedings of the 15th*

International Conference on Microelectronics, ICM, 2003, pp. 322-325.

[34] Bowen Zhou, Yuqing Gao, J. Sorensen, D. Dechelotte, and M. Picheny, "A hand-held speech-to-speech translation system," in *IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU*, 2003, pp. 664-669.

[35] M. Nofal, N.S. Abdel Kader, E. Abdel-Raheem, and M. El Henawy, "Arabic automatic segmentation system and its application for Arabic speech recognition system," in *IEEE International Symposium on Micro-NanoMechatronics and Human Science*, vol. 2, 2003, pp. 697-700.

[36] Mohamed Ali, Moustafa Elshafei, Husni Al-Muhtaseb, and Mansour Al-Ghamdi, "Automatic Segmentation of Arabic Speech," in *Workshop on Information Technology and Islamic Sciences*, Riyadh, 2007.

[37] Bing Xiang, Kham Nguyen, Long Nguyen, R. Schwartz, and J. Makhoul, "Morphological Decomposition for Arabic Broadcast News Transcription," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, vol. 1, 2006.

[38] A. Messaoudi, J.-L. Gauvain, and L. Lamel, "Arabic Broadcast News Transcription Using a One Million Word Vocalized Vocabulary," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, vol. 1, 2006.

[39] M.J.F. Gales et al., "Development of a phonetic system for large vocabulary Arabic speech recognition," in *IEEE Workshop on Automatic Speech Recognition &*

Understanding, ASRU, 2007, pp. 24-29.

[40] F. Diehl, M.J.F. Gales, M. Tomalin, and P.C. Woodland, "Phonetic pronunciations for arabic speech-to-text systems," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2008, pp. 1573-1576.

[41] H. Soltau et al., "The IBM 2006 Gale Arabic ASR System," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, vol. 4, 2007, pp. IV-349-IV-352.

[42] Mohamed Noamany, Thomas Schaaf, and Tanja Schultz, "Advances in the CMU-InterACT Arabic Gale Transcription System," in *Proceedings of the HLT/NAACL*, Rochester, NY, US, 2007.

[43] GALE project at NIST. [Online]. http://www.nist.gov/speech/tests/gale/

[44] B. Kosayba, H. Alkhedr, F. Jdeed, A. Shriedi, and E. Al-mozaien, "Arabic Phonetic Web Sites Platform Using VoiceXML," in *3rd International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA*, 2008, pp. 1-8.

[45] A. Emami and L. Mangu, "Empirical study of neural network language models for Arabic speech recognition," in *IEEE Workshop on Automatic Speech Recognition & Understanding, ASRU*, 2007, pp. 147-152.

[46] M. Afify, O. Siohan, and R. Sarikaya, "Gaussian Mixture Language Models for Speech Recognition," in *IEEE International Conference on Acoustics, Speech and*

Signal Processing, ICASSP 2007, vol. 4, 2007, pp. IV-29-IV-32.

[47] Z. Zemirli, "ARAB_TTS: An Arabic Text To Speech Synthesis," in *IEEE International Conference on Computer Systems and Applications,* March 8, 2006, pp. 976- 979.

[48] V. Kolias, C. Kolias, I. Anagnostopoulos, G. Kambourakis, and E. Kayafas, "A Speech-Enabled Assistive Collaborative Platform for Educational Purposes with User Personalization," in *Third International Workshop on Semantic Media Adaptation and Personalization*, 2008, pp. 157-163.

[49] A. A. Azeta, C. K. Ayo, A. A. Atayero, and N. A. Ikhu-Omoregbe, "A Case-Based Reasoning approach for speech-enabled e-Learning system," in *2nd International Conference on Adaptive Science & Technology (ICAST),* 2009, pp. 211-217.

[50] Zhang Long, Wang Jianhua, and Shan Linlin, "Research on caching strategy in a VoiceXML-based Mobile Learning System," in *4th International Conference on Computer Science & Education, ICCSE*, 2009, pp. 1718 - 1723.

[51] Liu Wenyun and Wang Jinglei, "The Application of VoiceXML to E-commerce," in *International Forum on Information Technology and Applications, IFITA*, 2009, pp. 357 - 359.

[52] Min-Jen Tsai, "The VoiceXML dialog system for the e-commerce ordering service," in *Proceedings of the Ninth International Conference on Computer Supported Cooperative Work in Design*, 2005, pp. 95 - 100.

[53] A. Sharp and S. Kurkovsky, "VOICE: A Framework for Speech-Based Mobile Systems," in *21st International Conference on Advanced Information Networking and Applications Workshops, AINAW*, 2007, pp. 38 - 43.

[54] C. Eccher et al., "On the usage of automatic voice recognition in an interactive Web based medical application," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '03*, 2003, pp. 289-292.

[55] Cristiana Armaroli et al., "An architecture for a multi-modal Web Browser," in *Seventh International Conference on Spoken Language Processing*, 2002, pp. 2553-2556.

[56] Zhiyan Shao, R. Capra, and M.A. Perez-Quinones, "Transcoding HTML to voiceXML using annotation," in *15th IEEE International Conference on Tools with Artificial Intelligence*, 2003, pp. 249 - 258.

[57] P.J.A. Reusch, B. Stoll, and D. Studnik, "VoiceXML-Applications for E-Commerce and E-Learning," in *IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems*, 2005, pp. 709-712.

[58] S. Kurkovsky, D. Strimple, E. Nuzzi, and K. Verdecchia, "Convergence of Web 2.0 and SOA: Taking Advantage of Web Services to Implement a Multimodal Social Networking System," in *11th IEEE International Conference on Computational Science and Engineering - Workshops*, 2008, pp. 227-232.

[59] Dongzhou Lian, Junwei Ge, Ying Xia, Zhengwu Yuan, and Heayoung Bae,

"Research of Voice Location Service Method Based on VoiceXML," in *8th International Conference on Signal Processing*, 2006.

[60] Hyeong-Joon Kwon and Kwang-Seok Hong, "A Design of User-Initiative Voice Web Using RSS and VoiceXML," in *5th ACIS International Conference on Software Engineering Research, Management & Applications, SERA*, 2007, pp. 281-288.

[61] Chia-Sheng Tsai, Ge-Ful Yang, F. C.-D. Tsai, and Ming-Hui Jin, "An Instant Messaging with Google Talk Handheld Devices Based on WEB2.0 for Tourist Call for Assistance," in *Proceedings of the Ninth International Conference on Computer Supported Cooperative Work in Design*, 2009, pp. 385-386.

[62] Juin-Ching Lin and Yao ming Yeh, "A Research on Telephone-Supported Multimodal Accessible Website," in *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, SUTC*, vol. 2, 2006, pp. 118-123.

[63] S. H. Maes, "A VoiceXML framework for reusable dialog components," in *Symposium on Applications and the Internet, SAINT*, 2002, pp. 28-30.

[64] W3C. (26 April 2000) Reusable Dialog Requirements for Voice Markup Language. [Online]. http://www.w3.org/TR/reusable-dialog-reqs

[65] O. Pietquin and T. Dutoit, "Aided design of finite-state dialogue management systems," in *International Conference on Multimedia and Expo, ICME*, vol. 3, 2003, pp. 545-548.

[66] D. Oria and A. Vetek, "Automatic generation of speech interfaces for Web-based applications," in *IEEE Workshop on Automatic Speech Recognition and Understanding*, 2005, pp. 397-402.

[67] Jun Kong, "Browsing Web Through Audio," in *IEEE Symposium on Visual Languages and Human Centric Computing*, 2004, pp. 279-280.

[68] J.A. Quiane Ruiz and J.R. Manjarrez Sanchez, "Design of a VoiceXML gateway," in *Proceedings of the Fourth Mexican International Conference on Computer Science, ENC*, 2003, pp. 49 - 53.

[69] H. Meng et al., "Bilingual Chinese/English voice browsing based on a VoiceXML platform," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '04*, vol. 3, 2004.

[70] W3C. Speech Recognition Grammar Specification Version 1.0. [Online]. http://www.w3.org/TR/speech-grammar

[71] Daniel Jurafsky and James H. Martin, *Speech and Language Processing*, Prentice-Hall, Inc., 2008.

[72] L. Rabiner and B.H. Juang, *Fundamentals of Speech Recognition*, Prentice Hall, 1993.

[73] L. Rabiner, J. Wilpon, and B. Juang, "A segmental K-means training procedure for connected word recognition," *AT&T Technical Journal*, vol. 64, pp. 21-40, 1986.

[74] X. Huang, A. Acero, and H. Hon, *Spoken Language Processing*, Prentice Hall PTR, 2001.

[75] Adobe Systems. (2011) Real-Time Messaging Protocol (RTMP) specification. [Online]. http://www.adobe.com/devnet/rtmp.html

[76] FFmpeg Project. [Online]. http://ffmpeg.org/

[77] infrared5. Red5 Server. [Online]. http://www.red5.org/

[78] Xuggle, Xuggler. A Java based media encoder. [Online]. http://www.xuggle.com/xuggler/

[79] JVoiceXML. [Online]. http://jvoicexml.sourceforge.net/

[80] CMUSphinx. [Online]. http://www.cmusphinx.org

[81] MBROLA project. [Online]. http://tcts.fpms.ac.be/synthesis/

[82] Acapela TTS. [Online]. http://www.acapela-group.com/develop-with-text-to-speech.html

[83] Amazon Simple DB. [Online]. http://aws.amazon.com/simpledb/

[84] William B Sanders, *Learning Flash Media Server 3,* O'Reilly Media, Inc., 2008.

VITA

Mohamed Ali received his B.S. in 2005 in Computer Science from King Fahd University of Petroleum and Minerals, Saudi Arabia, with GPA of 3.92 on 4.0 scale. He has diverse programming skills in Java, C#, PHP, C++, Java SE and EE using NetBeans, LINQ using Visual Studio.net, Scripting Languages: Python, Perl, Photoshop scripting, Microsoft Power Shell and Shell scripting. He participated in a number of research projects e.g., a Speech recognition system based on Gaussian Neural Networks, an Online Handwriting Recognition system using support vector machines, and a study of protein sequence analysis in Bioinformatics.

Nationality: Egyptian

Present Address: 22nd St., Al-Akrabia, Al-Khobar, Saudi Arabia

Permanent Address: 20 Ain Shams St., Helmeyeth El-Zayton, Ain Shams, Cairo, Egypt

Email Addresses: mohamed-a@live.com, antivirus.cs@gmail.com

Telephone: +966-03-8695177

Mobile: +966-530-717526