uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Yanyan Ju

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)

GRADE / DEGREE

Systems Science

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Voice-enabled Click and Dial System

TITRE DE LA THÈSE / TITLE OF THESIS

Tet Yeap

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Ahmed Karmouch

Yongyi Mao

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

VOICE-ENABLED CLICK AND DIAL SYSTEM

YANYAN JU

THESIS SUBMITTED TO THE
FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SYSTEM SCIENCE

SCHOOL OF MANAGEMENT
SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING
UNIVERSITY OF OTTAWA

SUPERVISED BY PROF TET YEAP

Your file Votre référence
ISBN: 0-494-11307-3
Our file Notre référence
ISBN: 0-494-11307-3

DECLARATION

I THE UNDERSIGNED HEREBY DECLARE THAT THE WORK CONTAINED IN THIS THESIS IS MY OWN ORIGINAL WORK AND HAS NOT PREVIOUSLY IN ITS ENTIRETY OR IN PART BEEN SUBMITTED AT ANY UNIVERSITY FOR A DEGREE

YANYAN JU

18 JAN 2005

ACKNOWLEDGEMENTS

I would like to express my gratitude to a very long list of people who supported me. A big thank you goes to:

- Professor Tet Yeap who sparked my interest in this research, and thank for his encouragement, patience and effort on my paper.
- My family, especially my parents for instilling in me the passion to learn and my husband Juchuan Song for his long-lasting understanding and support through the years.
- My friends, especially Monica for her support and assistance, and colleagues for their support, love and patience.

ABSTRACT

This thesis is targeted at proposing a framework for building a voice-enabled web application which allows users to interact with the system through the Internet, using either speech (microphone, speaker) or traditional devices (such as keyboard or mouse). Several voice-related technologies are briefly described in the beginning of this thesis. The original design of the Click&Dial system was modified and enhanced in order to adopt a popular J2EE framework named Struts. Compared with SALT, XHTML plus Voice and Voice XML, SALT is chosen as the main technology for building the speech-enabled web application. The architecture of integrating voice into the web application is presented, and the framework is implemented. With the help of it, the speech-based components are demonstrated to be embedded easily into the system, and the simplicity of the migration is guaranteed as well. The proposed system, with both visual interface and voice interface, is designed and developed using Java-based technology combined with embedded speech engines. The implementation successfully verifies that the proposed framework meets the design requirements and is capable of supporting two kinds of user interfaces for a web application, and the targeted objective is achieved. Additionally, a possible solution for voice-activated Click&Dial system on Pocket PCs is recommended, which enables the voice-enabled application to be accessed easily from anywhere using hand-held devices in the future.

CONTENTS

List of Tables

List of Figures

List of Listing

List of Acronym

ASCII	American Standard Code for Information Interchange
ASR	Automatic Speech Recognition
BNF	Backus-Noir Form
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
CPU	Central Processing Unit
CSS	Cascade Style Sheet
CVS	Concurrent Versions System
DOM	Document Object Model
DTMF	Dual Tone Multi-Frequency
DSP	Digital Signal Processing
EJB	Enterprise Java Beans
EIS	Enterprise Information System
FIA	Form Interpretation Algorithm
GUI	Graphical User Interface
HMM	Hidden Markov Model
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocal
IDE	Integrated Development Environment
J2EE	Java 2 Platform Enterprise Edition
J2ME	Java 2, Micro Edition
JAAS	Java Authentication and Authorization Service
JDBC	Java Database Connectivity
JDK	Java Development Kit
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
JSTL	JSP Standard Tag Library
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LTS	Letter-To-Sound
MMWG	Multimodal Working Group

MVC	Model-View-Controller
NLP	Natural Language Processing
PBX	Private Telephone Network
PDA	Personal Digital Assistant
PSTN	Public Switched Telephone Network
RPS	Requests Per Second
SALT	Speech Application Language Tag
SASDK	Speech Application SDK
SES	Speech Engine Services
SML	Semantic Markup Language
SSML	Speech Synthesis Markup Language
TTFB	Time To First Byte
TTLB	Time to Last Byte
TTS	Text-to-Speech
UML	Unified Modeling language
URL	Uniform Resource Locater
VoiceXML	Voice Extensible Markup Language
VoIP	Voice over IP
VUI	Voice User Interface
W3C	World Wide Web Consortium
WAS	Web Application Stress
WWW	World Wide Web
X+V	XHTML plus Voice
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

Chapter 1 Introduction

1.1 Motivation

The World Wide Web (WWW) is a huge source of information to which as many people as possible should have access [1]. As computers become more universally available to people, users are not satisfied with interacting with computers using keyboards or mice only. The traditional operation of computers has some problems for certain people such as disabled persons. In the meantime, with the explosive growth of portable devices such as pocket PCs and cell phones, there is a growing demand for technology that allows users to be connected to the Internet from anywhere through a non-traditional keyboard, mouse and monitor [2]. Our goal is to find a way to allow users to access and interact with the web anywhere without a mouse in hand. Obviously, speech is an alternative. Using speech technology provides more convenience to the traditional browser-based applications. With the help of it, it is not a dream anymore to carry on a conversation between a computer or other devices and people. Integrating voice activities into the web application is our major goal in this thesis.

Speech technology is rapidly emerging as the user interface of alternative for advanced web applications. Voice-enabled applications incorporate Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) to provide an efficient, convenient and friendly voice user interface. [3] Several software standards groups aim to drive the market of voice-enabled web applications and create a common specification based on existing Internet standards [4]. Choosing appropriate software tools allows us to build a more stable, flexible application and simplify the development and deployment of the voice-enabled application greatly.

Speech Application Language Tag (SALT), as an extension of standard web technology for voice applications, is adopted as the application's voice platform. It is aimed at revolutionizing the Internet industry by providing voice access to the web application, and leveraging the existing web infrastructure at the same time [5].

However, there is a critical problem that needs to be solved. The existing system is a Java-based application, but Sun Microsystems' Java Development Kit (JDK)[6] does not

support speech at present. We do not want to give up the whole system and rebuild a new one. Therefore, we have to find a framework that allows seamlessly integrating the voice interface into the existing system by embedding a speech engine provided by other company. Meanwhile the old system needs to be modified to meet the new requirements.

1.2 Contributions

In this thesis, a new framework is proposed, which allows integrating speech activity into the web-based application. We adopt the structure of the Java-based system, which enables "write once, run anywhere" [6], with embedding speech engines provided by Microsoft. Therefore, the dominant benefit of crossing the platform is kept, and Microsoft voice is taken advantage of. The major contributions related to the proposed framework are outlined briefly as follows:

- The Click & Dial system for the visual interface based on this framework is analyzed and redesigned by using Unified Modeling language (UML).
- We analyzed and designed the Click & Dial system for the voice interface based on this framework to provide a more stable and friendlier voice interface to users.
- The framework of voice-enabled application for Pocket PC is proposed.
- The required software, installation of software and setting of environment are summarized briefly.
- The back end of the system is built in Java and the voice interface is implemented using SALT as markup language.
- The scalability, functionality and performance of the implementation based on the proposed framework are tested and evaluated by using the Microsoft's Web Application Stress (WAS) tool.

1.3 Outline of the Thesis

Chapter 2 presents the overview of the web-based Click&Dial system from the Use Case View. The current three principal approaches for creating speech applications, VoiceXML, SALT and X+V, are introduced and compared by listing both the advantages

and the disadvantages. Two core speech technologies, Speech Synthesize (TTS) and Speech Recognition, are briefly introduced as well.

Chapter 3 shows the infrastructure of the speech-enabled Click&Dial system. Each component, both web-related and voice-related, is described in detail. The general speech interface framework is presented to indicate how to handle the speech process through the speech engine services.

Chapter 4 outlines the key design considerations for redesigning the Click&Dial web application. UML artifacts, such as the class diagrams, the sequence diagrams and the component diagram, are used to describe the design models following the Model-View-Controller (MVC) architectural approach. A set of the core design patterns we applied to the system is introduced as well.

Chapter 5 presents the detailed design of the SALT-based speech-enabled user interface. Three different structures of the SALT-based system for different scenarios are discussed separately based on the proposed framework. According to the design principles of the voice user interface, several designs, including navigation design, recognition mode, prompt of SALT, initiative model and grammar design, are discussed in detailed. The main function of this system, search, is discussed in depth as a typical example.

Chapter 6 describes the implementation of the speech-enabled Click&Dial system by seamlessly integrating voice and visual interfaces. The essential technologies for implementing the system and the tools for building the system are introduced as well.

Chapter 7 discusses the technologies, methodologies and tools used for evaluating the performance of the system through analyzing the test results.

Chapter 8 concludes this paper. Meanwhile, some future work is suggested for further research.

Chapter 2 Background

In order to integrate the voice into the visual user interface for the Click&Dial system, we should be familiar with the existing system first. Then the speech technologies, i.e., Speech Synthesize (Text-To-Speech) and Speech Recognition, are briefly discussed to show how they work. The three major approaches for building voice-enabled web applications are introduced as well.

2.1 Click and Dial System

2.1.1 Introduction

The Click and Dial System is a web application, which is initiated and developed by Bell Corporate Security Division. It allows online users to search the contact numbers of any employee in the corporate database and launch an online call to that person by selecting one of the contact numbers. Corporate users can use the Click&Dial system via different types of devices like desktop computers, PDAs, etc.

2.1.2 Use Case Views

Use Case is a standard way to express the requirement in UML [7]. Basically it defines who needs to do what with the software.

The following actors will interact with the Click&Dial system:

ID	Actor Name	Description
1.	System Admin	This role has administrative rights, such as registration.
2.	Client	Normal users of the system

Table 2-1 List of Actors

The following diagram shows the Use Cases in the current system:

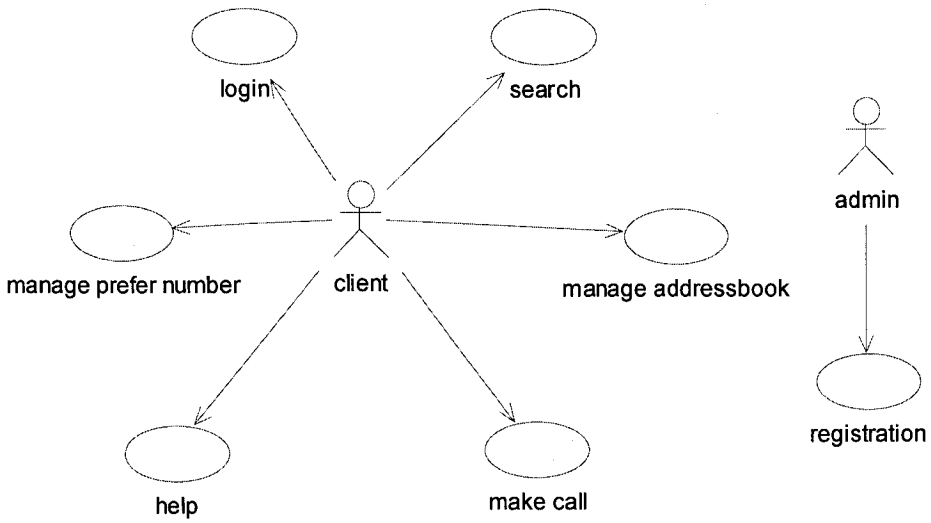

Figure 2-1 Use Cases of Click&Dial System

The descriptions are discussed in detail as follows:

Login

When a user types the URL of the Click&Dial web application in the Internet Explorer's address bar, a login page will be displayed to ask the user's Bell PEIN number and password. After user enters his or her PEIN number and password, the authentication operation is executed to verify the user's identity.

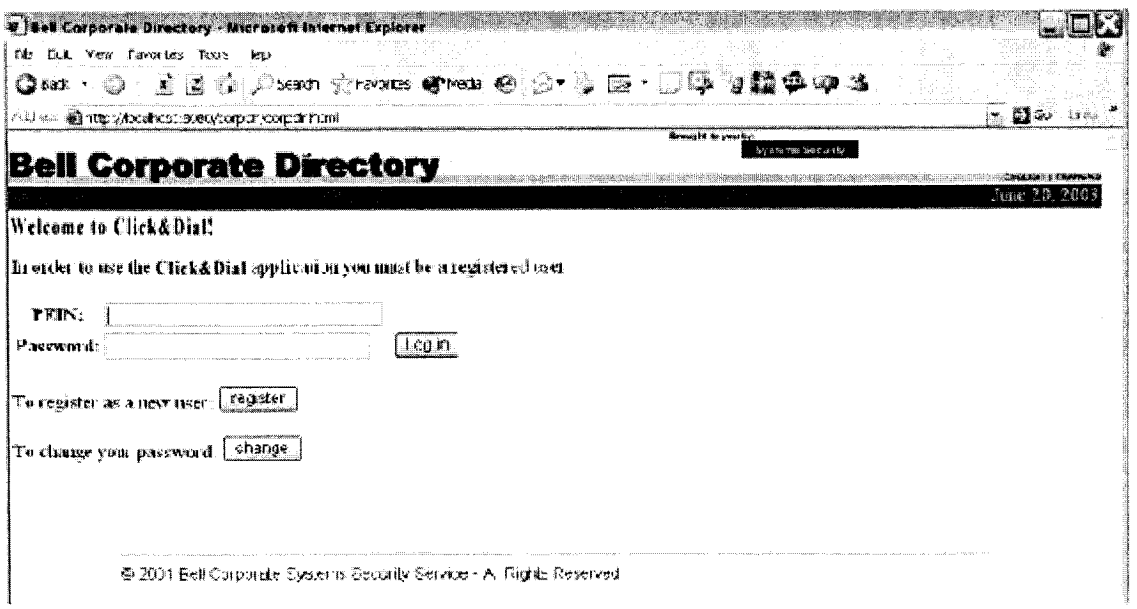

Figure 2-2 Click&Dial Login Page

Search

If the authentication is successful, the search page is displayed as shows in the following figure. Searching could be accomplished by entering the first name, last name, Bell PEIN number or their combination. For example, if the user wants to call his colleague whose name is "Mike Zhang", he enters "Mike" as the first name and "Zhang" as the last name, clicks the "Search" button, or the user can just type the first letter of the first name and click the Search button; the application will pursue an LDAP search for the people whose first name begins with M, then display them.

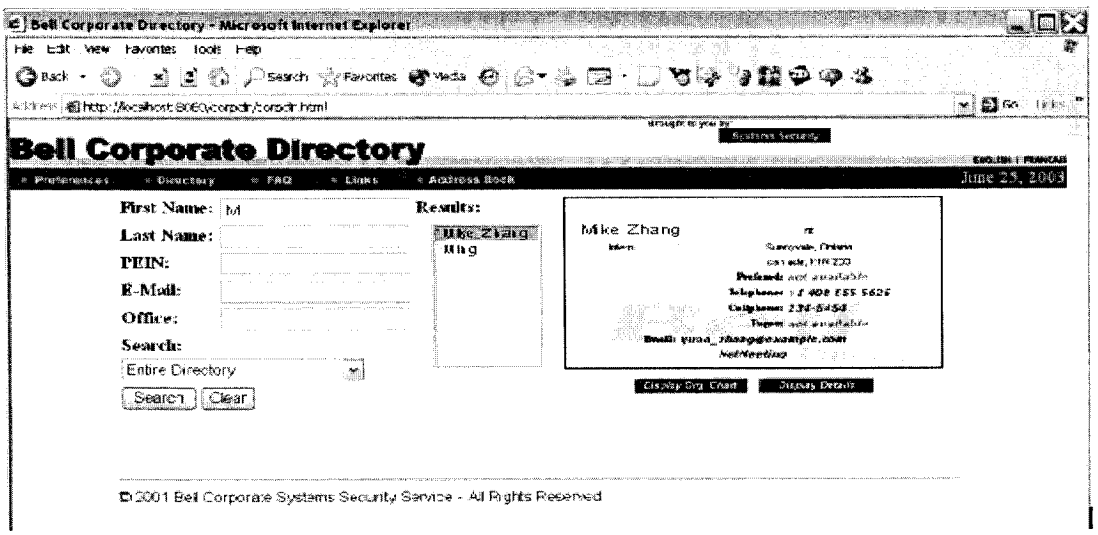

Figure 2-3 Click&Dail Search Page

Manage Address Book

The Click&Dial system allows users to manage the address book. The capacity of the address book list is 5, which are the last 5 numbers called using this application. And it can store people's Bell PEIN number that we call more often.

Help

If the user has any questions when he uses this system, he can click Help to get the information about how to use this system. Some Bell related sites are also provided in the page.

Make Call

At the right side of the above picture there is an e-card showing the detailed information of "Mike Zhang". The user then clicks on the telephone number, which shows as a link on the e-card. The application will invoke a C compiled file to launch a call from the user's preferred number, which is his/her desk phone, to Mike's phone number. The user's desk phone rings. When the user picks up his/her phone, Mike's phone rings. Thus the call has been established.

Registration

The admin can use this web page to create a new user and register him in the system. After the user is registered, he can log in to the system and use the other functions.

Security

The application's security requirement basically has two main functions: authentication and authorization.

- Any user must login before he can use the functions on the web page.
- Since there are two roles in the system, admin and normal client, the normal client cannot access and use the admin function like registering the new user. If he types the URL directly on to the admin web pages, he will be redirected to the login page and asked to login as admin account.

The existing Click&Dial system is a traditional web application. However, our goal is to build a voice-enabled web application system, which means extending the traditional user interface by integrating a voice user interface. Therefore, the user can interact with the application by speaking.

In order to successfully build the voice user interface (VUI), we must know the current speech technologies - two core technologies: Speech Synthesize and Speech Recognition.

2.2 TTS

2.2.1 Introduction of TTS

Text-To-Speech (TTS) is a computer-based system, which is able to 'read' the text by converting the text into voice output [8]. Generally, there are two types of speech output systems: interactive systems and non-interactive systems [9]. Obviously, the Text-To-Speech system is the interactive system that is able to generate the synthesized speech according to the input text.

The amount of vocabulary the TTS system can synthesize divides it into two systems: the restricted and unrestricted systems [9]. A restricted system can merely handle limited words that were stored in advance. On the other hand, an unrestricted system is expected to be able to operate any text input. The aim of our system is to generate all the sentences according to the input texts that may be unlimited, not only some isolated words or parts of sentences, so the unrestricted TTS system was adopted in this project. The goal of the TTS synthesizer is to make the synthesized speech as natural and intelligible as human speech. Unlike the restricted system, which stores several words, it is impossible to store all the words of language for TTS. So the TTS process and the substance stored in the database are extremely important for the TTS system.

In addition, TTS could also be monolingual or multilingual [8] - it can handle one desired language or more than one language together, respectively. In our project, we only consider the monolingual system for English rather than the complicated system, based on our requirements.

Since the TTS systems may differ in their internal processing, I will present the representative TTS systems in greater detail below.

2.2.2 TTS Process

Basically, the Text-To-Speech synthesis system is divided into three components: Natural Language Processing (NLP), Digital Signal Processing (DSP) and Letter-To-Sound rules [9,10], text database as shown in Figure [2-4].

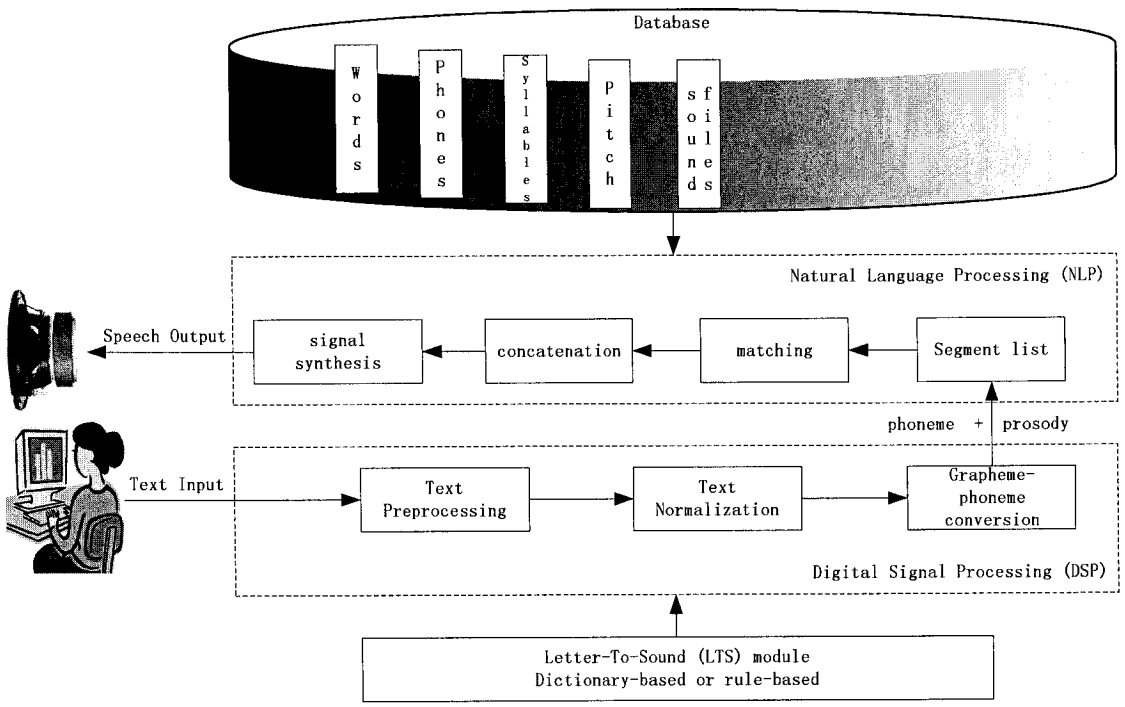

Figure2-4 TTS Process [10]

2.2.3 Natural Language Processing (NLP)

The goal of Natural Language Processing is to convert the input text into a phonetic transcription with the desired prosody (including rhythm speech rate, stress and intonation) [10]. However, the goal of Digital Signal Processing is to transform the phonemes or prosody signal information into the output speech corresponding to the desired target.

The Natural Language Processing model consists of several components shown as follows: text pre-processing, text normalization and grapheme-to-phoneme conversion, in turn.

2.2.4 Text Pre-processing

Text pre-processing is the very beginning of the synthesis process, in which a broad segment of texts presented as sets of ASCII characters are transformed into lists of word segments, which are manageable. Meanwhile, numbers, abbreviations, acronyms and idiomatic expressions are identified in this processing [10].

- Sentence end detection determines where to pause or stop for speaking. Obviously, punctuation could be the termination mark for sentences. But the complexity of this process increases as the punctuation is used in mathematical form embedding in the sentences that may be ambiguous. Therefore, it is important to separate the punctuation marks from the words.
- Numbers detection classifies numbers into several styles, such as dates, time and rational numbers, and converts them into text form according to the rewriting rules.
- Abbreviations and acronyms recognition. Generally, abbreviations are expanded to full words; thus the rules to be in charge of the acceptation for abbreviations are stored in a database. On the contrary, it is unnecessary to store most acronyms in the database since the full words for acronyms do not need to be pronounced.

In some cases, text pre-processing is integrated with the morph syntactic analyzer.

2.2.5 Text Normalization

The text normalization process is composed of two modules: the morphological analysis module and the contextual module [11].

The goal of the morphological analyzer is to present all possible categories for each word generated from text preprocessing. There are two fundamental categories for words. One is function word concatenating the sentences, such as conjunctions or determiners that are meaningless. The other is content word appearing in a variety of forms that makes up the meaning of the sentence. The morphological analyzer distinguishes the inflected, derived and compound words corresponding to the specific rules [10], and then translates them into their morphs.

Distinguishing the form of word segments will reduce the size of the lexicon stored in the database. For instance, inflectional morphology changes a word into different forms by applying affixes (both prefixes and suffixes) from lexeme stems. In fact, this transformation does not change the word category. Therefore, only the lexeme stems and the affixes group are stored, rather than the full form of the words. As opposed to inflection, derivational morphology creates a new word by changing the category of the

given word. On the other hand, compounding morphology creates a new word by combining two or more lexemes following the syntax of the language.

After finishing the morphological analysis, there is still more than one possible category that needs to be reduced via contextual analysis module, as desired. Two basic methods are implemented to solve this problem: N-grams [8], as a particular type of Markov model to estimate the probability of a string of words, and local non-stochastic grammars [8,10]. The result of the contextual analysis module is that useless words are omitted from the possible categories.

2.2.6 Grapheme-phoneme Conversion

The output of grapheme-phoneme conversion is a phonological representation of the utterance in terms of prosodic groups.

As showed in the figure, Letter-To-Sound (LTS) is in charge of the transformation from input text to phonemes [10], which provide rules or grammars to restrict the internal process. Two principal strategies are adopted in this transcription system, which are dictionary-based and rule-based [12].

As the term suggests, the performance of transcription is like looking up a word in the dictionary where the database acts as the dictionary in the synthesis system. The main problem for this solution is keeping the size of the dictionary as small as possible and maintaining its function as powerful as possible. It is impossible to store all forms of words, but part of them may be stored, combined with inflectional, derivational and compounding morphological rules.

Another method used by the LTS module is the rule-based strategy, which transfers phonological competence into letter-to-sound rules, storing only exceptions in the lexicon, rather than a maximum of phonological knowledge into a lexicon [10,12].

2.2.7 Digital Signal Processing (DSP)

Digital Signal processing takes the symbolic linguistic representation from text analysis and converts it into actual sound output. Typically, four components are involved in this process. There are two main technologies use for generating synthetic speech output in TTS system: rule-based synthesis and concatenative synthesis [13].

Rule-based synthesis does not use samples of human speech; it uses a series of rules to create the output synthesized speech. It keeps a powerful approach to speech synthesis that can be rather intelligible. Sometimes this technology is called formant synthesis [13]. Concatenative synthesis is based on the concatenation of segments of recorded speech stored in a speech segment database to create synthysized speech.

The output of natural language processing, phonemes combined with prosody, is a target phonological utterance to digital signal synthesis. First, a series of segments is listed in a segment list generation. Once prosodic events have been correctly allocated to individual segments, the parameters are retrieved from the database. Then the prosody matching module adapts them according to the prosody. The selection algorithm is used to find the optimal chain of the segments. The segment concatenation module is responsible for dynamically matching by smoothing discontinuities while it considers a few concatenation criteria. Intuitively, the task of the last module in digital signal processing is to create speech output corresponding to the input combined with a set of parameters[10].

Included within the Microsoft speech server, the Speechify TTS[14] system is based on the concatenation mudule, which offer a far more human-like tone because it uses actual recorded speech. In order to improve the sound further, there are several factors we have to take into account, such as voice interface design, the code to invoke TTS and how to enhance TTS. The detailed design will be presented later.

2.3 ASR

2.3.1 Introduction of ASR

The Automatic Speech Recognition (ASR) engine converts the acoustic signal from user utterance into text. A number of significant milestones have been reached in the 1990s and 2000s. Since the performance of speech recognition has greatly improved, and due to the progress in microprocessors and digital signal processors, they no longer need special hardware [15]. Current research in speech recognition concentrates on the following fields: robust speech recognition against noise, advanced language models and multilingual speech recognition. [16].

2.3.2 ASR Process

The speech recognition process, which is the process of analyzing an acoustic signal to determine the words uttered by speakers [17, 18], contains several stages. A general speech recognition model is shown in figure [19]. The model begins with a user creating a speech signal with different input devices, such as a microphone, a pocket PC, a telephone and a Mobil. Since many different applications are involved, several recognition components must be developed and integrated. For example, the very first step for speaker recognition is to identify the speaker. It's quite useful for some voice implementation. As for language recognition, we are interested in identifying the language the user uses. Through the major components of speech recognition, a series of meaningful words is created from the speech signal. Through these three recognition components, pre-processing is finished. The recognized words are input into a higher level processing to achieve the meaning of words, language understanding. The meaning representation component is used to get the appropriate information in the form of text, tables and graphics [20, 21]. To minimize the errors for sequential inputs or to get sufficient information, discourse context is maintained and fed back to the speech recognition as well as language understanding.

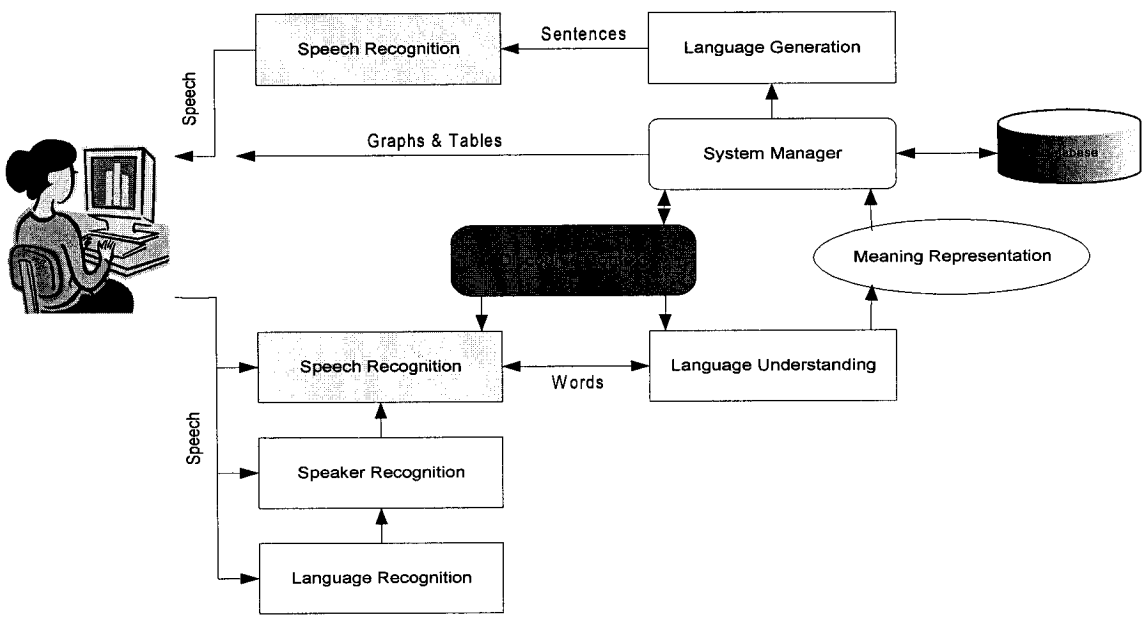

Figure 2-5 Speech Recognition Process [20, 21]

2.3.3 Recognition Mode

Based on the different applications, the speech recognition system can be classified into several different modes [16,22, 23]:

- **Isolated Word Recognition**: It is a relatively simple mode where the word boundary is quite clear so that it's unnecessary to find the beginning and the end of each word. For example, the English digits 0-9.
- **Continuous Speech Recognition**: It is a natural and user-friendly mode. Unlike discrete speech recognition, it allows the person to speak normally without pausing between every word.
- **Keyword Spotting**: It is a mode between isolated word recognition and continuous speech recognition. The main idea is to capture the keyword in a sentence rather than the whole sentence and then perform the corresponding command.
- **Speaker Dependent**: It is a system that has a training process. A specific speaker will be asked to train before using this system in order to get a high rate of recognition.
- **Speaker Independent**: The opposite of the speaker-dependent system, the independent mode is a system without a training process, and it can be used by different speakers.

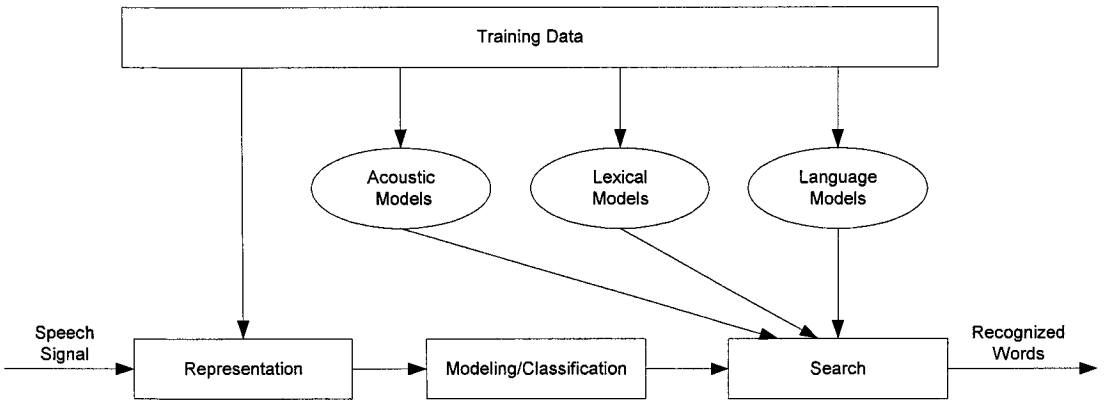

Figure 2-6 Components of Speech Recognition [21]

There are several main speech recognition algorithms, such as the statistical, knowledge-based and neural approaches [24]. In recent years, the algorithm based on statistical methods becomes most popular.

Refer to figure [2-6]; there are three components in a typical speech recognition model. HMM is used as the algorithm of these components, described as follows:

Acoustic Models

The acoustic model is the first step in the recognition of natural speech. With a given speech representation, the acoustic model will detect and classify possible acoustic patterns. In such statistical models, we use an inventory of fundamental probabilistic models of linguistic units to create word representations [21]. An HMM contains two stochastic processes, a hidden Markov chain and an observable process. A hidden Markov chain represents the changing statistical characteristics, which exist in the actual observations of speech signals, and the other process that describes the spectral variability is a double stochastic process [20]. This model combined with two stochastic processes is very powerful and flexible.

According to the structure of the elements of the transition matrix, HMM can be classified under several types. Considering the observations as discrete symbols, HMM uses a discrete probability density within each state of the model, namely discrete HMM. To overcome the limitations of discrete HMM, continuous density HMM is recommended [25]. Computation of probabilities with discrete models, however, is faster than with continuous models.

Lexical Models

The lexical model, namely vocabulary definition [26], is required, as the rules that determine what sequences of words are grammatically well formed. Networks of phonemes represent words and each path in a word network accounts for a pronunciation of the word [20]. If the same phoneme is pronounced in different contexts, it could have different acoustic distributions of observations. So for a given phoneme, how many allophones should be considered depends on many factors. The key factor is the

availability of enough training data. Each allophone model is a composition of states, transitions and probability distribution as an HMM model [21].

Language Models

The language model is the recognition system component to combine the syntactic constraints of the language [16]. Through the language model, the probability of a sequence of words is calculated. The word hypotheses can be generated in a single sequence of words, in a collection of the n-best word sequences or in a lattice of partially overlapping word hypotheses [20,21]. Since word boundaries may not be clear while transmitting the speech signal, word boundary detection would be a key factor to be considered in this hypothesization process.

Microsoft Speech Recognition Engine, which we use in our project, belongs to continuous speech recognition mode, which means the user can speak normally without pausing between every word. It is more natural and user-friendly. The details on how to design the grammar to invoke the speech recognition engine are presented later.

2.4 W3C Markup Language Interface

In the last few years, World Wide Web Consortium (W3C) has made an effort to make the languages standard to develop voice-enable web application to satisfy the increasing requirements of customers. Many approaches founded by the industry consortia were proposed to W3C. There are three principal approaches for creating speech applications: VoiceXML, SALT and X+V [27]. The goal of all these standard languages is to convert utterances to the special format that a computer can understand, to process it, and then send it back to client browser in either speech or text by using speech recognition and TTS (i.e., Speech Synthesis). This is discussed in greater detail below.

2.4.1 VoiceXML

VoiceXML [28], an XML-based markup language for creating voice driven application, has already been adopted by W3C [29]. In contrast to HTML, which is also a markup language for writing visual web applications, VoiceXML is a completely self-contained markup language for creating speech interfaces instead. Furthermore, it is a dialog-

oriented language that is originally designed to allow users to access the speech application through the telephone.

Voice application is not appropriate in scenarios where there is a lot of information to be displayed or listened to, but it is in situations where the key points or words are known by the user, which is quite simple. Since VoiceXML arose out of the requirements for the telephony application, it works properly with telephones, but not with other devices.

For instance, if the user wants to search for one person whose name is "Mike" using the Click&Dial system, he or she just speaks the name using the phone, and the result of the searching will return in speech. In such simple cases, the VoiceXML-based speech application will work well. But it is very difficult to deal with the case if there are more than ten results. VoiceXML-based voice-only application is not convenient for such complicated situations, and is even impossible to use.

As discussed above, a more user-friendly application that allows either data entry or speech input is expected by users. W3C has already established the Multimodal Working Group (MMWG) [27] for developing standards to extend the user interfaces. Mainly, there are two specifications available for multimodal interfaces: XHTML plus Voice [30] and Speech Application Language Tags [5].

2.4.2 XHTML plus Voice

Due to the limitation of VoiceXML, IBM, Motorola and Opera Software proposed XHTML plus Voice (X+V) for building multi-modal speech applications. It combines the features of five W3C standards in both telephony and web applications. The X+V specification uses XHTML to handle the graphical output and extends it for voice input and output by using VoiceXML. In fact, XHTML1.0 is the XML-based format for HTML4.0. XHTML1.1, however, is the modular form for XHTML1.0 [27]. This model allows voice dialogs to be added combined with the DOM2 event model, where voice dialogs are created by using the modularized version of VoiceXML.

X+V adopts all specifications defined by HTML4.0, and most VoiceXML events as well. In addition, X+V assumes filled events for fields to handle the event [30].

IBM has already developed X+V plug-ins for browser platforms by integrating the voice interfaces. But this technology is still under way and is not mature. Furthermore, it has

not gotten confirmation from W3C yet. So I did not adopt X+V for my project due to the drawbacks of this technology.

2.4.3 SALT

Another currently available specification for multimodal web application is Speech Application Language Tags (SALT), which is contributed by SALT forum. SALT leverages the event-based DOM model to integrate the multiple browsers [5]. Unlike VoiceXML, which provides dialog, speech browsers and call management, SALT tags contribute a set of XML to HTML. There are four major tags: <prompt> for speech output, <listen> for speech input, <dtmf> (dual-tone multifrequency) for telephone touch tone input, and <smex> for messaging and call control [5].

SALT can add speech dialog by inserting the SALT tags into the codes written by other markup languages, so it reuses the existing standards.

Based on the explanation above, all of these technologies have key benefits in certain applications, as well as drawbacks. The status and capabilities of these technologies, similarities and differences among them are listed in the table 2-2.

	VoiceXML	**SALT**	**X+V**
Status	VoiceXML 2.0 was released in October 2001	SALT specification 1.0 was released in July 2002	The latest version of XHTML+Voice Profiles was released in March 2003
Status at W3C	Recommended	Submitted	Note
Forum	VoiceXML Forum was founded in 1999	SALT Forum was founded in 2001	N/A
Members	So far, there are 376 companies are members of VoiceXML forum (e.g., AT&T, IBM,	So far, there are around 70 companies are members of SALT forum (e.g., Cisco System, Intel,	Three companies (IBM, Motorola, Opera Software) submitted this specification to W3C

	Lucent, Motorola, Nuance, HP)	Microsoft, Philips)	
Similarity			
Speech Recognition	W3C Speech Recognition Grammar	W3C Speech Recognition Grammar	W3C Speech Recognition Grammar
TTS	W3C Speech Synthesis Markup Language (SSML)	W3C Speech Synthesis Markup Language (SSML)	W3C Speech Synthesis Markup Language (SSML)
Mixed-initiative	Supported (both by user and system)	Supported (both by user and system driven)	Supported (both by user and system)
Difference			
Input	Speech, DTMF	Data, Speech, DTMF	Data, Speech, DTMF
Mode	Voice-only mode	Voice-only mode or multi-mode	Voice-only mode or multi-mode
Web Interface	Not supported	Combined with HTML	Combined with XHTML
Dialog control	Through Form Interpretation Algorithm (FIA)	Through script	Through Form Interpretation Algorithm (FIA)
Event handle	Dialog-based	Combined with script and DOM	XML-based
Devices	Telephone	Various devices: telephone, computer, PDA, cell phone with browser	Various devices: telephone, computer, PDA, cell phone with browser

Table 2-2 Comparisons of VoiceXML, SALT and X+V [5, 28, 29, 30, 31, 32, 33]

From the above table, it is noticeable that VoiceXML contributed by the VoiceXML forum, is the first standard, which is more mature than others. Surely there are more contributors or partners.

Similarly, they all use the standard Speech Recognition Grammar from W3C for speech recognition and the W3C Speech Synthesis Markup Language (SSML) for Text-To-Speech processing. Additionally, all of them support the mixed initiative interaction, which means the ability for both the user and the system to drive the conversation. But the control of mixed initiative dialogs is, to some extent, different for every specification.

VoiceXML only has voice-only mode, which does not support web interfaces, but speech interfaces. On the contrary, SALT and X+V support web interfaces, which have both voice-only mode and multi-mode.

Since VoiceXML is designed for voice-only applications, standard telephone is the only device for accessing this application. Therefore, it merely allows speech input or DTMF input. By contrast to VoiceXML, XHTML plus Voice and SALT are designed for use with multimodal applications, which allow not only speech input and DTMF input, but also data input via various devices from standard telephones and computers, to mobile devices, including pocket PCs, PDAs and Internet-enabled cell phones.

Controlling the dialog is another main difference among these technologies. VoiceXML and X+V allow developers to create dialogs that are interpreted by using the Form Interpretation Algorithm (FIA) in a web page [28, 30]. Just like the form of HTML, they can be interpreted by FIA without any dialog management logic. On the contrary, SALT needs to use script to control the dialogs.

Events provide a useful mechanism to handle exceptional conditions and application events as well. The types of event handlers used in these three technologies are pretty different. Event handlers are inherited through the various VoiceXML scopes that are dialog-based. SALT elements expose a DOM interface, so the event is controlled by the scripting module in the HTML page. In contrast to SALT and VoiceXML, X+V inherits the features of event handlers from XML since it is actually an XML-based format [27, 33]. Handling the events for VoiceXML and SALT is discussed in further detail below.

From the view of many observers, VoiceXML and SALT will coexist and may even merge together. "At the end of the day, VoiceXML may be the standard for pure voice

applications, and we may see a SALT-like language for multimodal that leverages existing voice standards [31]."

According to both advantages and disadvantages of these technologies and the view of many observers, we proposed using VoiceXML for the telephony version of the Click&Dial system and using SALT for the multimodal version of this system. In this project, we only focus on how to build the voice-enabled multimodal web application using SALT.

Chapter 3 Infrastructure of Voice-enabled Click&Dial System

This chapter presents the proposed framework for building the voice-enabled Click&Dial system according to our requirements.

3.1 Proposed Framework

How to seamlessly integrate the voice interface into the traditional web application is the key point for building our system. The proposed framework of a multimodal system is illustrated in figure [3-1], which is designed based on the general scenario.

The whole application can be separated into two major sub systems:

1. A web application based on J2EE technology: browser, web server, application server and database.
2. Voice related: speech server

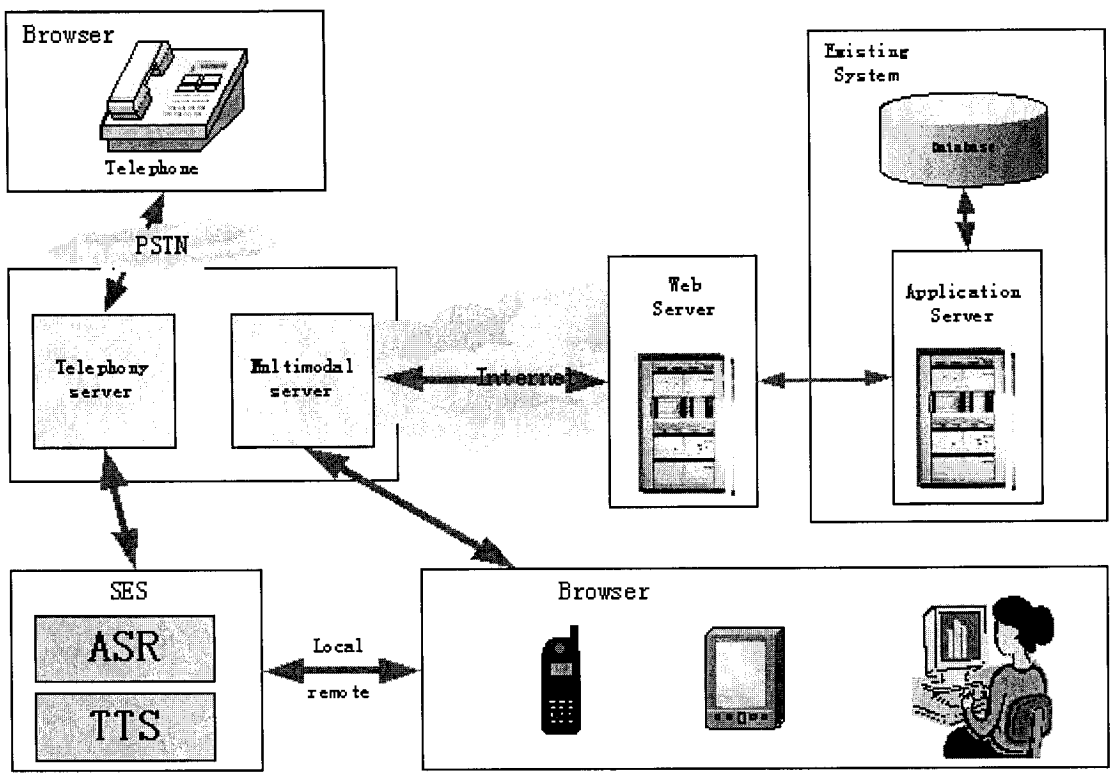

Figure 3-1 Framework of Click&Dial System

3.2 J2EE Web Application Components

3.2.1 Client Side

On the client side, there are two modules: the web browser and a voice user interface built in markup language. They are responsible for the simple processing, such as prompt dialog, text or voice input and displaying the web pages. The response from the web server or the speech server is displayed on the browser.

A telephone is required as a client device for voice-only applications or telephony applications. As for multimodal applications, the browser could vary from PC to cell phone with view enable to pocket PC.

3.2.2 Web Server

The web server is a sort of 'gateway' between the speech server and the application server. Data received from the client side is separated into voice data and text data via the multimodal server. The voice data is transmitted to speech services and interpreted. The result of the interpretation will be transmitted to the web server, and the HTML file is generated at the same time. In other words, when a user visits the website, the connection between the web server and the client is created to transmit requests from the user to the web server, return the response from the server to the client, and display the result in the browser. When the searching is executed, the connection between the web server and the data source is established. The Apache web server will be deployed to play this role.

3.2.3 Application Server

Architecturally, the application server resides on the middle tiers of the system, which executes the business logic. Namely, the application server is in charge of fulfilling the request, handling business transactions and establishing connections among the resources.

Tomcat is used as the middleware platform, which is a servlet container and the Reference Implementation of the Java Servlet and Java Server Pages technologies [37]. Tomcat version 4.1 is deployed in this case, and it implements the Servlet 2.3 and JavaServer Pages 1.2 specifications. It is open source software and released under the Apache Software License.

In production mode, to reach the high availability, scalability and performance, BEA's WebLogic is highly recommended. The bundled JRocket is very robust JVM and best for the server side application.

3.2.4 LDAP

LDAP is Lightweight Directory Access Protocol. It's an Internet protocol and designed for those applications that have more read and search operations than write operations. The OpenLDAPv2 is installed to simulate the Bell's LDAP server. In the Click&Dial system, the query function mainly runs on the LDAP server. In order to simulate the Bell LDAP, a new schema has to be designed, created and deployed on it. Some fake data are populated as well.

3.2.5 Database

The database is only used for the authentication purpose at present. Since the Bell's LDAP cannot be extended by us, several tables have to be created in the database to store the usernames/passwords and roles. CloudScape is a simple open source Java database. It meets the application's requirement and is easy to install and deploy.

3.3 Voice-related Components

3.3.1 Speech Server

The speech server enables client devices to access the speech applications, and it includes three components:

- Telephony server
- Multimodal server
- Speech Engine Services

Telephony Server

Most phones do not have the processing capability to run markup interpreters and speech processing software. To solve this problem, the telephony server, connecting the web application and the telephone client, is created to interpret HTML or speech markup language and script, and it also provides the physical connection and signal path to the

voice network. The network between the telephone and the telephony server could be PSTN (Public Switched Telephone Network), PBX (Private Telephone Network) or Voice over IP (VoIP) packet network [34].

Multimodal Server

In multimodal applications, there are roughly three types of multimodal input: sequential multimodal input, uncoordinated simultaneous multimodal and coordinated simultaneous multimodal input [27, 36]. The first one allows users to access the application by either speaking or typing, but never both. Thus this is the simple type. The latter two allows concurrent activation of more than one input. So interpreting the information from parallel inputs is the main problem in this application. To solve this problem, the multimodal server is deployed to perform the integration of the incoming signals from various inputs.

Speech Engine Services

Speech Engine Services consist of Automatic Speech Recognition (ASR) and Text-To-Speech (TTS) synthesis for recognizing speech input, playing audio prompt and responding to the user, respectively. The knowledge of speech recognition and text-to-speech synthesis was described in the previous chapter.

3.4 Summary

This architecture separates the client, the application and the database into different tiers to overcome the drawback derived from the tight coupling of all components.

The benefits of this architecture include:

- The application layer only transmits the data to the client in need without handling the extra information. The load of the network, therefore, is minimized.
- It makes the system easy to scale, since changes to the client side or to the application side are largely independent from one another. In other words, it is easy to add new functions to this system.
- The information and structure of the database could be hidden from the client, because there is a higher level of abstraction between the database and the client

side. In addition, modifying the database or removing it is quite easy to do without affecting other components.

- It is unnecessary to change the client side if the logic is changed.
- Other standard languages can be easily embedded into the application layer.

The main advantage of this architecture is that changes to front-end, middle or back-end tiers are largely independent from one another. As a result, if we modify the client side, it's unnecessary to change the application server or database a lot. Conversely, if we modify the database, change it, or even remove it, it affects other components or tiers a little. In our project, we expect to upgrade the web-based Click&Dial system to a multimodal system by keeping the existing function for graphic user interface and adding voice interface by using speech markup language. To reduce the time of development, we would like to rewrite the codes as little as possible. This architecture implemented by us makes it possible. We keep the application server for handling the logic part, and the database for data searching. The client side allows not only the normal computer but also other devices to access the application by both text and voice input and output. Embedding SALT markup language to HTML to handle not only visual pages, but also voice pages, modifies the web application.

The speech server is a new tier to achieve the new function for this system, which is responsible for speech recognition and speech synthesis. Therefore, the latter three components will be discussed in depth.

3.5 Speech Interface Framework

As we discussed above, speech engine service that handles the speech process is the key component for building a speech-enabled web application. In figure [3-2], the framework for speech interface recommended by W3C, along with the data flow and markup language, is represented in great detail. Mainly, there are two distinct parts: ASR and TTS that convert speech to text and break down the words of text to phonemes, and generate the speech audio for playing back separately. Based on the above figure, the

upper part represents the speech input flow, while the lower part demonstrates the process of speech output.

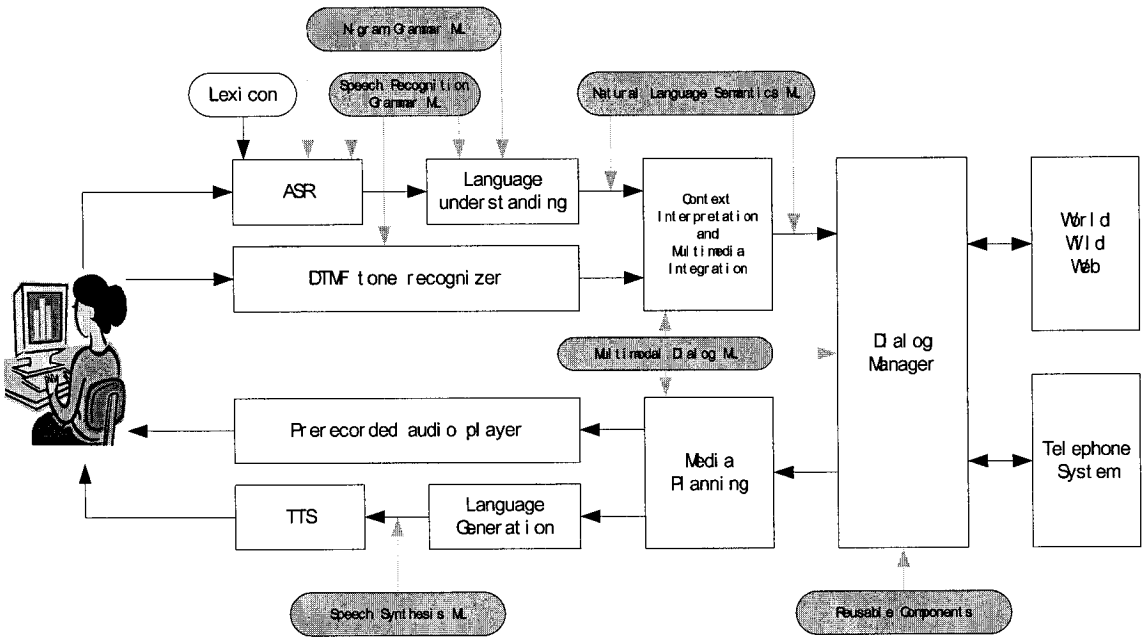

Figure 3-2 Speech Interface Framework (from W3C) [27]

In our project, we use Microsoft speech application SDK that includes the speech recognition and TTS engine, which are composed of several components for developing. As shown in figure [3-2], the first component for dealing with speech input is ASR, which involves converting speech to text. Speech grammar markup language that is a kind of N-gram grammar markup language defined by W3C is used to specify the words or phrase what is expected to be said by the user. The second component, language understanding, specifies the recognized text from ASR as semantics in order to determine its meaning. Namely, this engine compares the output from ASR with the special rule defined in grammar files iteratively. If the input is from the keyboard, the DTMF tone recognizer produces input strings directly when the user presses the key. A semantic markup language document is generated by the recognizer to represent the matched items. The semantics are enhanced to get the meaning of the input through context interpretation. Again, the improved SML containing the recognition confidence score and the recognition value is preceded.

Dialog flow is controlled by the dialog manager using markup language [27]. In our project, for instance, it determines the prompts to be played back, what the user should say and the dialog flow using SALT embedded Javascript. Several reusable components are defined in SASDK, which we could use across the multiple applications.

The first component in the speech output part is media planning, which determines which audio will be played back, since the prompts could be inline ones specifying the text that is anticipated to output or prerecorded audio files with the link referring to the director where the files are stored. If the audio to be returned is pointing to a prerecorded audio file, the prerecorded audio player is in charge of obtaining the files according to the link and playing them back. If the prompt is in the text, the text has to be converted to audio signals, which could be played in the client side. The text from media planning is transformed to the form presented using speech synthesis markup language. Finally, the audio signal is created through the text-to-speech synthesizer based on the special speech synthesis markup language that contains the information for sound generation.

Chapter 4 Design of the Web Component

Based on the proposed framework of the voice-enabled Click&Dial system, we separate the components into two parts for designing and building, and then integrate them together. In order to add the voice interface, the original system has to be modified to meet our requirements. In this chapter, we outline the main design considerations, objectives for redesigning the web components. The Unified Modeling Language (UML) is used to model and architect the system following the Model-View-Controller (MVC) approach. A set of the core design patterns are applied to the architecture as well.

4.1 Architectural Goals and Constraints

This section describes the objectives that have a significant impact on the software architecture.

1. This solution is an enhancement for the original Click&Dial system. Most of the functions, page layout and look and feel are kept intact to be consistent with the original one.
2. One important principle is to reuse the existing system as much as possible. Original hardware like servers will not be replaced and redeployed. Correspondingly, the database schema, LDAP schema will not be changed either.
3. The Java code will be redesigned and redeveloped based on the Object Oriented Design principles. The solution should be scalable from a small application to a bigger, more complex system.
4. The latest technologies, methodology and standards will be used to innovate the implementation of the system. A new MVC framework named Struts is introduced in this release. It changes the structure of the application fundamentally and also makes the whole system extensible, flexible, maintainable, etc. JAAS (Java Authentication and Authorization Service) [38] is adopted to replace the previous way to handle the security related operations such as login, authorization, etc.
5. The new J2EE standards will be followed: JSP 2.0, Servlet 2.4 and JSTL. They have several important features that are mandatory for a mature enterprise web application.

6. The solution must work in a non-standard hardware and software environment for corporate users (various combinations of Windows operating system versions and IE browser versions).

4.2 Design Motivation

According to the architectural goals and constraints indicated above, a number of factors were considered when redesigning the Click&Dial web application. The following design aspects are identified as the key design considerations:

4.2.1 Multi Tiers

One important design objective is to map the application modules and functionality to different tiers on the J2EE platform [40]. The benefit of a multi-tier system is that each tier can be built and executed on different platforms so that the application can be made scalable and extensible. For instance, if the business rules need to be changed, only the middle tier will be affected. At the same time, the presentation tier and enterprise information system tier are intact.

A typical web application is a three-tier architecture, as the following figure shows. The three tiers are the web server, application server and the back end enterprise information system for data management. Whether to use other tiers may depend on several factors such as data access and transaction, security, etc. An important question is whether to use an EJB in the middle tier or not. After analyzing the application's requirements, we choose not to use enterprise beans and therefore no EJB tier is required. The reason is that LDAP server is the main place to store the applications data and EJB could not bring much value in this case.

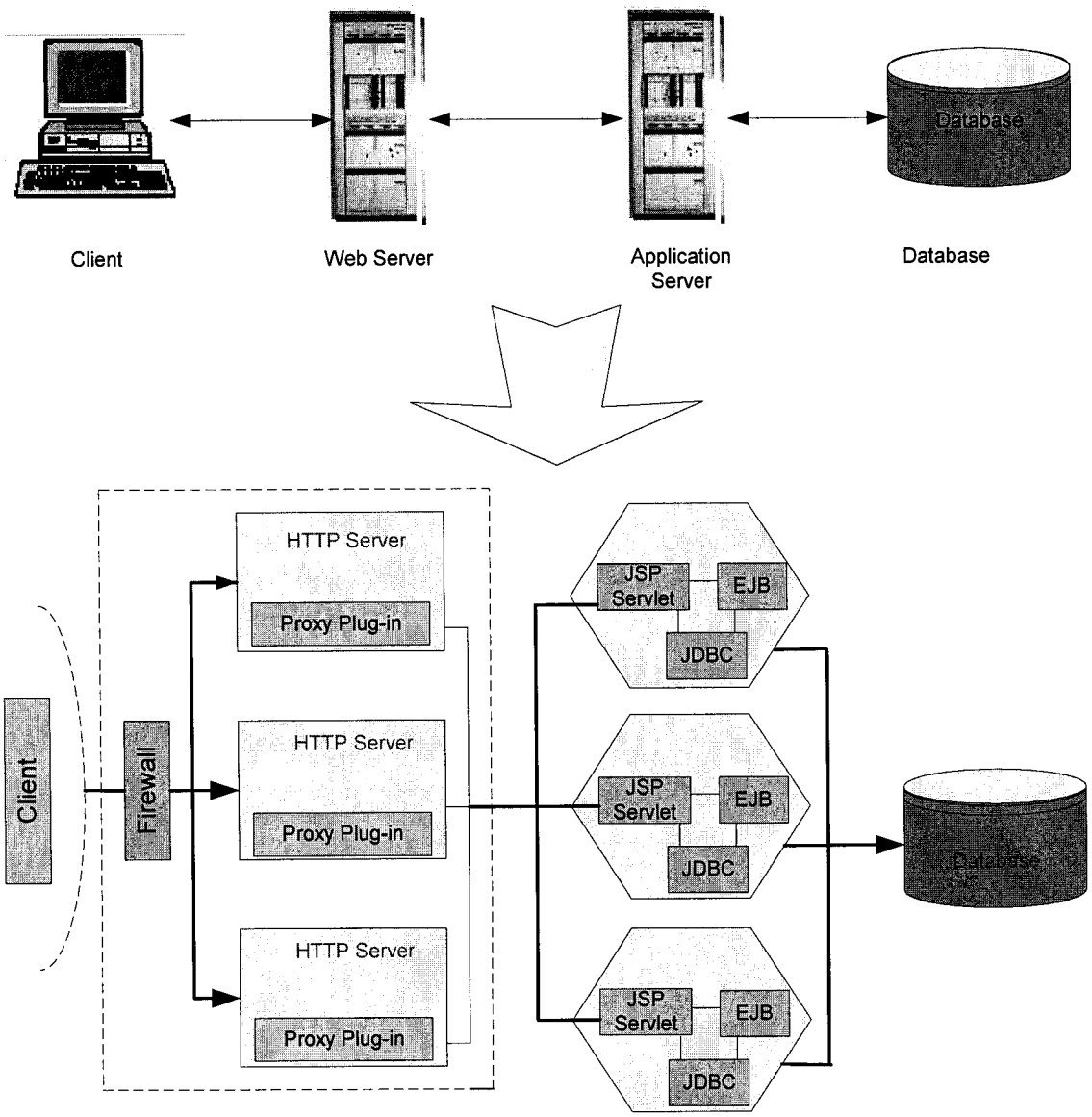

Figure 4-1: A typical multi-tier structure [75]

4.2.2 Data Access and Transaction

Bell's LDAP server is the main data source of the system. When the user searches the address book or browses the list, the LDAP access actions are predominantly read operations. Only when a user updates his preferred number or when the admin creates a new user account, the system performs update operation. In addition, LDAP is the only data source we use. Therefore, distributed transactions across multiple databases and LDAP are unnecessary for the Click&Dial system.

4.2.3 Framework

The overall framework of this system is presented in the previous chapter. In this section, we focus more on the framework of the web-based part. Using an existing mature framework such as Struts or JavaServer Faces becomes the main stream in developing J2EE web applications. It is a key design choice when architecting a web solution. With the help of the framework, most of the tasks are greatly simplified and standardized such as internationalization, error handling, validation, layout control, etc. Struts is adopted in our design because it is a popular Model-View-Controller and open source framework for building a modern web applications based on published standards and proven design patterns [39]. It not only shortens the development time, but also provides a consistent and unified way to develop the web application, which makes the develop process repeatable. This is very important in a teamwork situation.

Below is a nice diagram to illustrate the essential components of the Struts framework and how it processes the incoming request:

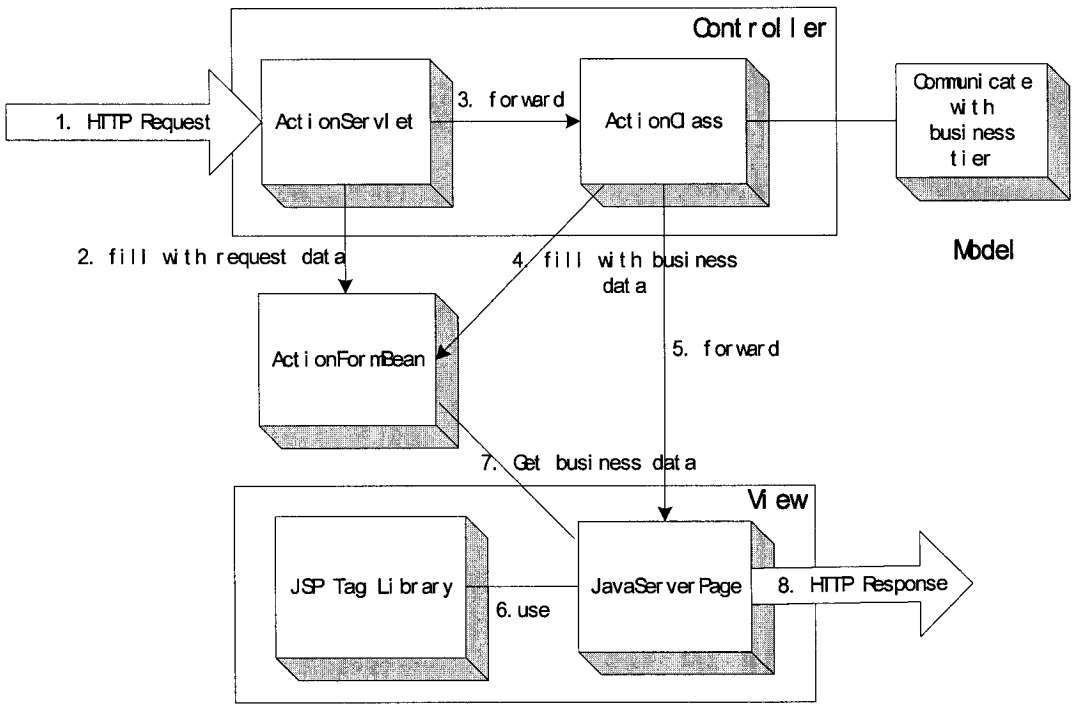

Figure 4-2 the essential Struts Components [39]

Once the request reaches the ActionServlet, which acts as the front controller, the ActionFormBean will be populated based on the parameters in the request. Then the

corresponding Action class will be called, and the backend business process will be executed in sequence. The result is filled to the ActionFormBean and forwarded to the right JSP file. In the JSP, the Struts tag will be translated, and the business data will be embedded in the response stream and sent back to the client.

4.2.4 Application State

When the client uses the application, lot client state information needs to be saved temporarily like the user profile, authentication flag, etc. This application state can be stored on any tiers [41]. Each tier has advantages and disadvantages for saving or maintaining the state. The key question is on which tier to store the session state for Click&Dial system. Due to the security, bandwidth problem and the HTTP protocol, which is connect-less, it is not the best way to handle the user status on the client side [40]. It is easy to maintain the state using stateful session beans on the EJB tier. EJB, however, is not adopted on our system. So the session state is determined to be maintained on the web tier using the servlet's HTTPSession, which is perfect for the Web-tier centric design like the Click&Dial system.

4.2.5 Security

The original security model is implemented by the developers themselves. It includes database lookup, session control and check login status on each page. To simplify those operations, JAAS is introduced and configured in the deployment environment. It provides a standard way to handle the authentication and authorization in a normal J2EE application. It automates the authentication and authorization procedures and meets the requirements of the Click&Dial system.

Normally the LDAP is supposed to be used for the user authentication because it is designed to have quicker performance for searching and reading data than updating. Since the Bell's LDAP server existed before the Click&Dial system was created, its schema could not be changed this time. Unfortunately, there is no user password attribute in the LDAP at this moment. As a walk-round, a database is created to save the user name and password. When an authentication request is sent, the database is queried instead of LDAP.

4.2.6 Communication

When a user submits a call request, it is sent from the web application to the back-end call processing application. Ideally, the communication protocol between the web tier and the call processing application requires a message service for decoupling these two systems. This will help build a high throughput, robust system. Because the Tomcat is a simple J2EE application server, there is no message service bundled with it. The trade-off is to use the Runtime.exec methods create a native process that can be used to run the command and obtain information from it. It is a not robust design but it is good enough to demonstrate the calling function.

4.3 Application Architectural Model and Design

4.3.1 Design and Model

The Click&Dial web application leverages the J2EE 1.4 platform features available for implementations that rely heavily on the web tier. Its architecture follows the Model-View-Controller (MVC) architectural approach. In the following sections, UML artifacts like classing diagram, sequencing diagram, etc. are used to describe the design models in detail.

Class Diagram

We use the class diagram to indicate the classes we have to create and depict the relationship between each of these classes. The diagram below expressed the main part of the class hierarchy:

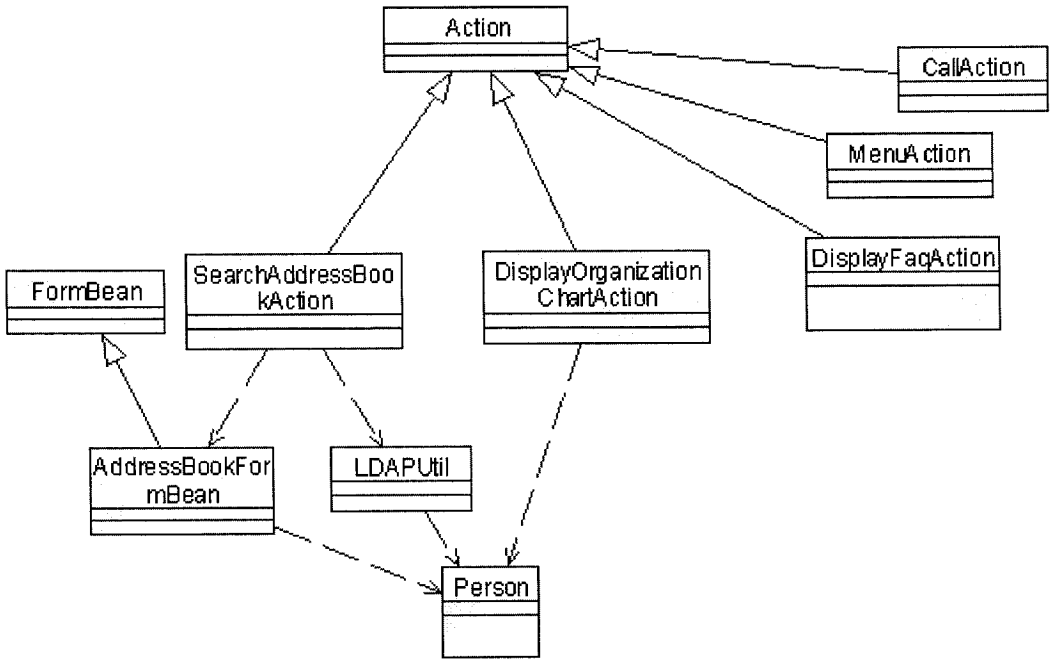

Figure 4-3 Class Diagram of the Click&Dial System

The Action class is the parent class provided by Struts. Other action classes are extended from it. It provides the web tier handlers like SearchAddressBookAction, DisplayFAQAction, CallAction, etc. Each of them provides their own implementation of the perform method. All of the parameters will be retrieved from the request object.

Person is a plain Java object. It stores the person entity's properties. Due to the simplicity of the application, it almost has all of the attributes used by persons in the Click&Dial like dn, name, bellpein, etc.

Use Case Realization

We use sequence diagrams, which are a collection of objects interacting to accomplish a given task or series of tasks over time [49], to realize the use cases.

This section shows the sequence diagram for each use case.

Login

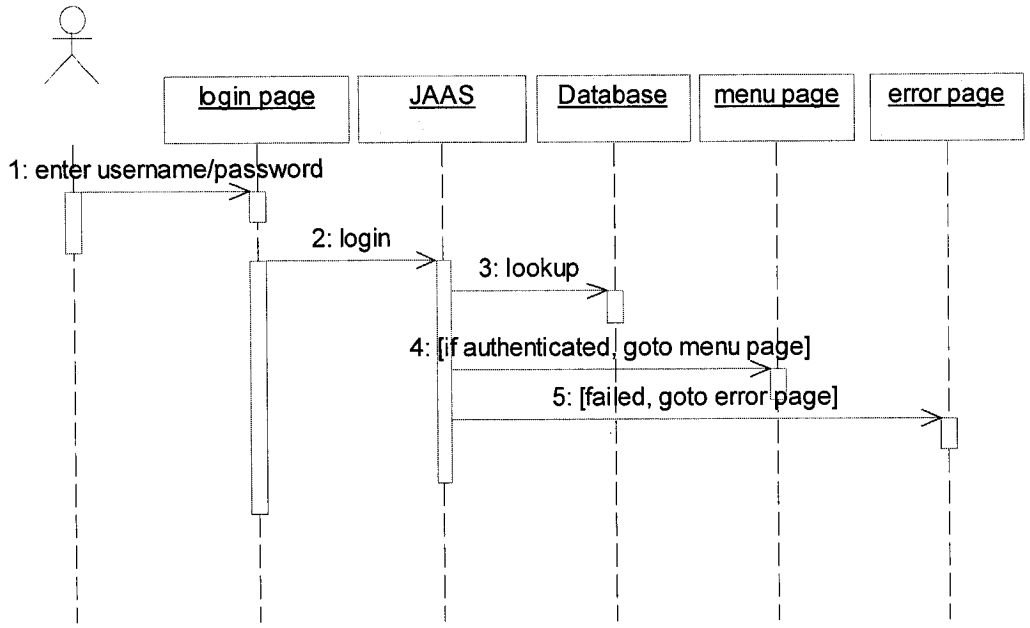

Figure 4-4 Login Sequence Diagram of the Click&Dial System

After the user inputs the username and password and submits it, the JAAS is triggered, and the pre-configed JDBC Realm will do a lookup in the database to match the username/password pair. If it is found, the menu page will be displayed. Otherwise, it will be redirected to the error page.

Manage address book

Once the user inputs the address info and submits the form, the AddressBookAction calls the AddressBookManager to modify the data. LDAPUtil will first call the ServiceLocator to get the JNDI connection and then update the LDAP.

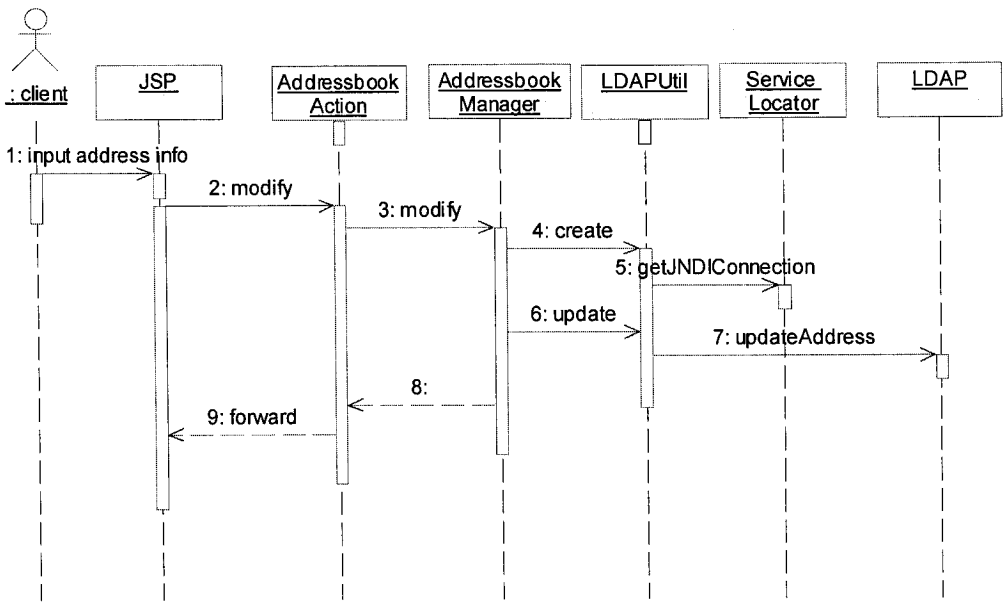

Figure 4-5 Manage Address Book Sequence Diagram of the Click&Dial System

Search

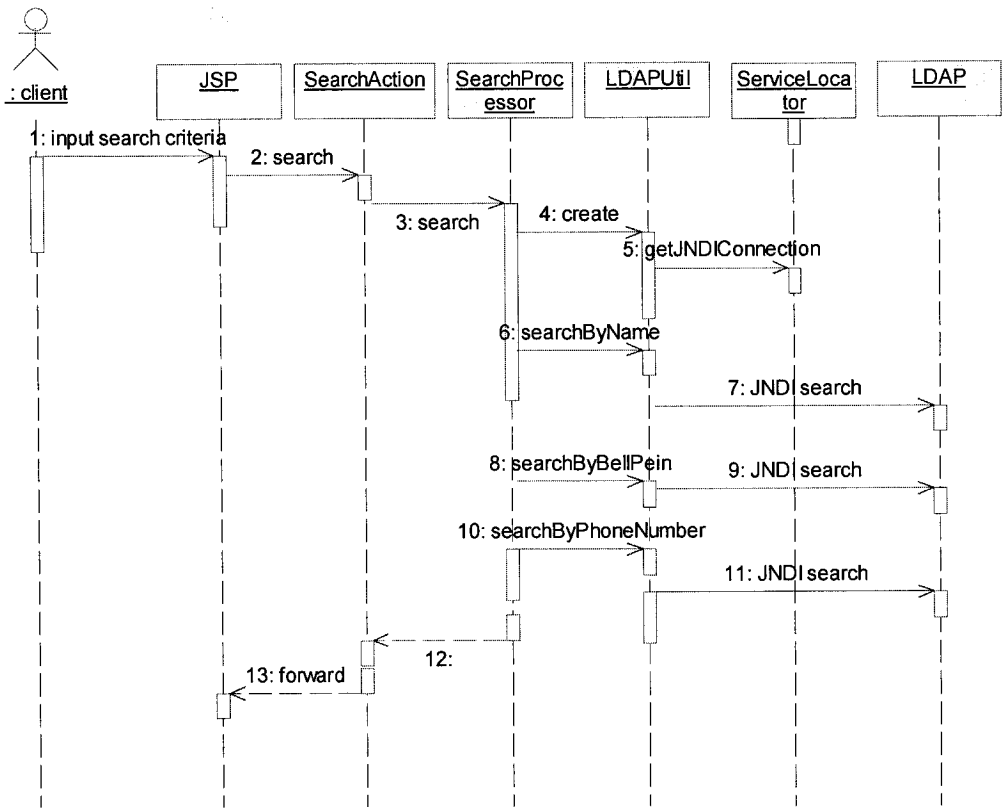

Figure 4-6 Search Sequence Diagram of the Click&Dial system

After the user submits the search form, the SearchAction calls the SearchProcessor to query the LDAP. LDAPUtil will first call the ServiceLocator to get the JNDI connection and then do the real query. If the search has results, it will be displayed in the results page.

Profile management

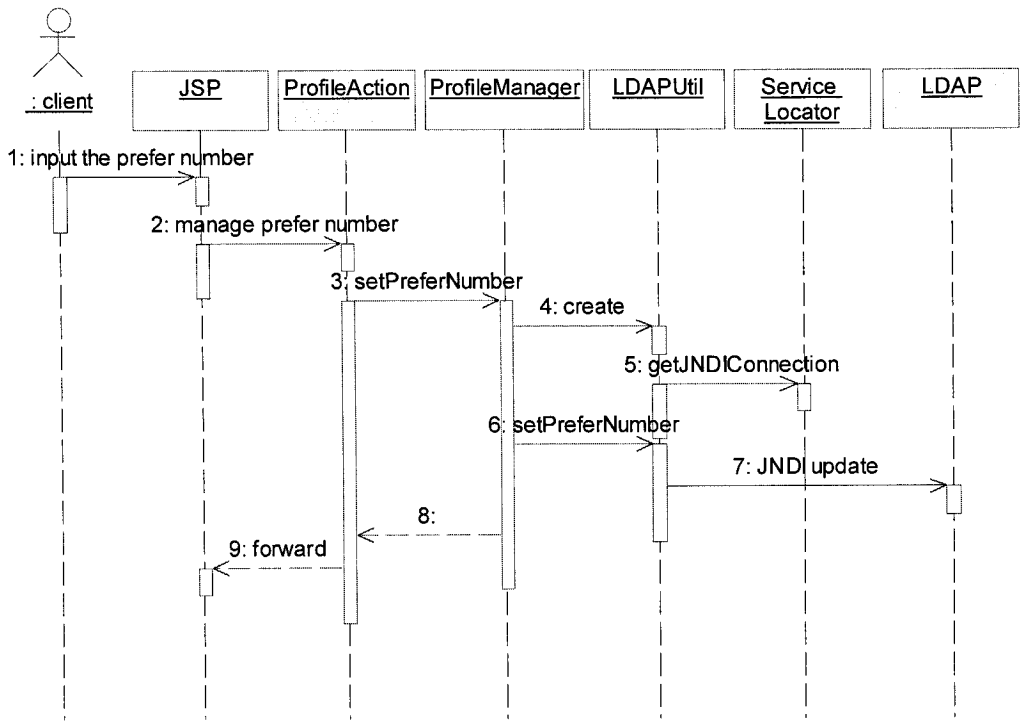

Figure 4-7 Profile Management Sequence Diagram of the Click&Dial System

The user inputs the preferred number and submits the form, and the ProfileAction class calls the ProfileManager to modify the LDAP data. LDAPUtil will first call the ServiceLocator to get the JNDI connection and then update the data.

Component Diagram

A component diagram is a simple, high-level diagram that shows the organization of and dependencies among a set of components [50]. We use this diagram to address the static implementation view of the system.

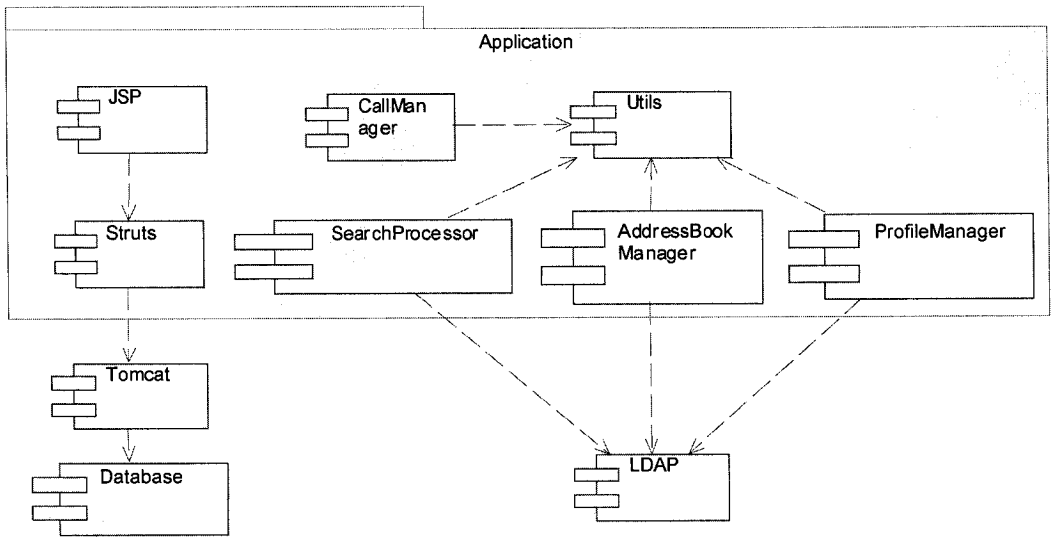

Figure 4-7 Component Diagram of the Click&Dial System

The above diagram shows the following components inside the system:

- In the web tier, there are SearchProcessor, AddressBookManager, ProfileManager, CallManager, Utils, Struts, JSP pages components.
- Tomcat is the application server itself and plays a key role in the architecture.
- Database and LDAP components are in the EIS tier. They store the data and provide the lookup service to the upper tier.

Deployment Diagram

A UML deployment diagram is used to show the hardware on which the system is running, the software installed, and the middleware we used to connect each machine [50].

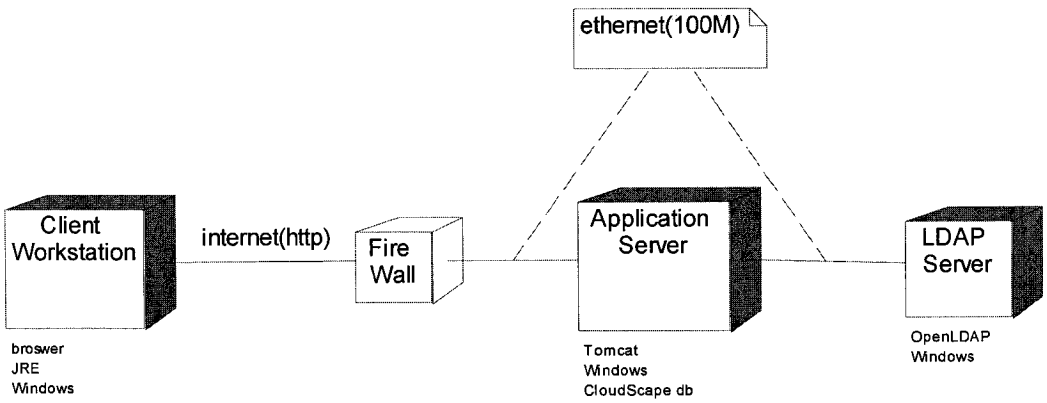

Figure 4-8 Deployment Diagram of the Click&Dial System

This diagram represents the deployment view of the Click&Dial system only for the web-related tiers:

In the client side, it is a thin client using web browser. It could be any OS platforms that supports Microsoft IE, Netscape, or Mozilla, etc.

In the server side after the firewall, since it is not a production deployment, the web server, application server, database and LDAP server are deployed in one single machine. A small cloudscape database is also installed in this box. The LDAP service is provided by OpenLDAP. It has the same schema as Bell's LDAP server and simulates the same behavior of the real one.

4.3.2 Design Pattern Based Architecture

Generally speaking, design patterns are proven design solutions for recurring design problems [42]. The Click&Dial application is architected to adopt several common design patterns cataloged in the Java BluePrints pattern section that provide a solid foundation for the application. The patterns work together and follow an overall MVC architecture. The following is a brief description of the patterns we used.

Model-View-Control Pattern

MVC organizes an interactive application into three separate modules [43, 44]:

- The application model has its data representation and business logic

- The views are responsible for the interaction with the users. It provides data presentation and gets the user input
- The controller is to dispatch requests and control the program flow

Benefit of MVC [44, 45, 46]:

- MVC decouples view, controller and system model.
- MVC minimizes the code duplication, centralizes the control.
- Because of the decoupling, the same components can be called by any interface. In other words, the user interface can be the multiple views of the same model.
- High scalability, and ease of modification and maintenance

By using MVC pattern, it is easy to migrate our system and it becomes possible to maintain a system, which comprises different technologies. Since the client view is separated from the controller and data model, once we embed the voice interface into graphic interface, we do not need to change the controller and model. As we discussed in previous chapter, we will use openLDAP to simulate Bell's LDAP during the development period. Eventually, the Click&Dial system will be deployed on the product server connecting with Bell's LDAP. Migrating from one data source to another data source or adding a new data source becomes very easy when we adopt the MVC pattern. We only need to write few code that adapts the new data source, Bell's LDAP, to the Click&Dial System. The interface and the controller portion will not be affected.

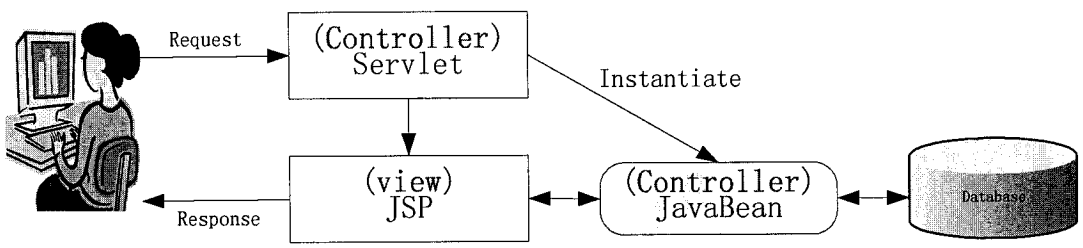

Figure 4-9 MVC Design Pattern for J2EE [47]

The figure above shows a typical mapping between Java object and the MVC. The model is Java Beans, the view is JSP web pages and the controller is the servlet. Most Web-tier application frameworks use some variation of the MVC design pattern.

J2EE Design Patterns

There is a set of core design patterns for the J2EE platform. This section briefly mentions those J2EE design patterns that apply to the Click&Dial application. These patterns are combined and used in the application architecture.

Pattern Name	Use Scenario
Composite View	Tile, which uses this pattern, is adopted in our system to divide the page to three parts, including header, footer, and body.
Façade	There is a SearchMgr class playing the role as the façade in our application.
Front Controller	Struts' ActionServlet works in the same way described here.
Service Locator	It's used to retrieve the LDAP server, URL, etc., information
View Dispatcher	Struts framework handles this function.
View Helper	Struts tag library supports this pattern.
Value Object	In the code, Person JavaBean is used as the Value Object to transfer data between the tiers.

Table 4-1 List of Design Pattern applying to the Click&Dial system

The detailed description for each pattern is summarized and listed in Appendix J2EE Design Pattern.

Chapter 5 Design of SALT-based Voice Interface

According to the framework we proposed, it is notable that our system is combined with two main sub-systems: web-related and voice-related systems. The design of the web-based components is greatly described in the previous chapter. This chapter mainly emphasizes the design of the voice-related system using Speech Application Language Tag (SALT) to build a more flexible and user-friendly voice interface.

5.1 Introduction of SALT

As a result of the comparison among VoiceXML, XHTML plus Voice and SALT we discussed in the second chapter, SALT is our choice used as the markup language for building the voice interface.

The advantages of using SALT are [34, 60]:

1. Easy integration of speech with web pages

SALT, which is designed to leverage the web application, can be seamlessly embedded into the Web browser side with HTML, XHTML, XML or other standards to implement the speech interfaces. It allows reusing the knowledge of web developers. Therefore, designing speech applications in SALT becomes simple.

2. Reuse of existing application logic

Separating speech interface from business logic tiers and database tiers is one of the fundamental design principles in SALT. In other words, the existing application logic layer or database can be retained, so that the time of development is shortened tremendously.

3. Reused in different pages or applications

The dialog framework, which drives the SALT speech interface, could be combined with underlying data structure loosely or tightly. The dialog components, therefore, can be reused in different pages or across applications.

4. Reuse existing standards

SALT developed by industry consortia (SALT Forum) was contributed to W3C as a part of their ongoing work on speech standards. It is a lightweight application-level markup language because it reuses existing standards for speech output and grammar formats.

5. Power and flexibility of the programming model

Power and flexibility are the important criteria for high-quality speech application. The programming model of SALT makes it extremely powerful and flexible by offering the graceful control of speech dialog.

6. Use in various devices

The speech application written in SALT is able to deploy on a broad range of devices that enable various services. The devices could be ranged from normal PCs and ordinary telephones to small devices with wireless access, such as pocket PCs and cell phones. It definitely satisfies different requirements from different clients.

7. Reduction in cost

Since the SALT speech application separates speech components from logic and data tiers, both the logic and speech components can be reused across applications. Therefore, the reuse of our work is maximized, and development time and application maintenance time are minimized. In other words, it is able to realize the reductions in cost of development.

5.2 *Infrastructure of SALT-based Click&Dial System*

The overall framework of the whole system for the general scenario is presented in the previous chapter, which gives the common idea on how to build the voice-enabled web application. This section presents a more detailed description based on three different scenarios [34, 35].

There are several use scenarios for the SALT. Here is the introduction for each of them:

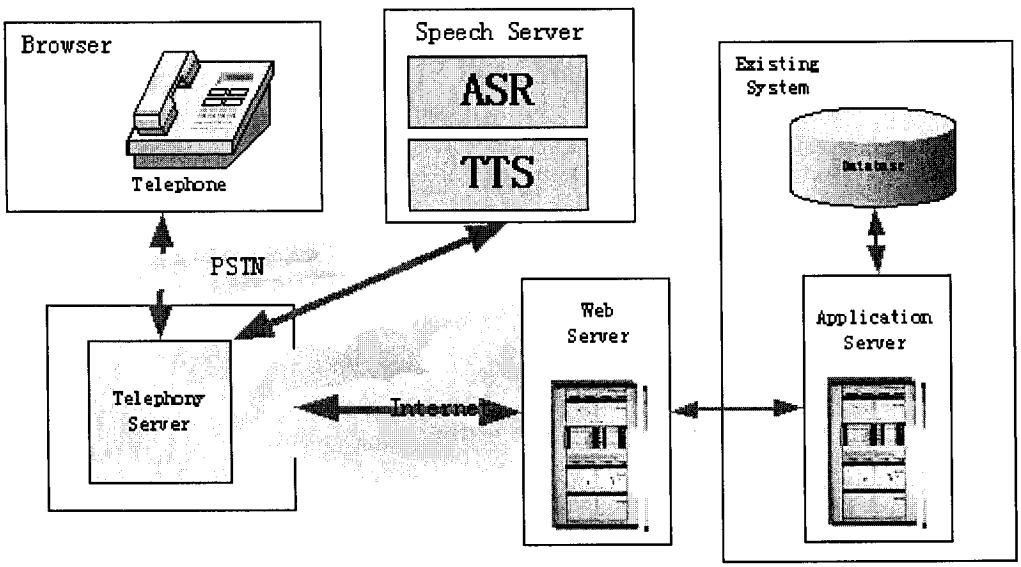

Figure 5-1 Framework of Click&Dial System for Telephone Scenario

5.2.1 Telephone Scenario

In this scenario, a telephony server is built to interact with the web server. In some cases, it will be classified as the client side. The client device, telephone connects to this server through Public Switched Telephone Network (PSTN).

Once the user dials a number to access to the application, the connection between the telephone and the telephony server is established. According to the results from the SALT interpreter, a corresponding voice page is returned to the client side. At the same time, another connection between the telephony server and the speech server is established, where speech is sent to the speech server from telephony for audio processing, and the recognized result is returned.

5.2.2 Desktop Scenario

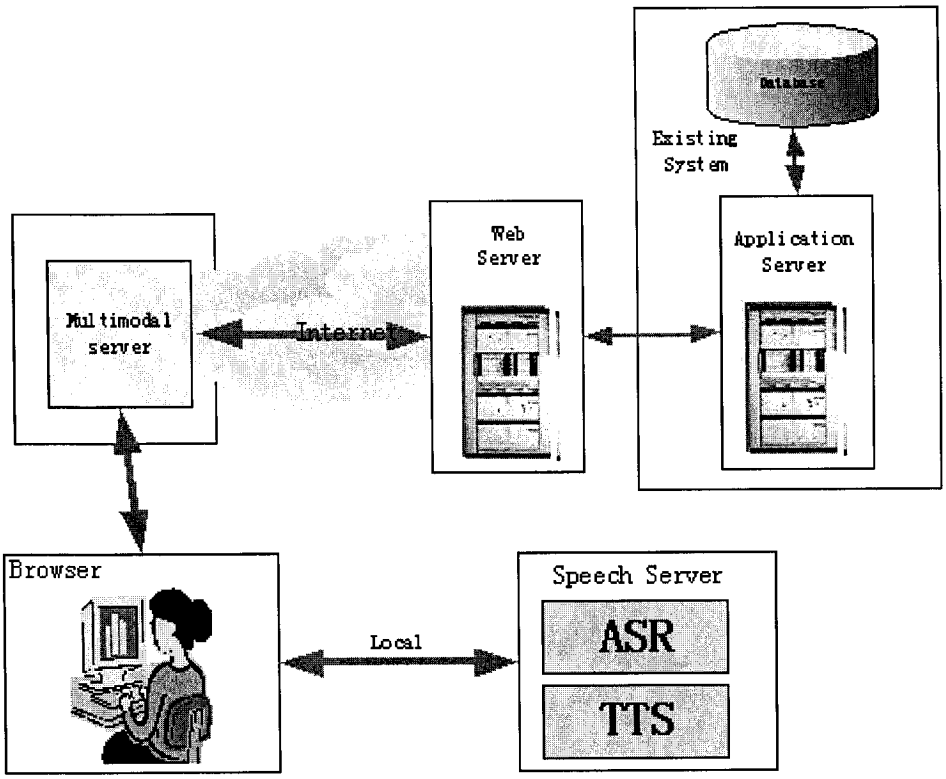

Figure 5-2 Framework of Click&Dial System for Desktop Scenario

In this case, the client side is the desktop computer installing Microsoft Internet Explorer (IE) with the speech add-in for Microsoft Internet Explorer combined. The speech engine services, including the recognition engine and the TTS engine, are installed on the client side. Namely audio processing, speech synthesis and speech recognition are performed locally.

Once the user accesses the application by entering the URL in the Internet Explorer, the corresponding visual page and voice page are turned back to the client. HTML handles the Graphical User Interface (GUI), SALT handles the voice interface, and JavaScript deals with the speech control, separately. Once the user accesses the application, the system will automatically prompt the user to tell him what he can do. Since this system is multimodal, the user can operate the application either by typing using the keyboard or by speaking. Both the text input and the spoken utterance will be sent to the multimodal server, and then the spoken utterance is recognized through the speech recognition engine

locally. If necessary, the prompt engine and TTS synthesis play back the prompts. If the interaction between the client and the server is necessary, such as searching from the database, the input text or recognized value will be sent to the back-end side, the results will be turned back to the client side and both will be displayed in visual and prompted in voice.

5.2.3 Pocket PC Scenario

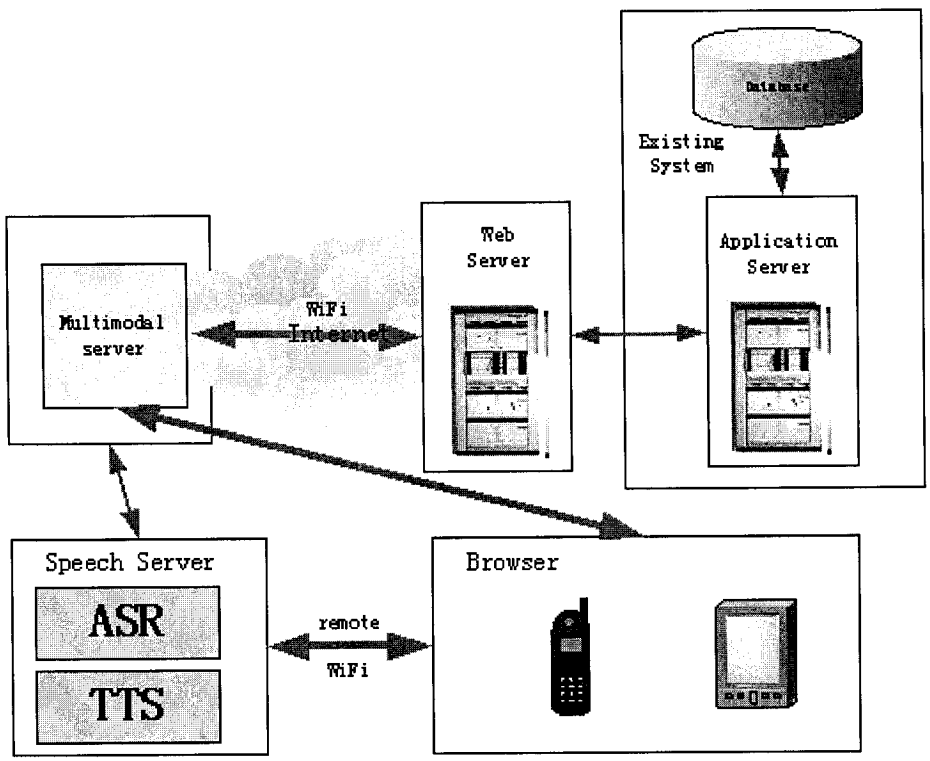

Figure 5-3 Framework of Click&Dial System for Pocket PC Scenario

Unlike the desktop, the pocket PC needs to install the Pocket Internet Explorer with a speech add-in for Pocket Internet Explorer. Compared to the rich desk computer client, the pocket PC is a kind of thin client due to the limitation of its capability. Thus the speech engine services, including the recognition engine and the TTS engine, cannot be installed on the client side so far. Instead, we propose building a speech server separately to handle all these processes remotely.

For example, once the user accesses the application by entering the URL in the Pocket Internet Explorer, the connection between the client and the web server is established. A corresponding page including HTML, SALT and JavaScript is sent, along with the pointer for the speech server. Meanwhile, the connection between the user and the speech server is also established. The prompt file, along with the page, is sent to the speech server and played back through the TTS engine. Therefore, the graphic user interface, along with the voice interface, is launched on the client side. If the user enters the information by typing, the text input is sent to the web server directly. If the user browses the Web by speaking to the computer, the spoken utterance is recorded first, and then the recorded audio, along with the grammar that is sent back from the web server, is sent to the speech server for speech recognition. The Semantic Markup Language (SML) document generated by the speech engine is returned to the client side. The result value needs further processing on the application side if the page contains dynamic content.

As we discussed above, three frameworks are proposed based on the different scenarios. The business logic part, backend of these three scenarios is same, which is Java-based.

In the first two scenarios, J2EE is adopted to implement the business logic part. But it can not fit in well with the pocket PC scenario, i.e., wireless devices, due to the following problems:

- J2EE provides a rich set of classes, which support network communication. It can not run on the pocket PC, which is a sort of thin client with a few hundred KB of memory.
- Different wireless devices have different networking capabilities and different file I/O requirements [76].

J2ME (Java 2, Micro Edition) is a set of technologies and specifications, which support a variety of wireless devices [77]. Therefore, J2ME is the solution for implementing the proposed framework of the pocket PC scenario.

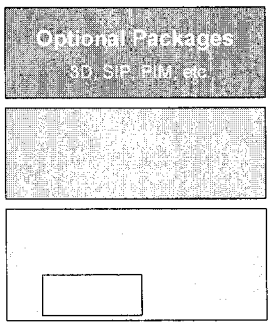

Figure 5-4 J2ME Overview [79]

J2ME consists of three elements: Configuration, profile and optional packages shown in figure [5-4].

Configuration: provides a set of libraries and a smaller virtual machine. The connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC) are defined.

Profiles: builds on the top of the configuration to provide a runtime environment for building applications [78].

Optional packages: is a set of APIs providing functionality, which may not belong in a specific configuration or profile.

In our project, we only give the design and implementation for the desktop scenario to prove the feasibility of our proposed framework.

5.3 SALT-based Model and Design

In the voice-enabled application system, speech interface design has a great impact on overall system quality. There are several interface design issues, which should be considered during the system design, implementation and evaluation phases. For instance, all the predicted words and phrases, which the user might speak, along with the meaning of these words, will be considered when we design the grammar. Meanwhile, the validation and tuning of these utterances are required. According to the design principles of a voice user interface, several designs are presented as follows:

- Navigation design
- Recognition mode of SALT

- Prompt of SALT
- Initiative model
- Grammar design

Based on our goal to provide both the visual interface and the voice interface clearly and leverage the previous logic tier and the back-end tier, key features of voice interface design are presented:

- Login authentication
- Searching phone numbers quickly, easily and precisely by user name, user pein number or phone number directly
- Easy-to-edit personal profile, such as browsing the address book on setting or changing the preferred number
- Easy to get help once the user gets lost
- High quality services, including high rate of speech recognition and speech synthesis naturally

5.3.1 Design Requirements

The system is designed to satisfy all users, varying from newcomers who use this system for the first time or seldom use it to advanced users who are familiar with this system. The explicit flow is a main point for designing voice-enabled applications. A well-explained system with powerful capabilities will attract all users. Thus the application should be flexible enough to provide an explicit and easy-to-understood interface for novice users, and enable power users to get results as quickly as possible. More natural, intellective interaction is designed in our system. Speed of prompt is another key factor of voice-enabled applications. A reasonable pace is chosen by us to avoid speaking too slowly to make users lose patience, and speaking too fast to make users lose their way.

5.3.2 Navigation Design

Designing the voice-enabled web application is quite different from the traditional web applications [35] that provide only graphical user interfaces. However, the voice-enabled

system allows users to 'view' the pages both by seeing and listening. And this system is designed mostly for users whose hands are not available or for customers who use pocket PCs or cell phones with a small view. Thus, one design requirement is a well-explained and well-organized navigation with main menus.

From the view of the user, easy to operate, friendly user interfaces, and powerful functions are the key point; while from the view of development, it is important to be able to maintain the system and scale the existing system easily.

So the very first step for designing the navigation of the application is to make a sketch of the whole system. The high-level flow of our system is presented below in figure[5-4]:

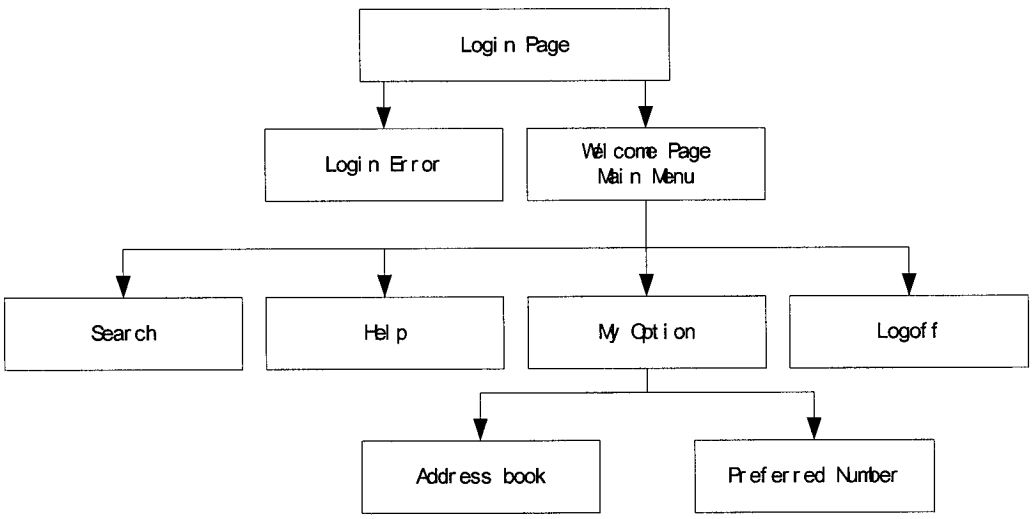

Figure 5-5 Navigation Flow of the Click&Dial System

The above flow diagram illustrates all of the functions with the Click&Dial system at a high level.

Login Page

As we expected, the first page is the login page, which asks users to type or speak their username and pein number to be sent back to the server as usual. Authentication will be accomplished in this part. As a result of authentication, two possible pages will be displayed on the browser corresponding to the result of authentication.

Login Error

The login error page will be launched automatically once the authentication has failed.

Welcome Page (Main Menu)
As a result of a successful login, the welcome page is started to welcome current users and provide users with several operations. Users can either search the telephone number of the person they are looking for or edit their personal profile by choosing 'my options'. Furthermore, 'help' and 'logoff' are also available for users to get timely help or exit the application.

Search
On search flow, users can search the telephone number by user name, user Bell PEIN or by entering the phone number directly.

Help
This flow will help novice users with little experience to use this system or give a hand to users who are lost in other flow.

My Option
Once the user chooses this menu from the welcome page, there are two options the user may choose: address book and preferred number.

Address Book
From this page, the user can get the last five phone numbers he called. Furthermore, the user is also able to store the Bell PEIN number of the person he calls quite often.

Preferred Number
This functionality enables users to set their preferred number while it's empty and change the preferred number, where the preferred number will be shown to other users when they search your name or Bell PEIN number.

Logoff
The last component shown in the above flow diagram is Logoff. Once the user has done the search or other operator, the user can leave this application safely via Logoff flow.
Through the above flow diagram, the available functions and components are outlined to users. A great detailed explanation of each component will be presented later.

The predefined flow of the application gives users the main idea about this system. It is perfect for the user who is familiar with this system. But the true case is not as simple as expected. Let's think of this scenario, if the user does not hear the prompt clearly, and the view of the cell phone is too small to see the results clearly, repeating the prompt may be required at this time. And also, the user anytime and anywhere could require 'help'. All

these scenarios make our system more complex. So to achieve the requirements for making the system more flexible and intellective, we created several global commands that are available throughout the application.

Main menu: let users return to the welcome page (main menu)

Repeat: repeats the previous prompt

Help: enables users to get help anywhere by going to the help page

Logoff: lets users leave the application safely

As the term suggests, global commands can be used throughout the application. Therefore, our system allows users to turn back to the main menu, repeat the prompt, get help and logoff anytime and anywhere.

5.3.3 Recognition Mode of SALT

Controlling the recognition process is quite important for speech recognition. When and how to start and stop the speech recognition process and output the recognition results are varied in different situations.

For example, in telephony or voice-only application cases, while the connection is set up between the client side and the speech server, the recognition platform has to decide when the process is started and when to stop it, and then output the result of recognition. In click-and-talk scenarios, it is pretty simple to control the period of speech. Speech is detected as long as the user taps the button or key, and tapping it again will cease the listen. However, the multi-modal application, which is designed not for special devices, but all communication devices, doubles the complexity of control, since it is the combination of the above two scenarios.

In addition, the results of the recognition need to be proceeded during the period of speech, rather than after the speech is stopped. Dictation, the typical mode of this case, needs to return the recognition results from when users start talking until they stop speaking.

The basic controller methods are start() and stop(), which are the means to start the recognition process and stop the process, respectively. The key point is when to invoke these two methods. SALT defines three main recognition modes to handle the process of

speech recognition according to various cases. These modes are listed as follows: single mode, automatic mode and multiple mode [59, 60].

Single Mode

In single mode, the speech recognition process is triggered by the start() method and ended by invoking the explicit stop(). Typically, this mode is largely used for tap-to-talk cases. Speech detection is started while the user taps the key until the stop() method is called by clicking the key again. There are two possible results: failure or success; the event of onnoreco or onreco is triggered, respectively. In single mode, out of babble time is the sole way to end the process of recognition automatically [59, 60].

Automatic Mode

In contrast to single mode, the process of recognition in automatic mode ends automatically according to different situations, rather than by calling the explicit stop() method. As with most cases, the process of listening is begun as soon as the start() method is running. Either babble timeout or the event of onnoreco will directly result in the end of detection. Equivalently, the implicit stop method is called if the recognizer fails to recognize the input from the user [59, 60].

Multiple Mode

Unlike automatic mode, which is typically used in hands-free cases, multiple mode is more useful for complex scenarios. Just like its name, this mode has multiple functions. Due to flexibility of this mode, we choose this mode for the recognition part of the speech server. In addition, there can be a pause between two phrases, which is more natural. That is another one of our reasons why we choose this mode to recognize the utterance. In this mode, furthermore, the recognition result could be returned at intervals between two phrases rather than be returned after the listening is complete.

First of all, the most complicated case is presented in Figure [5-5]. Once a phrase is input, the speech recognition is started by handling the event of *onspeechdetected.* There are three phrases to be recognized in the case of Figure [5-5]; therefore, the *onspeechdetected* event is thrown three times. As the recognition of the first phrase is finished, the results are returned before the second utterance. The same process happens to the second and third phrase according to the figure. In this mode, the end of listening is under the control of *maxtimeout*, *stop* method and so on, which is relatively more complicated than the other mode. In fact, it is a combination of the above two modes, the single mode and the

automatic mode, since the recognition could be either directly stopped by calling the stop() method or automatically ceased in response to the maxtimeout excess.

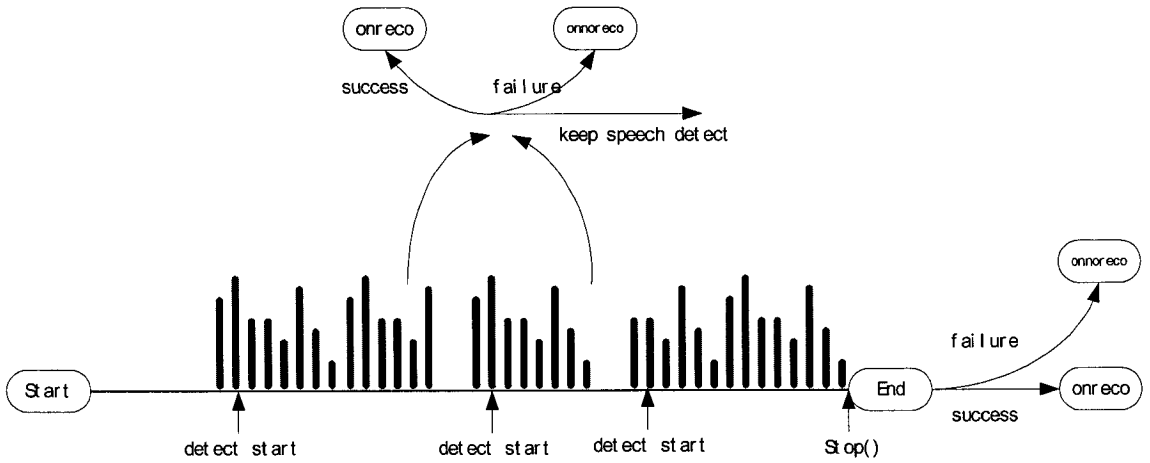

Figure 5-6 Multiple Mode [59, 60]

Based on the description of Figure [5-7], it is noticeable that exceeding the babbletimeout will directly terminate the speech recognition, no matter in which mode. The process of this case was shown in detail in both of the above modes.

Figure 5-7 Multiple Mode for Babbletime Scenario [59, 60]

The unrecognizable case shown in Figure [5-8] seems to be the same as in the single mode, but there are actually differences between these two modes. As we expect, the system will keep recognizing after the *onnoreco* event is thrown out. Stopping the recognition makes the distinction between multiple mode and single mode. In single mode the detection won't stop until the explicit stop() is run. However, there is more than one method to complete it in the later mode.

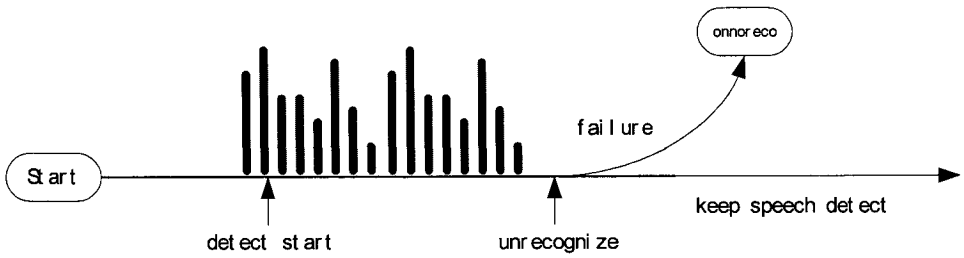

Figure 5-8 Multiple Mode for Unrecognize Scenario [59, 60]

Again, we could get the truth that the initial timeout won't stop the process in this mode, except to invoke either the explicit stop method or the implicit stop derived from babbletimeout or maxtimeout.

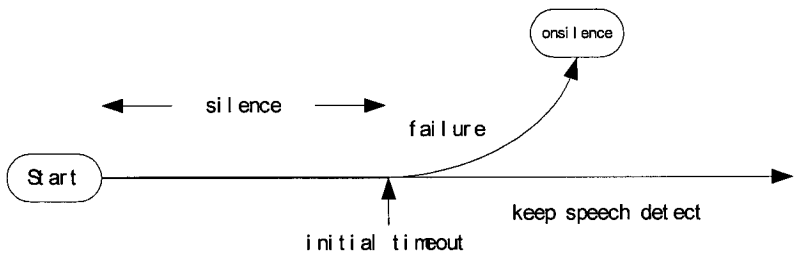

Figure 5-9 Multiple Mode for Initial Timeout Scenario [59, 60]

Once we know the features of the recognition mode, it roughly gives us some idea about when and which listen event might be thrown along the timeline. Therefore, we can handle them gracefully. There are four major listen events that might be fired: onreco, onnoreco, onsilence and onerror [5].

Onreco, is fired when the utterance was successfully recognized by speech recognition engine.

Onsilence, is thrown out when no speech is detected if it is out of the initial time.

Onnoreco, is fired when the utterance was unable to be recognized by speech recognition engine.

Onerror, is thrown when fatal error occurs.

It is important to process the results once the speech is recognized successfully. However, error handling is also the key point to perfect the application. The method to handle varies according to the different events. For example:

```
<salt:listen mode="multiple" id="userName"
        onerror="Report('onerror')" onnoreco="Repeat('onnoreco')"
        onsilence="Repeat('onsilence')" onreco="RunDialog()" >
        ...
</salt:listen>
```

Listing 5-1 Listen Events for Multiple Mode

In our project, once the utterance is recognized, the next prompt or listen will be executed as we anticipate. If the event of onnoreco or onsilence is fired, the method repeat() will be called to play the prompt to ask the user to enter the data again. If a serious error occurs, the application will be shut down, and the error message will be reported to the user. An example of event handling on the search part is given later.

5.3.4 Prompt of SALT

In SALT-based applications, audio input is specified by the listening element, which is introduced above. On the contrary, the prompt element is used for speech output.

Various content of prompts makes manipulation of audio output more complicated. Although prompts can be declared and played back individually, it is better to handle prompts by defining specific models. SALT does provide such kind of models to manage the processing of speech output, which will be introduced one after another, as follows.

Basically, a model of prompt queuing consists of the prompt, the subqueue and the queue, where prompt is the smallest element. In fact, rather than the prompt, prompt subqueue objects affect the management of speech output. Subqueue objects could include one or more prompts in sequence while prompt queue objects may contain several subqueues, similarly [59, 60].

The prompt queue greatly follows the principle of the queue: first come, first out as showed in Figure [5-10]. "A" displays the case that there is no prompt to be handled when the queue is empty. Once the prompt1.queue() is called, this prompt will be added

to the subqueue and put at the front of this subqueue. If the prompt.start() method is not invoked, as many prompts as the user wants could be sequentially added to the same subqueue. Diagrams B, C and D list the cases where the subqueue holds one, two or three prompts, respectively. The second and third prompts are added into the rear of the same subqueue one by one. It is illustrated that playback won't be started until the start() method is called in case D. Prompt.start() is the method not only to start the playback immediately, but also to close the current subqueue. It is just like placing a boundary at the end of the subqueue [5]. As a result, no more prompts can be added into this subqueue. Therefore, a new subqueue is built if there are more prompts to be input.

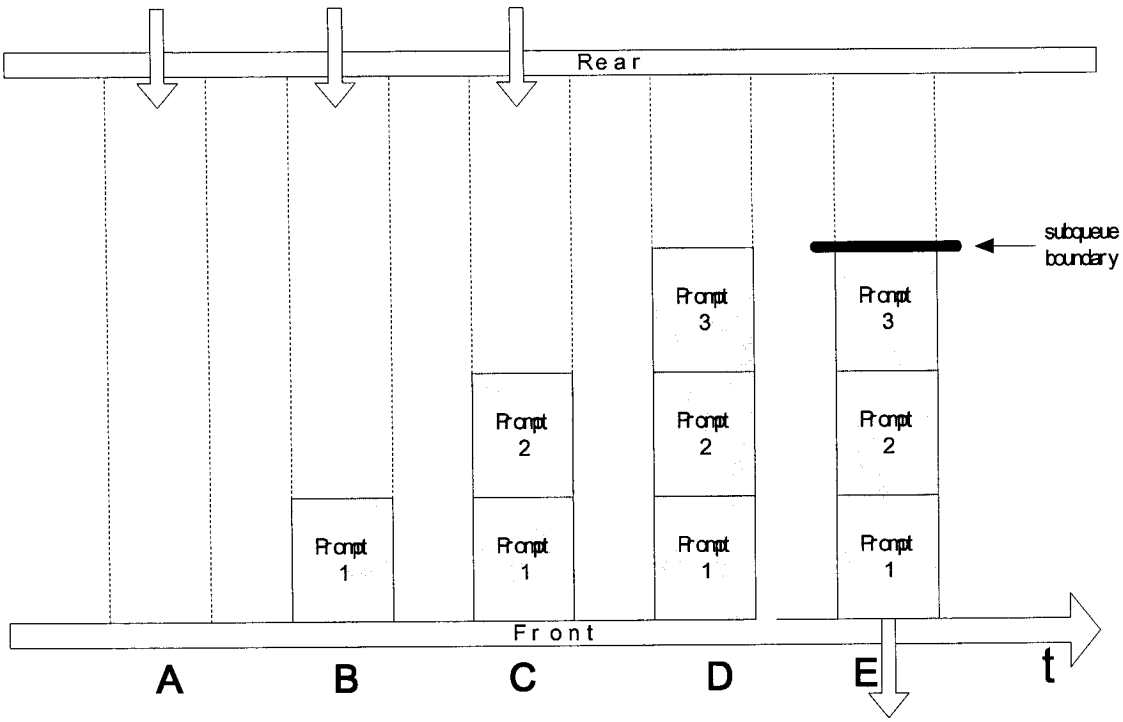

Figure 5-10 Prompt Queue Timeline [59, 60]

As soon as the start method is available to the queue, playback begins with the first prompt. The event of oncomplete for the first prompt is thrown out while it is finished. The next every prompt will start successively, and the end of the last prompt of this current subqueue will result in the onempty event for this subqueue. Finally, it returns to the initial status to wait for another prompt.

Since the current subqueue is closed, if there is a new prompt queued during the playback of the current subqueue, it builds and initials a new subqueue instead of adding into the current one. The new prompt won't affect the ongoing playback. After the current subqueue has been finished, the next subqueue will not begin to play, but will wait for the start call.

The event of onerror is thrown to stop queue or play somehow; for instance, it cannot be queued or played out. There are two different methods: stop and flush, defined to stop the prompt subqueue [59, 60]. Once the stop method is invoked, the prompt, which is in playback, is stopped and flushed, and other prompts in the same subqueue are flushed away as well. The stop method, however, does not affect the next subqueue of prompt that has not been scheduled yet.

In contrast to this method, the result of the flush method is that all the prompts in all subqueues are cleared away, no matter whether the prompt is in playback or not and no matter whether the subqueue is scheduled or not. Hence, the effect of this method makes the queue empty.

5.3.5 Dialog Initiative Model

In order to make the application more flexible to meet the increasing demand of users, the web application should not only offer a one-step process, but also multi-step interactions or complex applications ideally. Thus, a lot of dialog control logic is required for simple dialog structures. As the voice capability is added into the application, the web application becomes more complex, especially in multi-modal speech applications. Regarding the impact on the user's experience, it is important to use dialog models for both hypertext-based flow and speech dialog flow. Since the speech activity is the new application for the Click&Dial System, we will discuss the speech dialog flow below.

There are many different speech dialog models according to the various ways to classify them. From the view of the initiative model, three main dialog models are defined. Namely, these three dialog models are designed to determine the way to initialize the dialog flow [35, 61]. There are

- System initiative model (task-oriented model)
- User initiative model

- Mixed initiative model

System Initiative Model

First of all, the simplest dialog flow model of them is system initiative, or the task-oriented model, in other terms. This model means the dialog flow is oriented by task [62]. Using this model, once the call flow is driven by the system, the system will give the user commands one by one. Thus, it is unnecessary for the user to know the flow of application. Instead, the user just simply follows the instructions provided by the system to go through all the flows. So it is perfect for users who don't know the application very well. But this model is not suitable for customers who are quite familiar with this application, because this step-by-step model will take users a long time to go through all the pages. This model is very useful for voice-only applications [5, 35, 60].

For example, form-filling [59] is a typical task-oriented model. In our voice-activated Click&Dial system, we use this system initiative model to control the authentication dialog flow, since the process of logon is strictly fixed.

For example,

Prompt: Welcome to Bell.

Prompt: Please enter user name.

User: Mike (M-I-K-E).

Prompt: Please enter your PEIN number.

User: 123456.

User Initiative Model

In the user initiative model, the speech dialog flow is initialized and driven by the user instead of the system [5, 59]. Namely, users direct the system from one task to another by giving commands. A typical user initiative model is clicking to talk [59, 61].

In our system, we could propose adding buttons for every text input. In the authentication dialog flow, for instance, two buttons could be added for the user name and the PIN number for controlling how and when to enter the user name and the PIN number, respectively.

For example,

Prompt: Welcome to Bell.

User (action): Click the button (named username)

Prompt: Please enter user name.

User: Mike (M-I-K-E).

User (action): Click the button (named pin).

Prompt: Please enter your PIN number.

User: 123456.

Alternatively, the user can click the PIN button to enter the PIN number first, and then input the user name. This model makes the application more flexible, but the user is required to learn how to use this system.

Mixed Initiative Model

Unlike the general web applications, speech applications consist of various states to collect the spoken information entered by the user. The conventional dialog system only allows the user to get the information strictly following the system that provides information or commands piece by piece in a desired order. Thus, it is not flexible and convenient for customers to use this application.

As we discussed above, in the authentication dialog flow with the system initiative model, the system does not allow users to give multiple pieces of information such that users have to provide the user name and password separately.

However, with the mixed initiative model, users can request information by speaking their own words to the system, just like to a human. This model allows dialog flow to be controlled by the user as well as by the system so that it makes the speech application more natural and flexible.

Prompt: Welcome to Bell.

Prompt: Please enter user name.

User: My name is Mike (M-I-K-E). My PIN number is 123456.

As showed in above example, the user provides multiple pieces of information that include both the username and the password, while the system has not prompted the

command for the password. With this model, the system is able to recognize the phrases and then initialize the username state with Mike and the PIN number with 123456. Furthermore, it is also possible to recognize the input speech, as the utterance is not in the defined order.

For example,

Prompt: Welcome to Bell.

Prompt: Please enter user name.

User: My PIN number is 123456 and my name is Mike (M-I-K-E).

In this example, the user did not provide the information asked by the system; instead, the user gave the PIN number first and then his username without being interrupted by the system.

In fact, the semantic item object is used to determine the current state. In the above example, we create three semantic item objects for authentication flow:

- Welcome prompt
- Username prompt
- PIN number prompt

Every semantic item object has three states, as follows [61]:

- Empty
- Need to be confirmed
- Confirmed

The semantic item object will stay at the empty state until the system gets the corresponding answer from the user. The confidence score, which determines if the item object needs to be confirmed or has been confirmed, is set in advance. The recognition engine will return the confidence score once the system gets the information from the user. If the confidence score is lower than the desired score, the need to be confirmed state is set for the object. Then a confirmation is prompted by the system. Conversely, if the confidence score is higher than the desired score, the object is set to be confirmed. In other words, the system has already got the correct answer or information from the user, and the next action is triggered.

As we mentioned in the above example, there are three semantic item objects for different states. Since the welcome prompt is the prompt-only object, the welcome

greeting 'Welcome to Bell' will be launched when browsing this page. After the welcome prompt is played, it won't be launched again and is no longer active. Then the next prompt, the username prompt, is checked. If the state of this object is empty, it will be active to play. Otherwise, the system will keep on searching the next object with the empty state until the states of all objects are confirmed.

In addition, this system allows the user to switch tasks easily without having to start again from the beginning. It is of great benefit to users who are lost when they browse the application.

For instance, there are three ways to search the phone number in our system: search by name, search by Bell PEIN and search by phone number in turn. If the user chooses searching by phone number, when the system asks him to speak the phone number of the person he is looking for, but he cannot remember the exact number. In this case, it is unnecessary for him to go back to the main menu and then start again from the beginning. Instead, he can switch to another search method by saying "search by name" or "search by PEIN number".

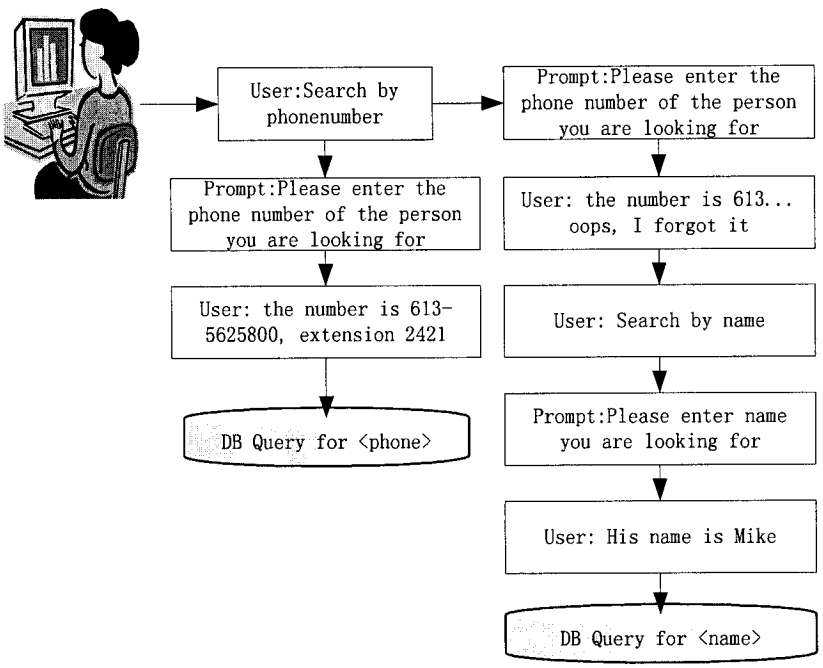

Figure 5-11 Mixed Initiative Flow

Figure[5-9] represents the flow of this case graphically. In short, mixed initiative systems make interaction between users and systems faster and easier.

5.3.6 Grammar Design

Grammar files are created to specify the possible words or phrases input by the user in speech recognition. There are two different forms of the grammar format defined by the World Wide Web Consortium (W3C): augmented BNF and XML syntax [63, 64]. In our project, XML syntax was recommended since the syntax of the XML format is simple and definite, and can be written using the XML editor provided by Microsoft Speech Application SDK.

Rules are the basic units of grammar, which can identify the following things to the recognizer to be listened for [64]:

- Words or phrases
- Patterns of words
- Languages of the utterance

The rules are used to specify the languages of the utterance, indicate the patterns of words and restrict the words or phrases defined in Items, thereby implementing the recognition. The grammar design greatly affects the performance of speech recognition, since the speech engine matches items corresponding to the grammar files. Users will be restricted to speak little if the grammar is too strict. However, having too many unnecessary items in grammar files will result in a lower rate of recognition. With this in mind, we design grammars carefully to make our application more flexible with high performance.

Basically, there are two ways we could put our grammar [35]:

- A grammar file
- Internal script embedded in code

As for the former, we save our grammars in files separately from other codes and store them in the grammar folder. A reference is used to indicate the path of these grammars in

codes. On the contrary, as for the latter one, the grammars are completely written in the codes.

Regarding the advantages of both forms, we embed the simple grammar in the codes to reduce the complexity of our applications. And most of our grammars are saved separating from other codes to make the structure of the application more clear and easy to maintain as well.

Preambles and Postambles

In the grammar files, the key words or phrases to listen for are listed. But practically, users will speak more normal and natural sentences, instead of just simple words. For example, in the main menu page, after the system asks the user to choose a menu, the application should allow the user to say different sentences instead of just saying the menu names, and it should recognize variants of similar phrases, such as "search please", "May I go to search?" and "I wanna go to the search page".

To achieve the requirements of the above scenario, preambles and postambles [35, 65, 66] are added to the main items in our project. Namely, we add the possible words or phrases that users may speak in front of the key words or after them.

Figure [5-11] illustrates the grammar for menu choosing. We add 'please' and 'go to' as preambles to the main phrase and also add a list with two phrases 'page' and 'please', at the end of the main items as postambles. Four options of main items are given by adding a list. As a result of this design, our application allows users to choose the menu not only by speaking the name, but also by speaking the sentences combining the main items with the preamble and postamble which is more natural. 'Please go to search page', 'search page', 'my option please', 'help please', 'logoff', 'go to search please' are all legal for choosing a menu, a corresponding page will be displayed according to the main items that were recognized rather than the preamble and postamble.

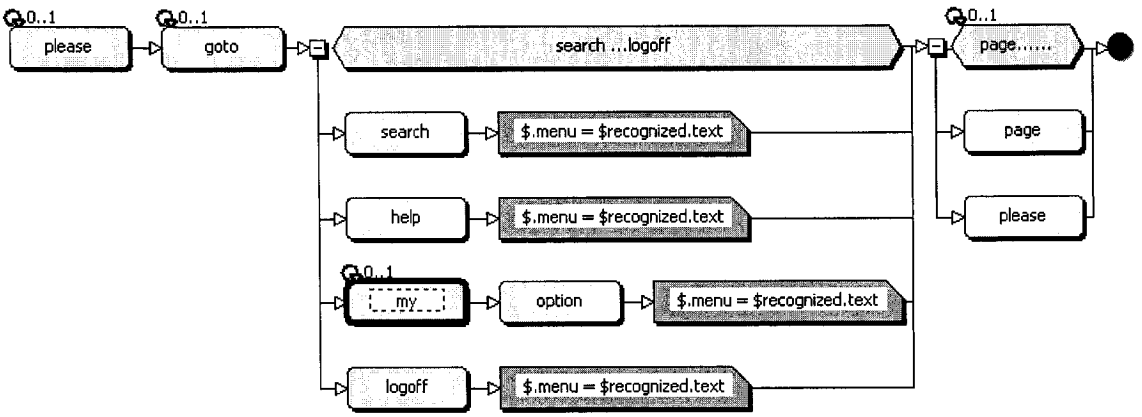

Figure 5-12 Graphic Grammar of Menu Choose

In our project, search is the main function of the Click&Dial system, which is going to be discussed in depth.

The search feature allows the user to get the phone number and call the person he is looking for by entering the username or the Bell pein number or entering the telephone number directly. As shown in the figure, at the very beginning, the system will automatically detect if users finish authentication. If so, the system will take the user to the welcome page, otherwise, user will be asked to log in first.

We will take the user's utterance and query for the results from the database. One or more results will be displayed and the system will ask the user to choose one. Before making a call, the user will be required to confirm the choice.

5.3.7 Speech Control

A very important step for building voice-enabled applications is to determine which type of speech controls to use and how many elements of SALT will be involved.

Speech control is designed for building voice-activated web applications while using the SALT markup language. Namely, it controls and manages the interactions between users and systems. Basically, there are three types of speech control [35, 59, 61]:

- Basic speech control: It is a representation of the main top-level elements of SALT, <listen> and <prompt>.

- Dialog speech control: It uses script to handle the confirmation and application flow.
- Application speech control: It consists of one or more dialog speech controls. This control is designed for collecting the common data that have special structures, such as currency, phone numbers and postal codes.

According to the requirements of our system, we adopt dialog speech control to handle the application flow using script. Reasonable semantic items are designed to hold the answer from users.

Figure [5-11] illustrates the high level of the searching process with the assumption that the user has already logged in.

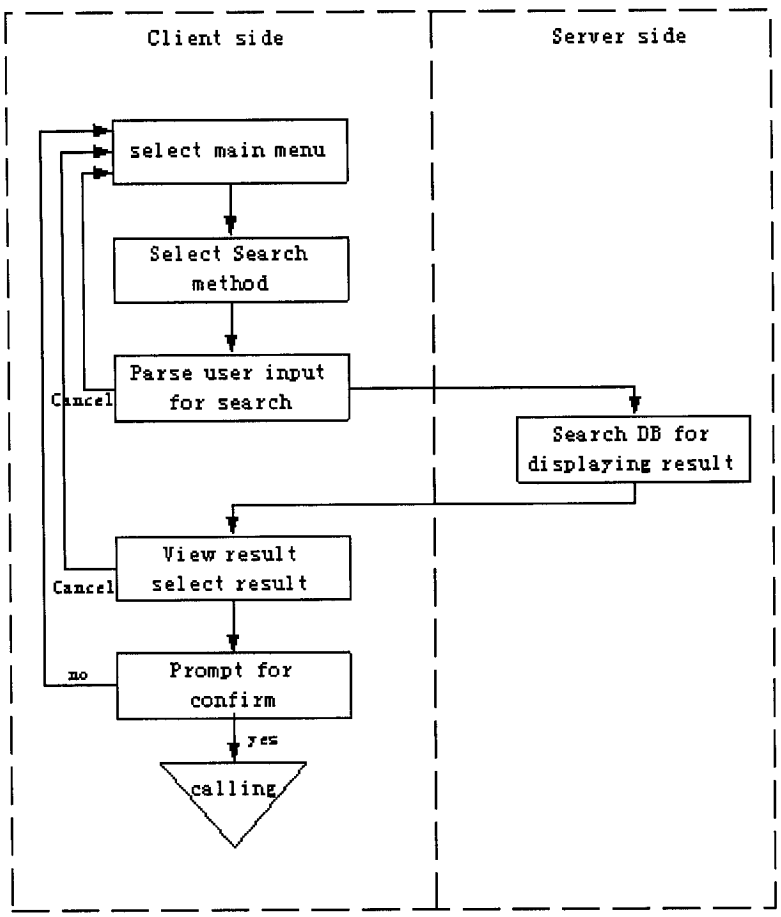

Figure 5-13 High Level of Search Process

Step 1: Select Main Menu

Once the authentication is done, the main menu page will be displayed in the browser. Several prompts are set to tell users what they should do with this step.

In order to make our application more natural, we classify these prompts into three different files. The low level of the main menu chosen is showed in figure [5-13]. We expect our application to speak to the user with different prompts according to certain situations. In this step, a welcome prompt is supposed to play while the user browses this page for the first time. If the user is sent back to this page from another page by giving a special command, the welcome prompt is skipped and the system will list the menus and ask the user to choose one of the menus directly. In the third case, both the welcome prompt and the menu list are skipped if the user is sent back due to the event of onnoreco and onsilence. To achieve this requirement, we separate these prompts into three different blocks or files and set the property of the welcome prompt to 'playonce'. Then, the system will determine whether to prompt the welcome or not according to the state of this prompt.

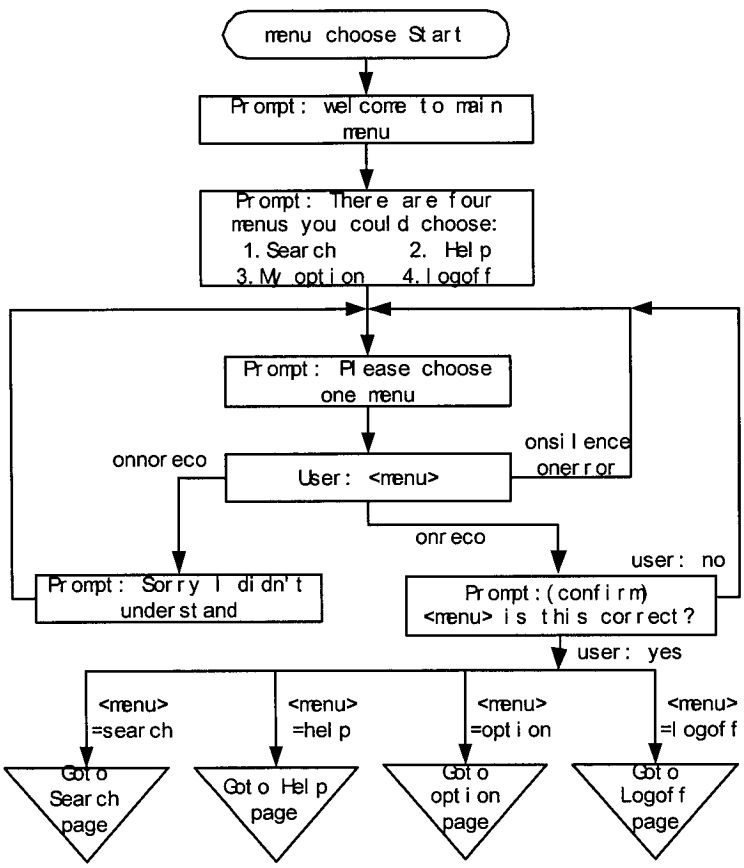

Figure 5-14 Low Level of Menu Chosen

Basically, the prompt could be designed in one of two forms[35]:

- Inline prompt: Embed the script that the system will read within two tags. Once the prompt.start() is called, the predefined text is synthesized and played back to the user.
- Wave files: In this form, the text we want to read is not written in the code directly. Instead, we give the URL of the sound file within the prompt tags. Once the prompt.start() method is called, the system will get the wave file by linking to the URL and play the pre-recorded sound back. One of the advantages of using this form is that the number of prompts can be reduced by discarding duplications.

So in this process, we create five prompts file to store the text. Among these prompts, the prompt 'Sorry, I didn't understand' is set in wave files. Since no matter in menu choosing page, searching page, or other pages does not matter in the menu, once the onnoreco event is thrown out by the system, the same prompt will be played back. Using the latter form, it is unnecessary to write this prompt every time. We just give the URL of this sound file where it needs to be prompted. The number of prompts is thereby reduced.

Next, we create listening objects for speech recognition or audio recording. In this case, we set multiple as the mode of this listening object to obtain and recognize the user input. Generally, a listening element contains one or more grammar elements, which are defined to specify the phrases we expect the user to say.

As shown in figure[5-11], the menu selection grammar contains a list of four acceptable menus combined with their preamble and postamble. Once the recognition process is triggered, speech detection starts to listen for menu selection from the user. When one of the menus is matched, the system will automatically return an SML document containing the name of the SML element, the text recognized and the value according to the recognized text as well.

For example, if the user chooses the search menu with the saying 'search please', the SML document with the expected name of the SML element <menu> and the recognized text 'search' and its value will be returned.

```
<SML confidence="1.000" text="search" utteranceConfidence="1.000">
        <menu>search</menu>
</SML>
```

Listing 5-2 SML of Menu Choosing

Different events will be fired corresponding to the result of the speech recognition. It is important to specify the event handlers to deal with the successful and unsuccessful results.

As we discussed in the chapter of listening mode, generally four possible events could be thrown out during the speech recognition process: onreco, onnoreco, onsilence and onerror. Since we set the listen mode as multiple, different events will cause different results. Thus, handling these events appropriately becomes significant. In our project, the same method is adopted to handle the onnoreco event. The user was sent back to the page with the requirements for user input, after closing the 'I do not understand' prompt played back. The onsilence event handles the case of no speech input detected by the speech engine during the initial timeout. So telling the user what he should do and asking the user to select the menu again is our solution. The onerror event will be thrown out if a serious or fatal error occurs during the speech recognition process. There is no explicit exception associated with it; we have to specify the way to handle it. Regarding this event as the same as the onsilence handler will make it easy to be solved. The event of onreco occurs when the utterance is successfully recognized by the speech engine. Namely, the utterance matches one of the menus and the confidence value is above the reject threshold. As we discussed above, a corresponding SML document will be automatically returned with the name of the semantic item and the recognized text.

Finally, we expect the user to confirm whether the recognized menu is right or not. Parsing the recognized value to prompt out is the key point in this process. Here, we use the bind element to bind values from speech input.

```
<salt : listen id= "menu">
        <salt:grammar src= "grammars/MainMenu.grxml" />
        <salt:bind targetelement="selectedmenu" />
</salt : listen>

<salt:prompt id="menuConfirm">
        O.K. your choice is
        <salt:value targetelement="selectedmenu" />
        Is that correct?
</salt:prompt>
```

Listing 5-3 Bind Element of Menu Choosing

The targetelement assigned with the value from the returned SML document is the one attribute of the bind element. We specify the targetelement with the name selectedmenu, as shown in the above list. Once the speech recognition is successful, the value of the spoken input is assigned to the attribute with the name selectedmenu. In the prompt block, the value is retrieved by giving the bind with the same name as the above targetelement.

Step2: Select Search Method

Once the menu selection has been done, the corresponding page will be displayed, and the user will be asked to choose one method for searching. Similar to the first step, this process only deals with the static page without interacting between the client and the server. So this step will be discussed briefly.

First of all, several prompts are set to determine what is said to the user. In this case, the system speaks to the user by listing the available methods for searching, such as search by user name, search by Bell Pein and search by telephone number. Another two inline prompts are designed for requiring the user to choose one of the methods and confirm. Since the wave file for handling the state of onnoreco has already been defined in the first step, a URL is just listed here. Again, we set the mode of recognition as automatic based on certain criteria. A grammar file is set up to define the phrases we expect to hear from the user, and it contains the main items of the three methods. The value of the semantic item with the name 'method' is captured from the user through the returned SML

document. As discussed above, it is necessary to specify the handlers to deal with the possible events.

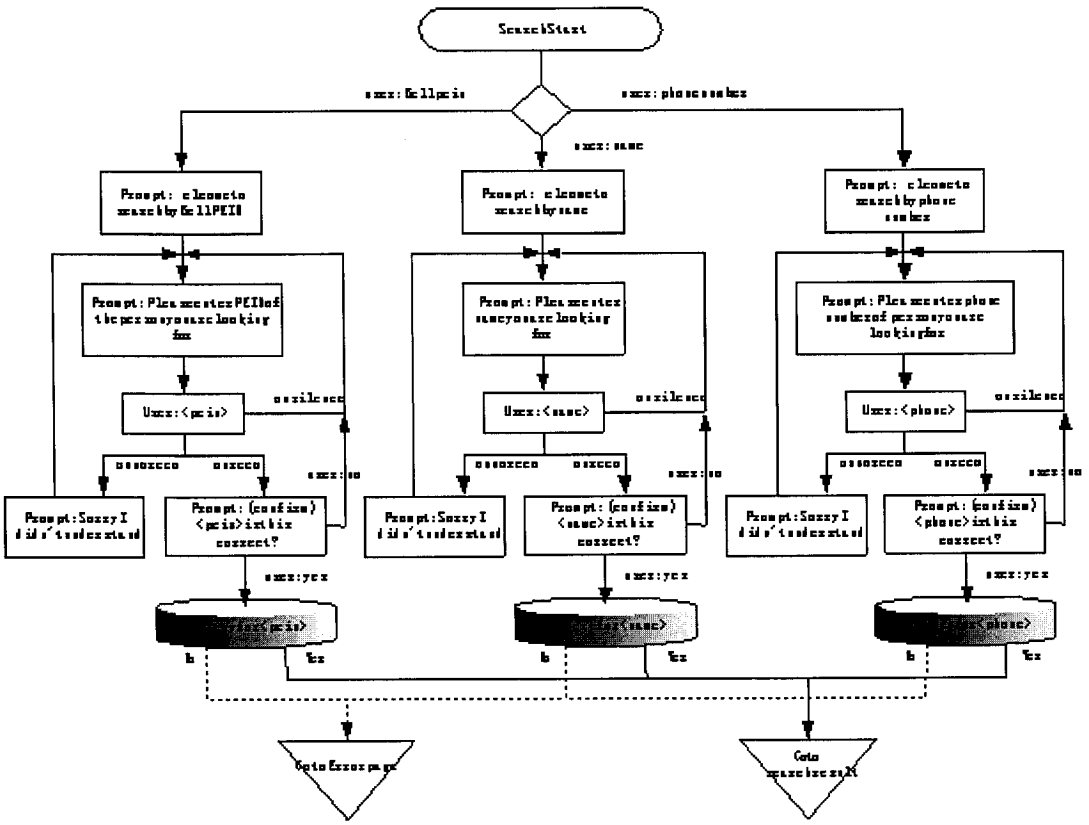

Figure 5-15 Low Level of Search Method Chosen

Step 3: Parse User Input for Search

As shown in figure [5-14], three possible flows are defined according to the user's choice. Except for the content of user input, all these three flows are greatly alike, so we gave an example with the assumption that the user chooses to search by name. The main points of the part from welcome prompts to confirm prompts including the mode of recognition, grammar design and the event handlers are the same as the points of menu selection that are described in great detail above; thus we are skipping this part to avoid the verbosity.

Once users confirm their utterance, we want to parse the input to the server side to get the corresponding results matching the user's input. At first glance, this process of parsing looks similar to the one in step one. Actually, where this parsing is operated makes the essential difference between them such that the above parsing process is handled only on

the client side. However, parsing the user input to the server side needs the interaction between the client and the server. Similarly, the bind element is used to retrieve the value from SML.

Step4: Search DB for Displaying Result

This step actually involves the backend components that we discussed previously. After the user submits the search form, the SearchAction calls the SearchProcessor to query the LDAP. LDAPUtil will first call the ServiceLocator to get the JNDI connection and then do the real query.

Step5: View Results and Select Result

After getting the results from DB, the results will be passed from the server side and displayed on the client side. Two possible pages are returned depending on the results. If no name matched is found in DB, a sort of error page is shown to inform the user of the possibility of wrong input. If more than one matching value is retrieved from the database, the system will output all the results and point out each value one by one. When too many choices are provided at once, it may make the user confused. So to avoid this problem, we assume that no more than 8 matching results will return to the user to limit the options in our case. Therefore, only one page is used to display all the result values so that the complexity of the results view is lowered down.

Usually, the results that satisfy the user's requirement could appear on the top of the result list, or the middle or even at the end of the list. In the first scenario, once the user gets the phone number what he is looking for, there is no need to prompt the remaining results. So a special function is defined to deal with this event on the client side. We define a listen object to keep on monitoring the input from the user. If the user selects one item from the list either by speaking or clicking with the mouse, the event of onselect is thrown out to terminate the current prompt by setting the state as true. Namely, the remaining options are flushed out without a prompt to the user.

In addition, the system takes advantage of the global command defined previously to cease the current process and then return to the main menu page. Again, the JavaScript function is written to handle the cancel command, which will be activated if the user says

'cancel'. Once that happens, the current prompt function is deactivated; all of the semantic items are cleared by setting their states to empty again.

```
<salt : listen id= "cancel" onreco= "Oncancel()">
        <salt:grammar src= "grammars/Cancel.grxml" />
</salt : listen>

<script>
        function Oncancel() {
                promptResult.stop();
        }
</script>
```

Listing 5-4 the Global Command: Cancel

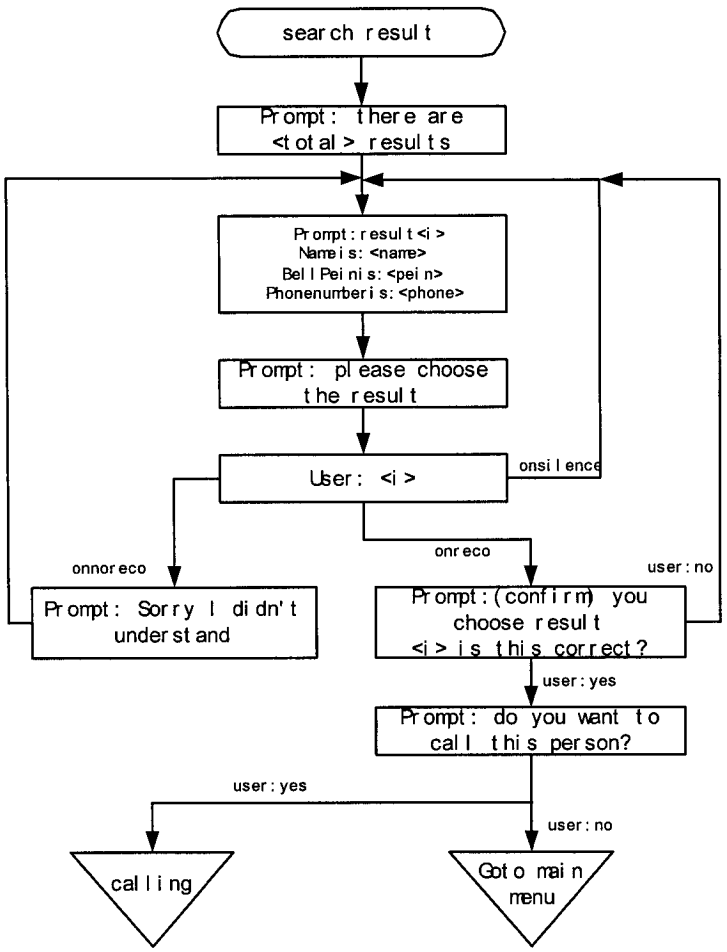

Figure 5-16 Low Level of Result Comfirm

Step 6: Prompt for Confirm

The last step of the search process is confirming with users whether they want to make a call right now or not. In the case where the user rejects the confirmation, the main menu page is reloaded on the client side. If the user agrees to make a call, the state of the semantic item is set to confirmed. Then the system establishes the connection to call the person for whom the user is looking.

Chapter 6 Implementation

The speech-enabled Click&Dial system is a web application, which is designed following the architecture. In this chapter, we will describe the steps to implement this system by seamlessly integrating voice and visual interfaces. The essential technologies for implementing the system, as well as the fundamental tools for building both voice and visual parts, are introduced.

6.1 Architecture of the Implementation

The high-level framework of this system is described in chapter 3. In this chapter, we mainly concentrate on the detailed process between the different components.

Since this is the implementation for a desktop scenario, we physically put the Speech Engine Services (SES) on the same side as client end. The Internet Explorer (IE) speech plug-in, which provides speech engine services, including a Speech Recognition engine and Text-To-Speech engine, is required to install on the client side.

Flows 1, 2 and 3 indicate the communication between the client and the server as shown in Figure [6-1].

1. In this multi-modal web-based system, the visual content is presented by the HTML. Users send requests to the server through HTTP.
2. Once users submit their requests by speaking, the voice input will be sent to the speech engine services to recognize first.
3. The recognized result from speech engine based on the utterance is sent to the Struts Controller layer through HTTP requests.

Flows 4, 5, 6 and 7 present the interactions on the server side. In our system, the web server and the application server reside on the same computer physically, although they are separated theoretically.

4. The controller Servlet class (Struts Action Servlet) is the gateway handling all incoming HTTP requests. The controller forwards each request to its appropriate action handler based on the rules defined in the struts-config.xml. Java bean is instantiated based on the user request and action.
5. After the process is done, the result is forwarded to the appropriate web page.

6. The JSP request contains JavaBeans components
7. If there is a requirement for querying, in this case, Java DataBase Connectivity (JDBC) is used to query the database.
8. Users get the response page according to their request and action.
9. Meanwhile, the required grammars and prompts, which are saved on the server side, as well as SALT, are parsed to the speech engine.
10. According to the request type, the corresponding response will be prompted to users through Test-To-Speech engine.

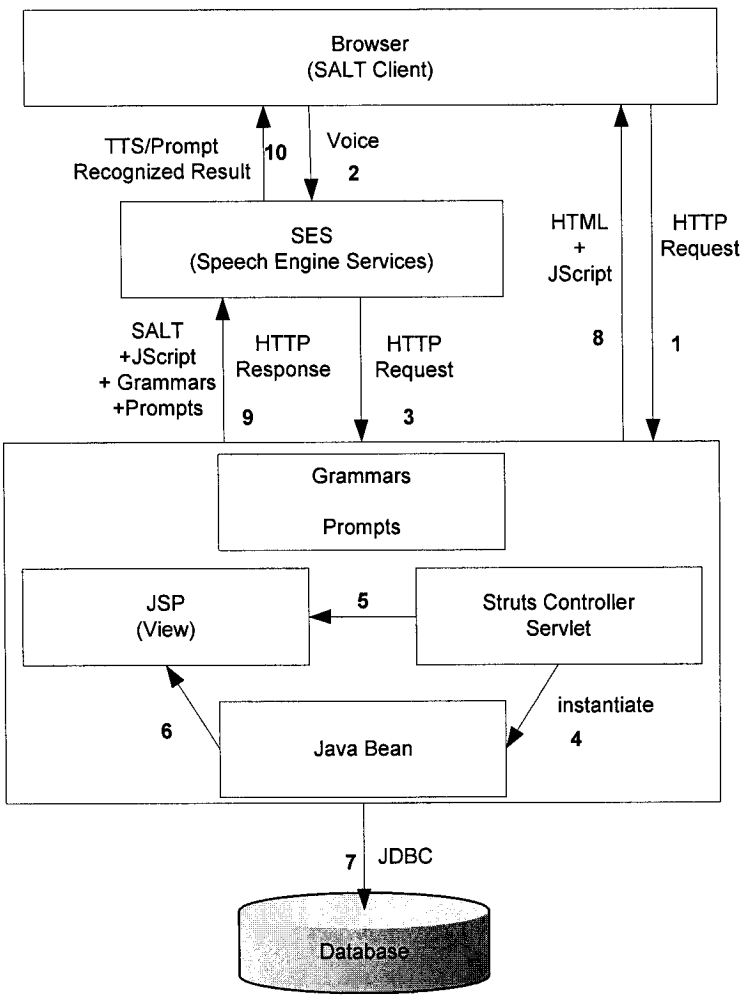

Figure 6-1 Implementation Architecture of the Voice-enabled Click&Dial System

6.1.1 Struts

There are several Struts-related modifications in the source code. Since Struts itself is a complex software system, only the most important modifications are listed, as follows.

Struts Tags

The W3C's HTML specification does not provide a data binding mechanism for the back end modules. To solve this problem, Struts provides a comprehensive facility for building, interacting the forms, based on the Custom Tag Library facility of JSP [39]. For example, the following Struts HTML tag is used:

```
<html:text property="userName"/>
<html:text property="password"/>
```

Listing 6-1: Struts Tag for User Log on

instead of the HTML tag:

```
<input type="text" name="userName" value="<%=formBean.getUserName()%>"/>
<input type="password" name="password"
                                    value="<%=formBean.getPassword()%>"/>
```

Listing 6-2: Sample HTML Tag for User Log on

Page Flow

Before Struts is introduced, the normal request process flow works in this way: the request will be processed by a servlet, and after finishing the backend operations, the servlet will forward the request to the next JSP page and send it back to the client. The navigation sequence is hard-coded in the servlet code. Struts make it possible to define the navigation rule out of the code. A XML configure file named struts-config.xml is created for this purpose. Here is a part of the struts-config.xml showing the page flow definitions:

```
<action     path="/searchAddBook"
                type="com.bell.corpdir.actions.SearchAction"
                name="searchForm"
                scope="request"
                validate="true"
                input="/home.jsp">
        <forward name="success" path="/search.jsp"/>
</action>
```

Listing 6-3: Code Segment for struts-config.xml

In the Java code, we just call:

return (mapping.findForward("success"));

In this case, the Struts framework will find the corresponding "success" forward file "search.jsp" and send it back to the browser in the above example.

Layout Management

Tiles is a famous layout manager and template library in the Struts package. It allows you to construct views by combining various "tiles". Its idea is to divide the web pages to smaller components like header, menu and footer, which make them reusable. Since each page extends from the template page, it makes the look and feel consistent across the whole web application. At the same time, Tiles is powerful enough to give the user full ability to set the attributes dynamically [39].

See Appendix Tiles for details.

Validator

With the help of the Struts' validation framework, the validation operations are greatly simplified. There are two options: using the client side validation with JavaScript or the server side validation. It also supports the regular expression, which makes it even easier to validate a string [39].

See Appendix Validator for details.

6.1.2 Handling Security

The application does not support self-registration and requires an administrator to verify a user before creating an account. A key design choice was to manage security using the container's functions. There are three J2EE container authentication mechanisms: HTTP basic authentication, SSL authentication, or form-based login [51]. The Click&Dial web application uses the last one. Since JAAS is a standard and the application server supports it, implementing security is mainly configuration work rather than coding.
See Appendix Security for details.

6.1.3 Initialize Parameters

Some parameters need to be initialized before the web application is ready to accept the request. Normally an InitServlet will be created for this purpose and it's guaranteed that the parameters will be initialized before the web application is running. To do this, we need to override the init() in the HttpServlet class.
In web.xml, config the load-on-startup as the example below:

```
<servlet>
  <servlet-name>initialize</servlet-name>
  <servlet-class>control.InitServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Listing 6-4: Code Segment for Initializing

6.1.4 Logging Tool

Log4j is used in the Click&Dial as a common logging tool to implement the flexible logging service. With the help of Log4j, it is easy to enable logging at runtime without

changing the application binary. Instead of using System.out.print, using a logger hierarchy to control the log statements reduces the amount of log output. There are several levels of message defined by Log4j, such as debug, info, warn, error and fatal [53].

See Appendix Log4j for details.

6.1.5 Ant

Ant is adopted as the build tool for the Click&Dial system. Ant configuration files are XML-based so that it is unnecessary to write the traditional shell command. In additional, since it is written in XML and processed by Java, the Ant script is portable across different platforms [54].

See Appendix ANT for details.

6.1.6 CSS

Cascade Style Sheet (CSS) [55] provides a standard way to manage the look and feel of the web page like font size, font color, background color, etc. It will help make the web pages look consistent and easy to maintain. For example, normally the font size, color will be defined in the CSS file. Once it needs to be changed, just change the CSS itself, and the settings will be applied to every page that includes the CSS.

6.1.7 Integrated Development Environment

Eclipse is chosen as the Integrated Development Environment (IDE) for building the web component of the Click&Dial system. Eclipse is a highly extendible develop tool. Its plugin framework allows other software to be plugged in to the Eclipse very easily. The plug-ins are optional but will definitely make development easier. The following Eclipse plug-ins are recommended to be installed in the development environment [56]:

- Aston: provides many code template
- Easystruts: helps developers create struts classes and modify struts config file automatically
- Sysdeo: can start, debug and stop Tomcat within Eclipse

- Lomboz: has useful J2EE project templates and wizards

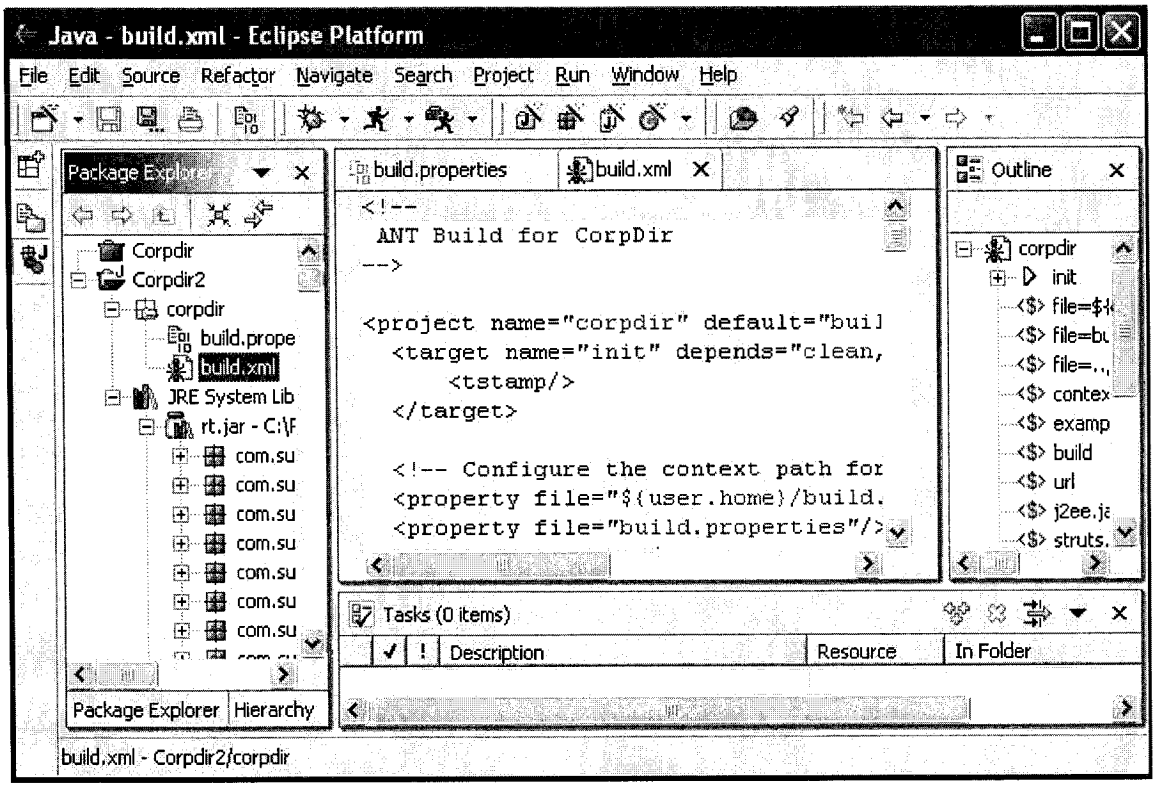

Figure 6-2: Eclipse Platform User Interface

6.1.8 Version Control

To manage the source code, version control software is mandatory for this purpose. CVS, which is free software bundled with many Linux releases, is adopted as version control tool in our project. It's the dominant open-source network-transparent version control system [57, 58]. CVS is useful for individual developers, as well as large distributed teams.

6.1.9 Build Procedures

In developer's machines, Eclipse is used as the IDE and the build tool. In the test server, a complete build and deploy process includes the following steps:

1. Check out the source code from CVS
2. Compile source code
3. Unit test

4. Build the war
5. Deploy it to application server
6. Integration test
7. Report the build results

Normally the above process will be executed automatically and daily. Ant script is created to execute these tasks.

The CVS build environment is shown in the following figure:

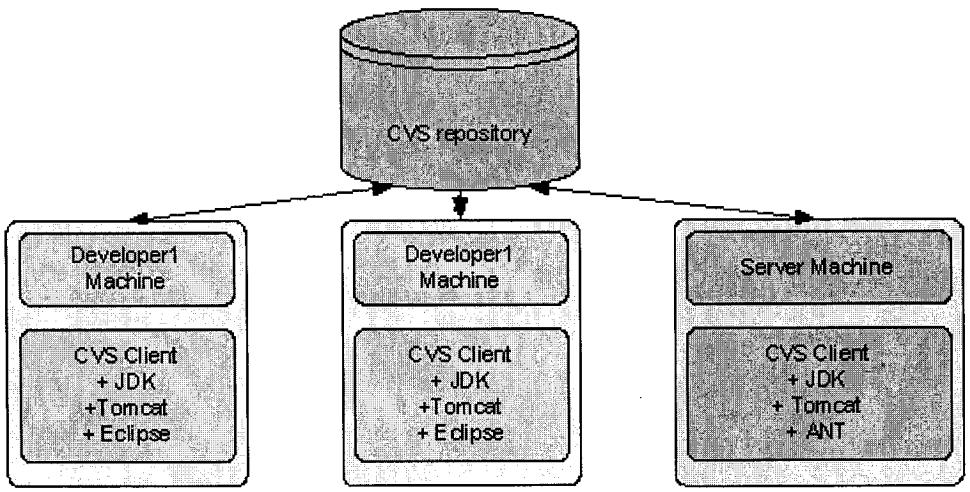

Figure 6-3: CVS and build tools

Chapter 7 Test and Evaluation

The voice-enabled web application is more difficult to test than the traditional web applications with only a visual interface. The strategies in design, coding and testing greatly impact scalability, functionality and high performance, which are the most important characteristics of a web application. Evaluation plays a crucial role in speech-enabled web applications, both for system developers and for end users. In this chapter, we will discuss the techniques, methodologies and tools used for evaluating the scalability, functionality and performance of the system and analyzing these characteristics based on the testing results.

7.1 Testing Overview

A web application normally contains hundreds of individual pages, and most of them are generated dynamically. Moreover, each page has a lot of internal and external hyperlinks and elements such as tables, forms, script, images and links. The errors, including wrong or dead links, missing content, difficult navigation and uncompleted functions, make users discontent. To avoid such problems, functionality test is used to verify whether the application performs all the required tasks correctly [68]. Namely, it evaluates whether the required functions are completed successfully.

In our project, we do not use any tools for functionality tests currently, because the application not only has the visual interface, but also the voice interface. So far, there are no tools good enough to test the dialog traversal. A set of tests is built manually according to the application requirements manually.

Scalability testing is defined as testing the server or application in a way that is considered operationally normal with a normal-to-heavy number of concurrent connections [69]. Performance testing includes both scalability and stress testing that is defined as testing the application in a way considered operationally abnormal to assess how the application performs.

With regard to the application with speech activity, the tests are mainly divided into two parts: speech performance like grammar validation and integrated system performance. Different tools are used in different test parts that are discussed in detail.

7.2 Speech Performance

Typically, high speech performance is the main characteristic for voice-enabled applications. There are bunches of testing types concerning the speech performance needed to be tested, such as dialog traversal, grammar validation, recognition performance, and so on.

At present, there is no special tool for dialog traversal tests (namely, dialog flow tests) as we mentioned above. We just simply build a set of test flows and go through the web application manually to check whether all the functions and pages we defined can be accomplished and reached through voice.

7.2.1 Recognition Performance

In our application, speech is recognized using the speech engine provided by Microsoft, which is combined with Microsoft Speech Application SDK 1.0. In order to know whether this speech engine is good enough for our application, it is necessary to set up an evaluation test.

Evaluation plays a key role in speech recognition processing. Typically, the performance of the speech recognition systems is measured in terms of a word error score, E (in percentile) [19, 70],

$$E = \frac{S + I + D}{N} * 100$$

Where:

N is the total number of words in the test;

S is the total number of substitutions;

I is the total number of insertions; and

D is the total number of deletions.

The performance of the systems depends on several factors. For example, the different analysis conditions could cause different error rates. In addition, the speed of speech recognition is another main criterion for evaluation.

With these in mind, we set a model with 200 utterances under a modern office environment where there may be someone talking close to our cubicle, but not loudly. Several tests have been done under the same environment with the same speed of input during the different time periods. Each test has been measured and recorded, and the average value has been calculated.

As a result of speech recognition performance testing, error rates (E) of below 8% for a wide range of word vocabularies are obtained, which is low compared to other products. This error rate is obtained with a normal microphone. Actually, a high-quality microphone, for instance, with an acoustic noise-canceling element and DSP processing for noise reduction and signal enhancement will greatly reduce the error rate. Additionally, in our application, we restrict the words or phrases the user can speak to the system instead of allowing the user to use a wide range of vocabulary. Furthermore, the user browses the speech-enabled application by giving the command with simple words or short sentences rather than a sequence of several long sentences, as we use in the test model. So with all these additional constraints on the utterances, the recognition performance is significantly improved in the application.

7.2.2 Grammar Validation

User input in speech-enabled applications consists of the words or phrases a user speaks through a microphone. However, the words or phrases users can speak are specified using grammars, which were discussed in the previous chapter. Before the application is deployed, we have to find the balance for the application between flexibility and the rate of recognition so that clients can speak more naturally without losing the high performance of recognition.

We use speech grammar editors to validate the grammar files and verify if the files contain valid XML codes, which follow the W3C speech recognition grammar specification V.1.0.

First, we create bunches of words or phrases we expect the user to speak to the application for each grammar file. Then we successively enter the string containing the words or phrases that are defined in advance to test the file. If the grammars are able to

recognize the testing words or phrases, the results SML will be displayed in the window as shown in figure [7-1].

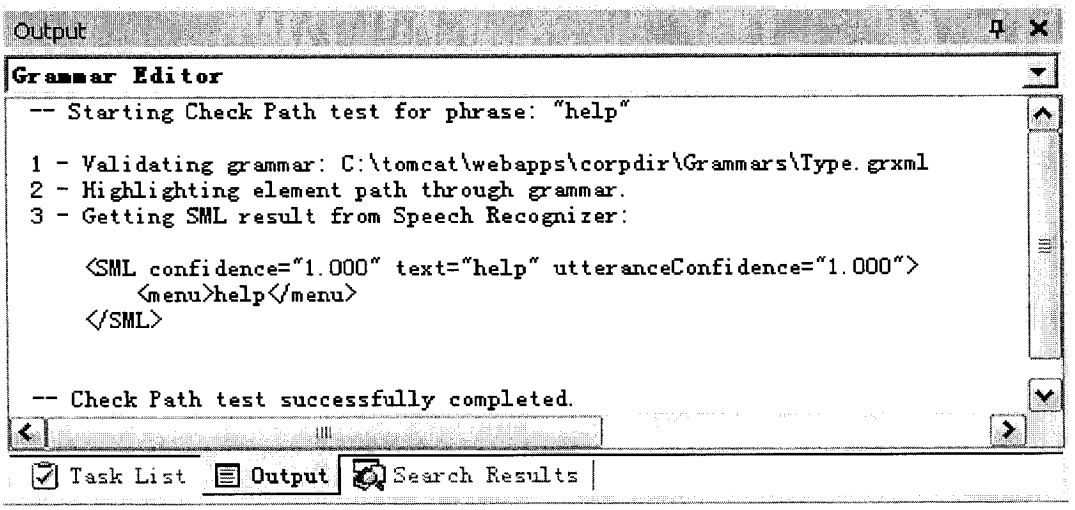

Figure 7-1 Grammar Validations

If the grammars are not able to recognize the words, the error message will appear in the output field. For example, we define the grammar file named 'type' to specify the words or phrases the user could speak to choose the menu. We anticipate that the utterance, input by the user, only contains one main item. In other words, if the user wants to choose the menu, he can only say one of the four options: search, help, option or logoff, rather than two or more. Otherwise, the error message will be output to the user. Like in this example, we enter the phrase 'help search', which includes two main items, and the SML output failure is shown to us as we expected.

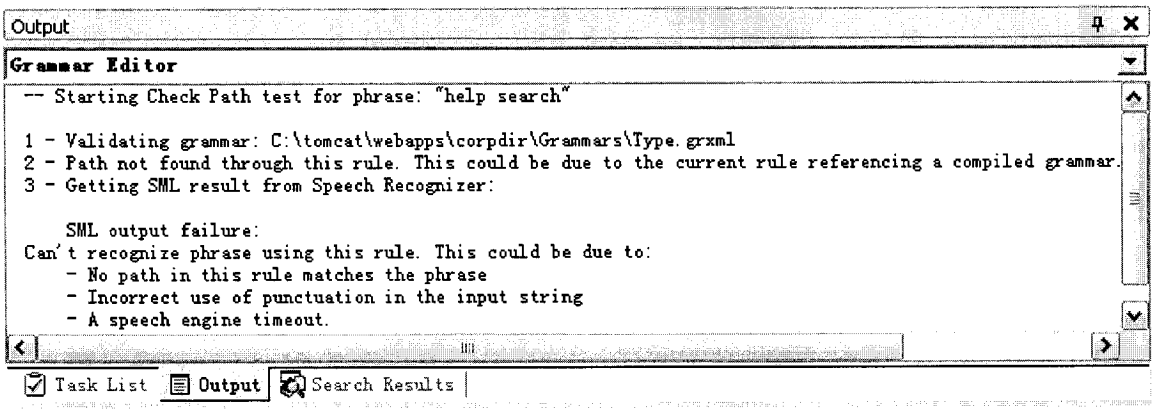

Figure 7-2 Grammar Validations Error Message

However, the most important point for grammar evaluation is to test if the grammar files can recognize all the phrases we defined and if SML with appropriate semantic value can be returned. Incorrect rule definition could result in the failure of phrase recognition. Based on the test results, we modify the grammar files to primarily satisfy the user's requirements by changing the property of the grammar, including setting the appropriate preamble or postamble, and attribute of items, such as repeat times and weight.

Using the same method, we test all the grammar files to ensure that they are valid. Thus it greatly reduces the recognition errors resulting from the invalid grammar passing to the speech application. Therefore, speech performance is considerably improved.

7.3 *Integrated System Performance*

The Microsoft Web Application Stress Tool (WAS) [72] is used to accomplish the testing of the performance of the application. The Microsoft WAS tool is designed to realistically simulate a large number of users requesting pages from a web application. Namely, this tool simulates to create an environment that is much closer to the real one so that we can find the problems prior to deployment. The performance information of the web application is collected through testing.

7.3.1 Test Scenario

The goal of testing is to check for the load of the application under the following scenario:

- Without throttle bandwidth (check for maximum load of the application)

7.3.2 Test Environment

The machine with the hardware and software specifications for testing is represented in Table [7-1] as follows:

Machine	Hardware	Software specifications
Server	CPU: Intel Pentium 4 2.80 GHZ RAM: 512M HD: 40GB	Windows XP Click&Dial application LDAP
Client1	CPU: Intel Pentium 4 2.80 GHZ RAM: 512M HD: 40GB	Windows XP Internet Explorer with speech Add-In Web application stress tool Version:1.1.293.1

Table 7-1 Test Environment

The test is performed in a lab environment. The Click&Dial application resides on the server with the web server installed. The LDAP is run on the same server physically, but is separated from the application server logically. All of the machines are located within the same Local Area Network (LAN), giving a maximum bandwidth of 100Mbits/s.

The Web Application Stress tool is installed both on the server machine and the client. Since the test is performed on LAN with a bandwidth of 100Mbits/s, the performance is quite similar for both local and remote tests. So only one of the results is analyzed in detail below.

7.3.3 Testing Methodology

The Microsoft's Web Application Stress (WAS) tool [72] is designed to simulate many users hitting the application simultaneously.

First, we create a test script named Click&Dial test script either manually or by recording to capture the content. In order to simulate more realistically, we use record mode to capture the browser session using IE going through the whole application. This tool captures all the content and requests as we browse through the application, including all the links, requests, form submissions and so on. The settings for each trial are shown in figure [7-3].

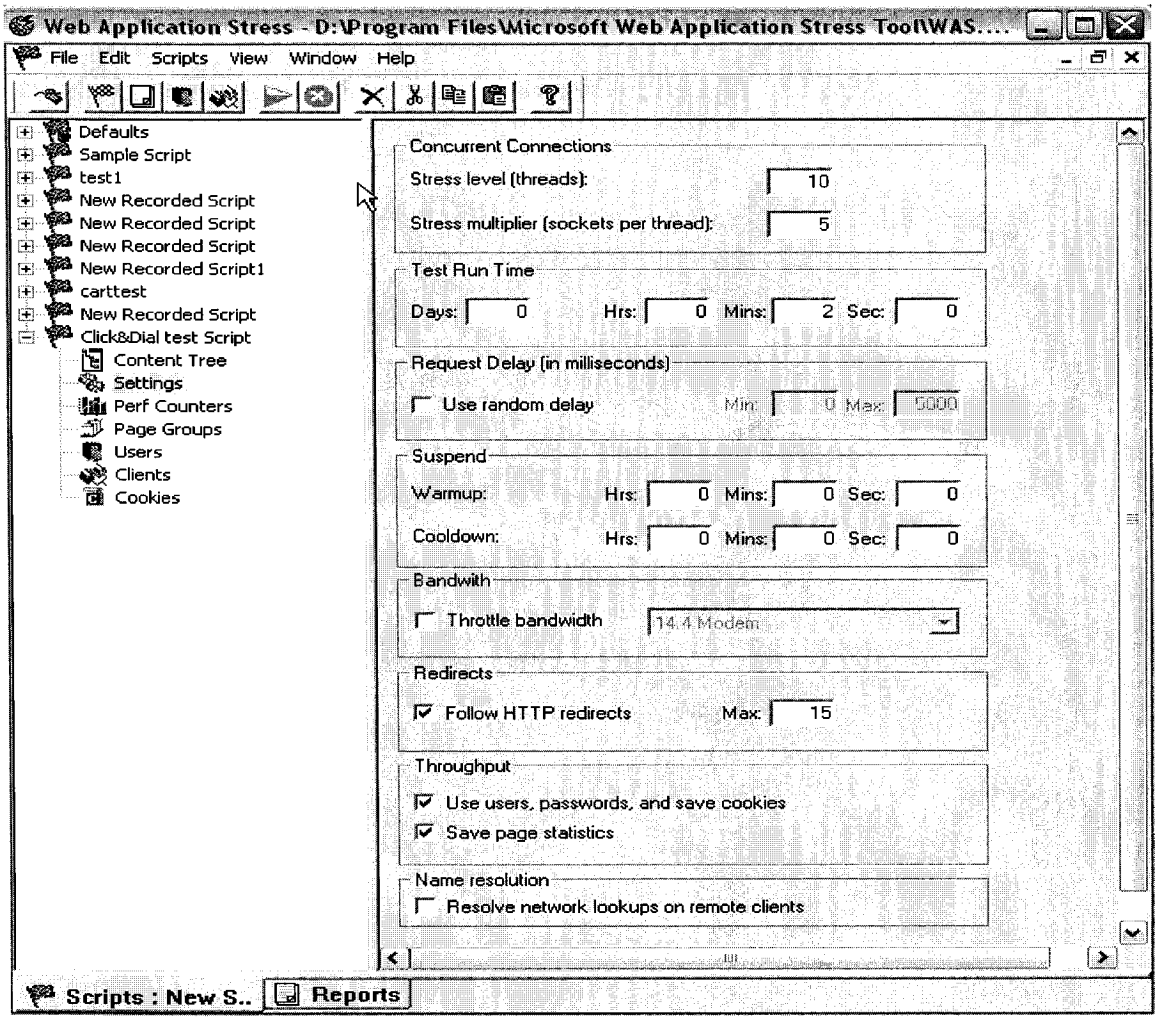

Figure 7-3 Web Application Stress Test Setting

We can simulate different situations by changing all the variables to test performance under different load levels. To achieve the appropriate load factor for our application, we change the number of concurrent users by setting the threads and stress multiplier (sockets per threads). The property of the stress level determines the number of threads that will be run by WAS to hit the application, while the property of the stress multiplier determines the number of sockets that are created on each of the threads [69]. The total concurrent clients equals stress level times stress multiplier. A maximum of 200 user accounts are set on the WAS client machines, so we change the number of concurrent connections varying from normal to heavy (1 to 200) to achieve the bottom line for the application. Two minutes were specified for each trial to run the test. The WAS tool also

provides the setting for request delay, which allows a more realistic simulation [69]. In general, it will take the user a couple of minutes or even more to go through one page, then link to the next one. So the request delay is reasonable for simulation. But we aimed at stressing the server to find the appropriate load in an abnormal way, so this option is unchecked in our testing.

Bandwidth is another important factor that impacts the performance of the application. We monitor the traffic that is generated both on the outgoing and incoming flow by setting the throttle bandwidth, which could be limited. In our program, the scenario without throttle bandwidth was tested for overloading.

Since we use HTTP redirects to forward user requests to the next page, the option of following HTTP redirects is checked in the testing by specifying 15 as a maximum number of redirects that avoids the endless loop.

We also select the two options of throughput: use users, passwords and save cookies, and save page statistics that capture cookie values associated with the active users and capture the statistics of the stress respectively.

Other options are set by default, such as suspend and resolve network lookups on remote clients.

7.3.4 Data Analysis

Overview of Report

Figure[7-4] and Figure[7-5] present the typical test report which includes eight parts: overview, script settings, test clients, result codes, page summary, page groups, page data and performance counters in turn. This sample report is generated under a special setting such that we set 50 clients (10 threads / 5 sockets) running at 2 minutes without a request delay and throttle bandwidth. The vital information that is generated from testing is the number of hits, requests per second, TTFB Avg and TTLB Avg:

The number of hits means the total number of requests hitting the server during the test period [72]. Similar to the number of hits, requests per second is the average requests per second in the test duration. TTFB Avg and TTLB Avg indicate the time clients have to wait to send the requests to get the required pages; where TTFB Avg shows the total time

from the initial request to when the first byte is received, TTLB Avg shows the total time from the initial request to when the last byte is received [72].

In this case, the total number of hits is 5964 for the two minute test, namely,

Requests per second = number of hits / run length

49.70 = 5964 / 120(seconds)

It indicates that the average number of requests the web server processed is 49.70 per second, which is impressively high. But we have to keep in mind that the value does not only include the number of hitting the backend server, but also covers all requests for static pages including images, and so on.

The averages of TTFB and TTLB are both less than 1 second, which demonstrate that the reply for receiving the page is pretty fast, as we anticipated.

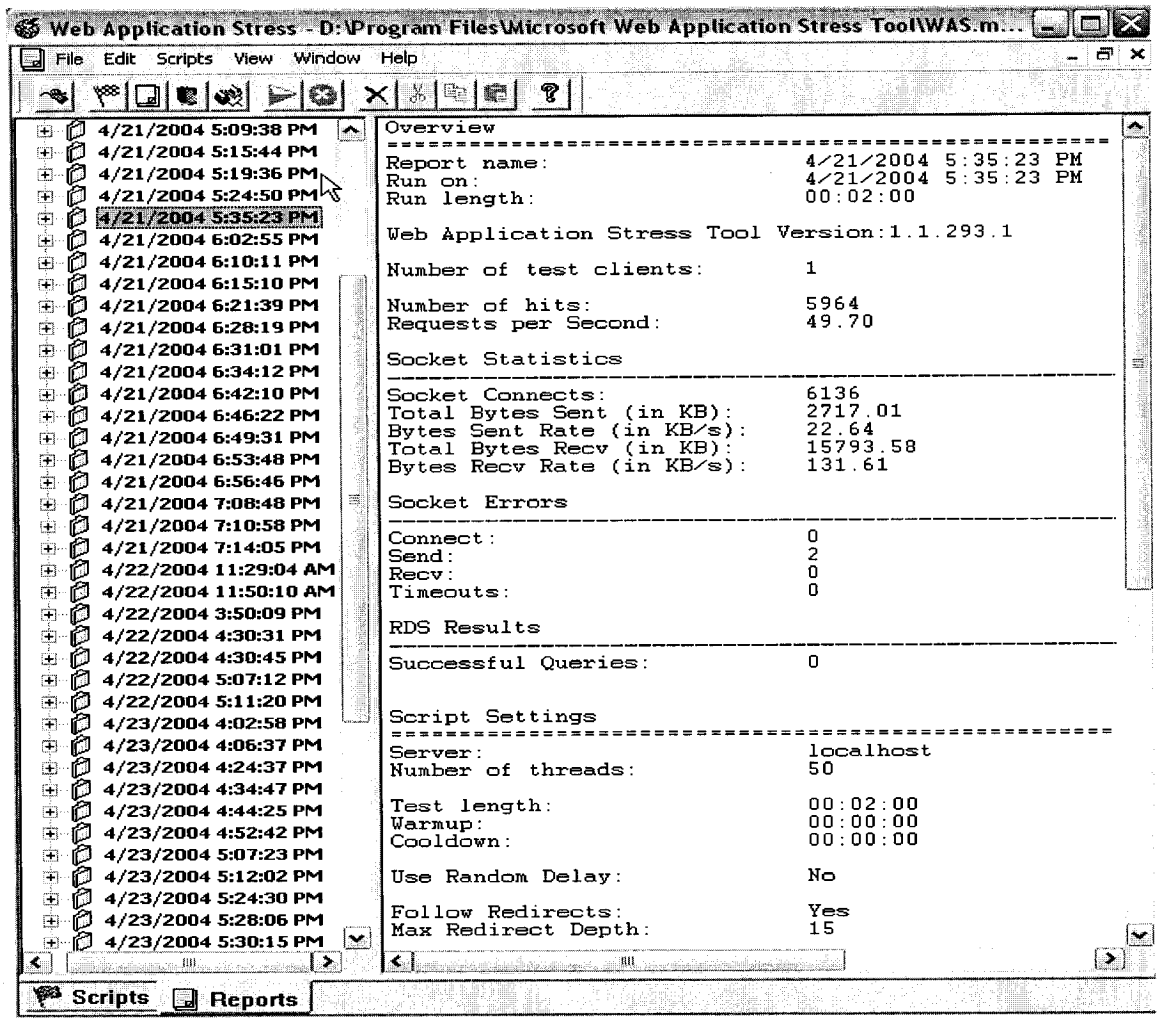

Figure 7-4 Performance Test Report

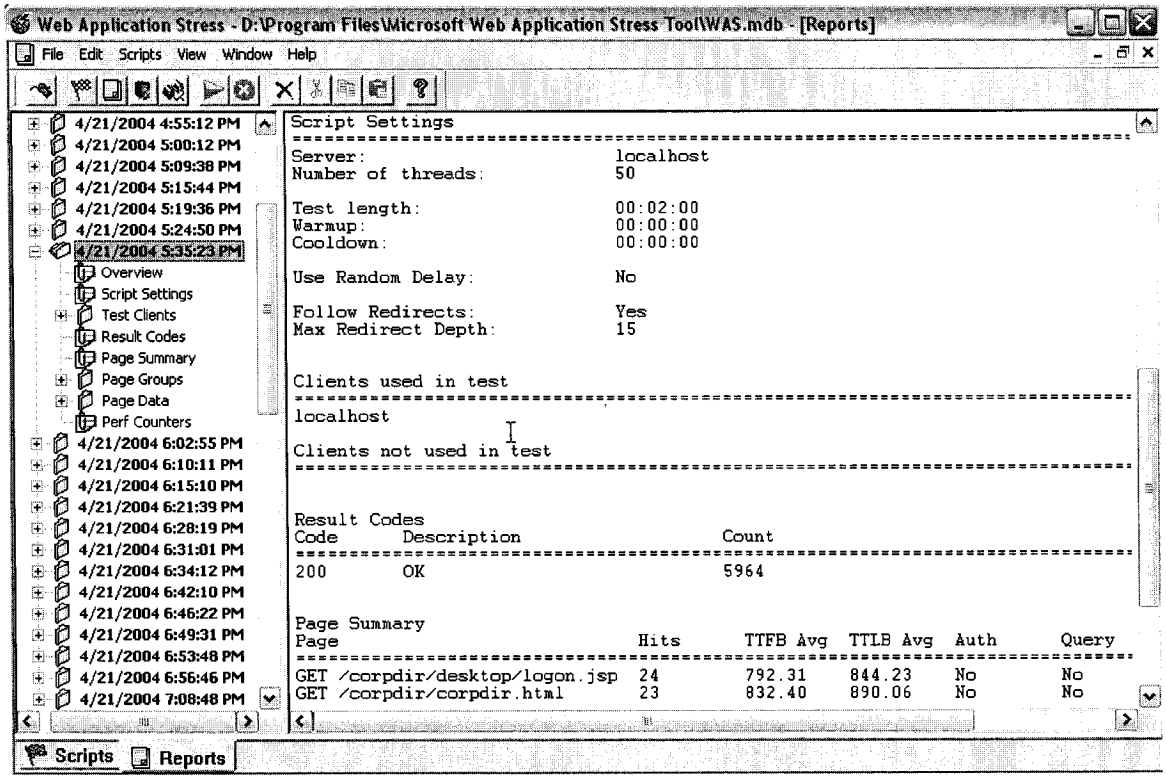

Figure 7-5 Performance Test Report (Continue)

In order to find the maximum load the server can support, we test the application under different conditions increased from 1 to 200 connections by setting the stress level and the stress multiplier. We keep on testing the application over and over at intervals of 10 connections with the duration of 2 minutes. One report with all the result data of the test was generated by one trial. We list load levels for all test cases and record the corresponding requests per second. At the same time, CPU usage is also monitored and recorded in percentile. The figure [7-6] and figure [7-7] show the testing results of requests per second and CPU usage respectively within the LAN. When the server stress load is low, the number of requests the server can handle per second is impressively high; it reaches about 440. As the stress load is increased by increasing the stress load setting, the number of requests per second decreases quickly. It is notable that the decrease almost stops after the concurrent connections increase beyond 25 and the requests level off at approximately 52 per second. Referring to the figure of CPU usage, we find that the shape of the curve is almost the same as the one for RPS (Requests Per Second). With

less than 20 continuous connections, the CPU is running at close to 100% utilization. And it drops down quickly as the simultaneous thread increases until it is beyond 30. Then the usage of CPU is smoothly increasing from 9% to 13% utilization. It is a strange phenomenon that was very hard to explain when we first saw this result. Nevertheless, we found the reason by comparing the curves of RPS and CPU usage. Under the relatively low load conditions, WAS sends bunches of concurrent requests to hit the server. In order to handle all these requests, the application server has to use up all the resources, especially the CPU, to achieve the extremely high performance. It is affected by RPS much more than by connections. As the concurrent thread increases, the maximum requests per second the server can handle are decreasing, which results in the decrease of CPU usage. Once RPS trends to be stable, CPU usage is mainly impacted by concurrent connections instead. In practice, it is impossible to have such a huge amount of requests hitting the server under less than 20 connections. So using up the CPU completely won't happen under these conditions.

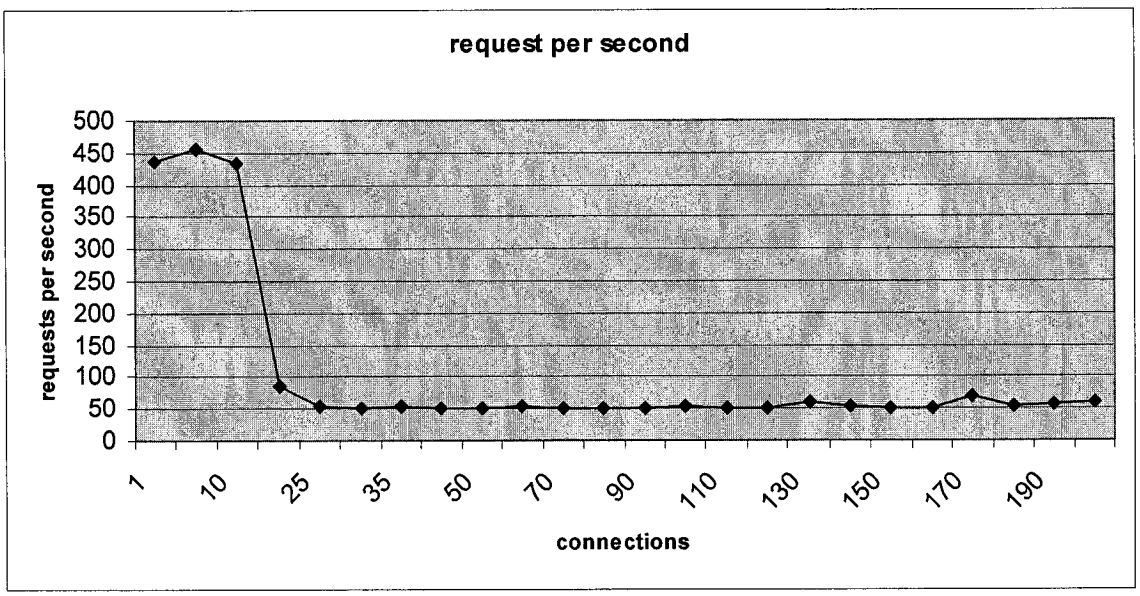

Figure 7-6 Test Results of Requests per second

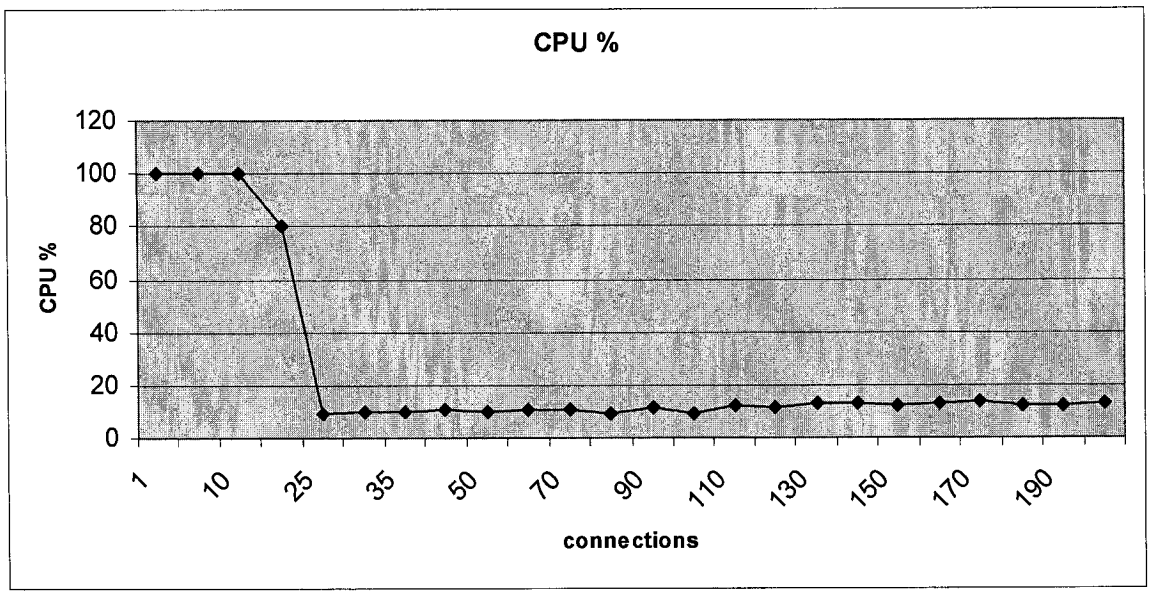

Figure 7-7 Test Results of CPU Usage

In addition, the scalability of the application is also tested and analyzed using the same testing environment. The response time, including both TTFB Avg and TTLB Avg, for every page is recorded under certain load conditions. For example, under a load condition of 50 threads running at the same time shown in Figure [7-5] above, there are a total of 24 times hitting the server to request the logon page sent by WAS during the test period. The average time from the request for the corresponding page till WAS receives the first byte of data (TTFB Avg) is 792.31 in milliseconds, while the average time from the request for the corresponding page until WAS receives the last byte of data (TTLB Avg) is 844.23 in milliseconds. The TTFB and TTLB for the second page, which is requested about 23 times, is 832.40 and 890.60 in milliseconds respectively. There are actually dozens of pages being requested, but we only list two of them by skimming the remains to avoid wasting space. Then we calculate one average value for all TTFB and TTLB Avgs generated from all pages including both the Get and Post methods. In other words, we record one average value of TTFB and TTLB for a specific load level like 50 threads simultaneously. As we increase the load condition from 1 to 200, the corresponding average value of TTFB and TTLB is recorded and calculated. Figure [7-8] and figure [7-9] show the TTFB and TTLB chart that is plotted based on the results of the test. As we

expected, the average value of TTFB tends to increase linearly as concurrent thread is increasing, as does the average value of TTLB. Up to 70 threads running at the same time, the average value of response time is less than 1 second, which demonstrates the high scalability provided by the application. In other words, almost all of the requests are accomplished within 1 second. However, it is notable that the application does not perform very well, as the simultaneous thread is beyond 170. We have to keep in mind that if the number of TTLB goes over 3 seconds, it probably keeps clients waiting too long for the corresponding page. As a result, the quality of the application is weakened.

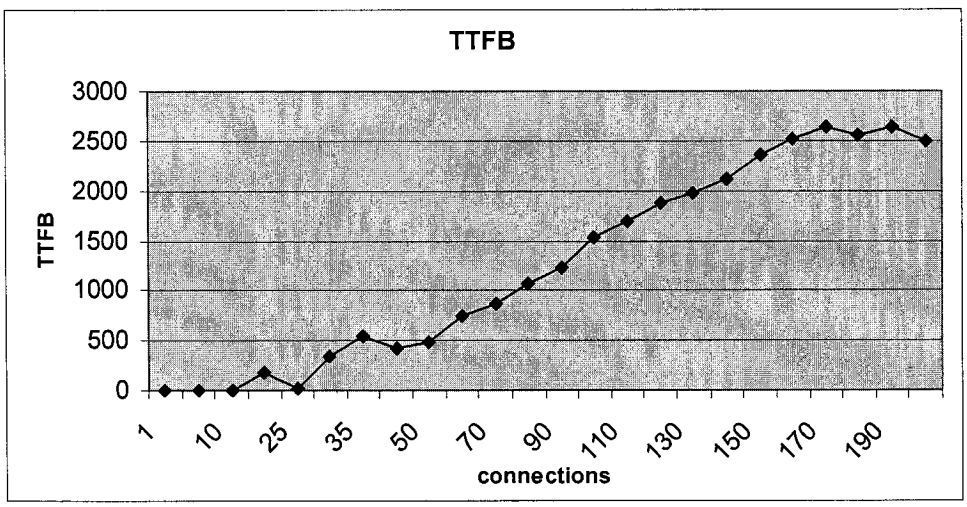

Figure 7-8 TTFB Results

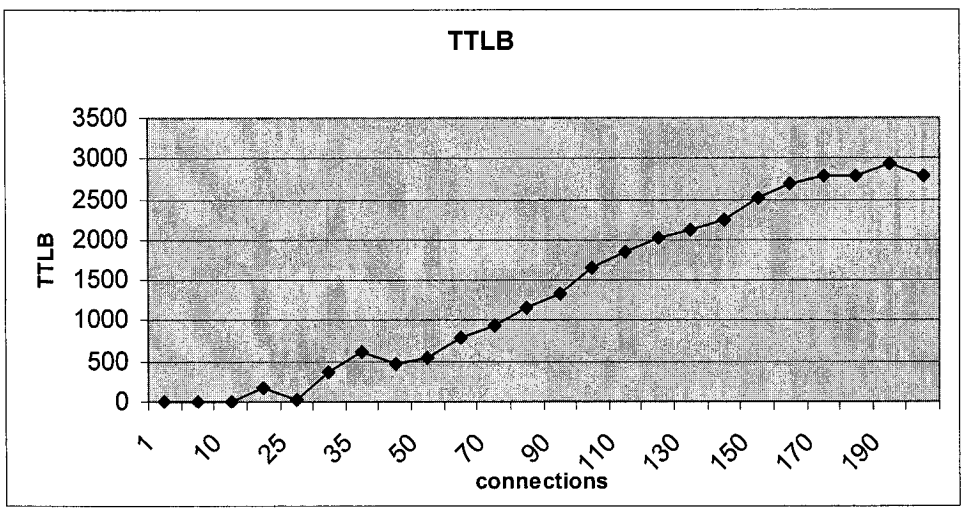

Figure 7-9 TTLB Results

7.3.5 Testing Summary

In summary, we obtain information about the performance of our application and know where to tune the system to reach the high performance we expected. As we discussed above, several factors, including mainly the bandwidth of the network, capacity of the server and efficiency of the database access, impact the performance of the application.

However, there are also several aspects we ignored in the test, which may affect the performance.

In our scenario, we test the application without setting delays in order to get the maximum stress. In a real environment, there must some delays between requests since it will take users time to look around the page and click to get the next page, no matter how familiar they are with the Web. Thus the application is able to serve a larger number of requests if the request delay is taken into account.

In addition, if we consider the scenario where most of the clients use modems, unlike LAN, which we use, more requests could be handled.

The configuration setting of the server is another key point that highly affects the performance of the application. The CPU of the server is pretty fast, but 512M of RAM on the server is not enough, especially for a heavy application. In general, 1G or even 2G of RAM is recommended for large applications. But the limited resources could be most taken advantage of by doing some performance tuning. It is certain that a larger number of simultaneous connections could be supported and the response time could be better with some performance tuning, such as adjusting the number of processors, including minprocessors and maxprocessors [73] in Tomcat.

The default garbage collection parameters were designed to be effective for most small applications, but they are not optimal for many server applications [74]. So it should be noted that tuning the garbage collection parameters properly would also improve the performance.

Chapter 8 Conclusion

8.1 Thesis Summary

In this paper, a brief introduction for the previous Click&Dial system is given, and its shortcomings in design structure are listed. It becomes increasingly important to integrate speech activity into the existing web applications as the explosive growth of the Internet technology. The overviews of VoiceXML, SALT and XHTML plus Voice (X+V) are presented for supporting voice-enabled web applications (Speech Recognition, TTS, Dialog control, etc.). Those technologies play key roles in developing flexible and extensible standards-based architectures. According to the comparison chart of their advantages and limitations, SALT is adopted because it is able to be seamlessly embedded into Web browser side with HTML, XHTML, XML or other standards to implement the speech interfaces. The SALT-based architecture of voice-activated web applications is presented in this paper. The voice interface is integrated, and the architecture of the web application is modified to meet the requirement. The architecture of this speech application plays an important role in scaling the system to meet increased demand and provide readily available service. The industry standard language Unified Modeling Language (UML) is used to visualize and specify the architecture of the web application. Moreover, we present the design of SALT-based system in detail, especially for voice interfaces, including navigation design, mode design, grammar design, and so on, which enables rapid development. Furthermore, the scalability, functionality and performance tests measure how well the web application performs under different scenarios with high load levels. Through these means, we verify that SALT fulfills the requirements perfectly, and the performance of SALT-based web applications is acceptable under most scenarios.

Although only part of the Click&Dial system is designed and implemented for voice activity, more functions can be added, and more mature and complicated voice-enabled web applications can be designed and implemented, based on the same idea or approach of this system. In addition, the test methodology used in this system has proved to be perfect for measuring system performance, which can be applied to other applications.

In sum, more and more voice-enabled web applications will be developed in the future. The standards of speech technology are still ongoing and improved greatly by the W3C Voice Browser Working Group. We believe providing efficient voice interfaces to web applications or services will become the main stream in the future.

8.2 Future Work

The World Wide Web (WWW) grows extremely fast every day. Major changes and improvements are likely to take place in the next few years. Currently, the application of the Click&Dial system is still not as mature as we expected. Some future improvements for voice-enabled web applications could be investigated in the following areas:

- More advanced frameworks could be used to replace the Struts like Java Server Faces. They will have more features and make development easier.
- In the application, the pein number and password are used to verify personal identification, but they are not secure enough for some scenarios. Adding another layer of verification will be the future work. Combining a voiceprint with a user ID and password is the proposed solution, since a voiceprint is a set of measurable characteristics of a human voice that uniquely identifies an individual.
- We propose taking advantage of the pre-record audio files instead of the voice generated through Text-To-Speech (TTS) synthesizer, to make the application more natural and friendly to users.
- So far, this application is developed to use at Bell. In the future, we propose modifying the current version to be employed in more companies rather than only Bell.
- The scalability, functionality and performance should be evaluated in a wider scope, not only in the lab environment, but also in the real practical situation.
- Currently, English is the only language for users may use to operate the application. However, the speech engine provided by Microsoft supports multiple languages, including English, French, and Mandarin. More flexible versions with more language choices are proposed to meet each individual's needs in multi-language markets.

- In the application, it is assumed there are no more than 10 results obtained for each search to simplify the process of results display. In some cases, such as in a big company, it is most likely that more than 10 results could be received, especially when searching by name. We propose using the selectable navigator to display the results by adding the previous and next buttons for turning the pages.
- Adding interruptions is proposed to make the application more flexible so that the user may interrupt the prompt when it is playing back.
- To date, the user has to type the URL of the web application into Internet Explorer to view the first page. The ideal design is to allow the user to enter the URL either by typing or by speaking.

In sum, with more effort, it is believed that the application could be greatly improved, and more efficient, friendlier voice user interfaces for web applications could be created.

Reference

[1]. Irina Ceaparu: Telephone Based Access to the Web – Speech Recognition Department of Computer Science, University of Maryland, College Park, MD 20742, USA, April 2001

[2]. The Next Net Craze: Wireless Access
http://www.businessweek.com/bwdaily/dnflash/feb1999/nf90211g.htm

[3]. Applications: Speech Enabled Telephony
http://www.brooktrout.com/applications/speech_enabled_telephony/

[4]. VoiceXML Forum releases preliminary specification, August 25, 1999
http://www.lucent.com/press/0899/990825.bla.html

[5]. Speech Application Language Tag (SALT) Forum
http://www.saltforum.org/

[6]. Sun Microsystems' Java Development Kit (JDK)
www.java.sun.com

[7]. Unified Modeling Language (UML)
www.uml.org

[8]. Richard Sproat, "Multilingual Test-to-Speech Synthesis: The Bell Labs Approach", by Lucent Technologies, 1998

[9]. Eric Keller, "Fundamentals of Speech Synthesis and Speech Recognition", by John Wiley & Sons Ltd, 1994

[10]. Thierry Dutoit, "An Introduction to Text-to-Speech Synthesis", Kluwer Academic Publishers, 1997

[11]. Text-to-Speech Synthesis: An Overview
www.telecom.tuc.gr/nlpcourse/resources/lecture_material/tts_overview.ppt

[12]. Byeongchang Kim, Gary Geunbae Lee, Jong-Hyeok Lee, "ACM transactions on Asian Language Information Processing(TALIP)", ACM Press, New York, 2002

[13]. Speech Synthesis, from Wikipedia
en.wikipedia.org/wiki/Speech_synthesis

[14]. Text-To-Speech Overview
http://www.microsoft.com/windowsxp/home/using/productdoc/en/default.asp?url=/windowsxp/home/using/productdoc/en/speech_tts_overview.asp

[15]. Shinta Kimura, "Advances in Speech Recognition Technologies", July9, 1999 FUJITSU Sci. Tech. J.,35, 2,(December 1999)
http://magazine.fujitsu.com/us/vol35-2/paper09.pdf

[16]. Olivier Deroo, "A Short Introduction to Speech Recognition"
http://www.babeltech.com/download/SpeechRecoIntro.pdf

[17]. K. Agaram, S. Keckler, and D. Burger. "A Characterization of Speech Recognition on Modern Computer Systems", in 4th IEEE Workshop on Workload Characterization, December 2001
http://citeseer.nj.nec.com/agaram01characterization.html

[18]. Rob Kassel, Senior Product manager, "How Speech Recognition Works", white paper, ScanSoft Inc.
http://216.162.203.249/downloads/How_Speech_Works_Article.doc

[19]. Lawrence Rabiner, Biing_Hwang Juang, "Fundamentals of Speech Recognition", AT&T, 1993

[20]. Marcus Fransoon, Marie Akerstrom, Paul Pylkas, "Speech Recognition", Lulea university of Technology, Department of Computer Science and Electrical Engineering Division of Software Engineering Multimedia systems, 30 October, 2001
http://www.cdt.luth.se/~peppar/kurs/smd074/seminars/3/1/2/SpeechRecognition.pdf

[21]. Cole, R., Cole, R., & Zue,V., "Spoken Language Input", Oregon Graduate Institute & Massachusetts Institute of Technology, 2001, in book "Survey of the State of the Art of Human Language Technology", chapter 1
http://cslu.cse.ogi.edu/HLTsurvey/ch1node3.html#SECTION11

[22]. Rehabilitation Engineering Research Center, "Applications of Automatic Speech Recognition for people with hearing loss", Washington
http://www.hearingresearch.org/asr.htm

[23]. Philip Felber, Illinois Institute of Technology, "Speech Recognition—Report of and Isolated Word experiment", April 25, 2001
http://www.ece.iit.edu/~pfelber/speechrecognition/report.pdf

[24]. Gerard Chollet, Maria Gabriella Di Benedetto, Anna Esposito and Maria Marinaro(Eds), "Speech Processing, Recognition and Artificial Neural Networks", Springer-Verlag London Limited, 1999

[25]. Jean-Claude Junqua, Jean-Paul Haton, "Robustness in Automatic Speech Recognition Fundamentals and Applications", Kluwer Academic Publishers, 1996

[26]. X.D.Huang, Y.Ariki, M.A. Jack, "Hidden Markov Models for Speech Recognition", Edinburgh University Press, 1990

[27]. World Wide Web Consortium (W3C)
http://www.w3.org/

[28]. VoiceXML Forum
http://www.voicexml.org/

[29]. Telera - Committed to Standards-Based Service Creation, white paper, by Telera
http://whitepapers.zdnet.co.uk/0,39025945,60039808p-39000590q,00.htm

[30]. Introducing XHTML+Voice – IBM's proposal to the W3C on developing multimodal Uis, Senior Technical Staff Member, IBM Group, 19 Aug 2003
http://www-128.ibm.com/developerworks/library/wi-xvlanguage/

[31]. Two Technologies Vie for Recognition in Speech Market, by Neal Leavitt
http://www.leavcom.com/pdf/Speech.pdf

[32]. The Microsoft Speech Server
http://www.microsoft.com/speech/

[33]. Atul Shenoy, "A Software Framework for Out-of-Turn Interaction in a Multimodal Web Interface", by U.S. national ScienceFoundation grants IIS-0049075, IIS-0136182
http://arxiv.org/PS_cache/cs/pdf/0307/0307011.pdf

[34]. Waggener Edstrom, "Microsoft .net Speech Platform Making Speech Mainstream", White Paper, February 2002

[35]. Microsoft Speech Application SDK
http://www.microsoft.com/speech/default.mspx

[36]. Georg Niklfeld, Robert Finan, Michael Pucher, "Component-based multimodal dialog interfaces for mobile knowledge creation", Proc. of EACL/ACL'01 Workshop on Human Language Technology & Knowledge Management, Toulouse, 2001
http://acl.ldc.upenn.edu/W/W01/W01-1016.pdf

[37]. Tomcat
http://jakarta.apache.org/tomcat/

[38].Java Authentication and authorization Service, technical article
http://java.sun.com/products/jaas/index-10.html

[39]. Struts
http://struts.apache.org/

[40]. Architecture Guide, Adventure Builder Application Architecture, Early Access

http://java.sun.com/blueprints/code/adventure/1.0/docs/architecture.html

[41]. Greg Murray, Inderjeet singh, "Storing Session State on the Client", technical article
https://bpcatalog.dev.java.net/nonav/webtier/clientside-state/

[42]. Vijay Ramachandran, "Design Patterns for Building Flexible and Maintainable J2EE Applications", Web architecture, January 2002
http://java.sun.com/developer/technicalArticles/J2EE/despat/

[43]. Abhijit Belapurkar, Use continuations to develop complex Web applications –A programming paradigm to simplify MVC for the web, Web architecture, 21 December 2004
http://www-106.ibm.com/developerworks/library/j-contin.html

[44]. Inderjeet Singh, Beth Stearns, Mark Johnson, "Web-Tier Application Framework Design", Web architecture
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html

[45]. Jeff Moore, Model View Controller, technical article
http://wact.sourceforge.net/index.php/ModelViewController

[46]. Brian Kotect, "MVC design pattern brings about better organizations and code reuse", October 30, 2002
http://builder.com.com/5100-6386-1049862.html

[47]. Govind seshadri, "Understanding Java Server Pages Model2 Architecture", discussed in a JavaWorld-jGuru forum, December 9, 1999
http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

[48]. Inderjeet Singh, Beth Stearns, Mark Johnson, "J2EE Architecture Approaches", technical article
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html

[49]. Daniel Windle, L. Abreo, "Software Requirements Using the Unified Process: A Practical Approach", chapter13: Use Case Realization by Means of Sequence Diagrams, Pearson Education, 2002, ISBN: 0-13-096972-9

[50]. Component Diagram, Visual Paradigm, technical article
http://www.visual-paradigm.com/VPGallery/diagrams/Component.html

[51]. Eric Armstrong, Jennifer Ball, "Understanding Login Authentication", the J2EE1.4 Tutorial, 2005, technical article
http://java.sun.com/j2ee/1.4/docs/tutorial/doc/Security5.html

[52]. Michiel Toneman, "Using JAAS with Tomcat", 2004
http://www.kopz.org/public/documents/tomcat/jaasintomcat.html

[53]. Log4j, Logging services
http://logging.apache.org/log4j/docs/

[54]. Apache Ant
http://ant.apache.org/

[55]. Cascading Style Sheet Home Page
http://www.w3.org/Style/CSS/

[56]. Eclipse
http://www.eclipse.org/

[57]. CVS
http://fox.wikis.com/wc.dll?Wiki~CVS~SoftwareEng

[58]. Milos, "ECots software component open directory project"
http://www.ecots.org/display.jsp?id=c_30398&cids=c_3499

[59]. Speech Applicastion Language Tags(SALT) 1.0 Specification, 15 July 2002
http://www.saltforum.org/devforum/spec/SALT.1.0.a.asp

[60]. Speech Application Language Tags(SALT) Technical White Paper
http://www.saltforum.org/whitepapers/whitepapers.asp

[61]. Kathy Frostad, "Best Practices in Designing Speech User Interfaces", Voice Web Consulting, July 2003
http://www.microsoft.com/speech/docs/VUIBestPracticesART.htm

[62]. Kuanshan Wang, "Salt: A Spoken Language Interface for Web-Based Multimodal Dialog System", Speech Technology Group, Microsoft Research, One Microsoft Way, RedMond, WA, USA

[63]. Speech Recognition Grammar Specification Version 1.0, technical article
http://www.w3.org/TR/speech-grammar/

[64]. Andrew Hunt, "Introduction to the W3C Grammar Format", Voice-XML forum
http://www.voicexmlreview.org/Apr2001/features/w3c-grammar1.html

[65]. Ye-Yi Wang and Alex Acero, "Combination of CFG and N-gram Modeling in Semantic Grammar Learning", Speech Technology Group, Microsoft Research, One Microsoft Way, RedMond, WA, USA

[66]. Paul Osburn, Matt Hempey, Chris Idzerda, "Design and Implement a Voice-only Web Application in ASP.NET", whitepaper April 23, 2003
http://www.devsearch.com/dotnet/Article/11958/0/page/7

[67]. Arny Epstein, "Jave Servlet 2.2 Introduces the Web Application",
SYS-CON Media, Inc., 2004

[68]. Functionality Testing Process, technical article
http://www.keylabs.com/services/hardware-software-functionality-testing.html

[69]. Rick Strahl, "Load Testing Web Applications using Microsoft's Web Application Stress Tool", Feb. 24, 2000
http://www.west-wind.com/presentations/webstress/webstress.htm

[70]. Henrik Soderquist, "An Application-Independent Speaker Adaptation Service", Department for Speech, Music and Hearing, KTH, Stockholm, 26 April 2002

[71]. Khalid Anwar and Arif Saleem, "Web Application Stress Test and Data Analysis", April 15, 2002,
http://www.webstar.co.uk/Stress.pdf

[72]. Web Application Stress Tool
http://www.microsoft.com/downloads/details.aspx?FamilyID=E2C0585A-062A-439E-A67D-75A89AA36495&displaylang=en

[73]. Rahul Kuchhal, "J2EE Application Performance Optimization", technical article
http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html

[74]. "Tuning Garbage Collection with the 1.3.1 Java Virtual Machine", performance documentation, Sun Microsystems. Inc.
http://java.sun.com/docs/hotspot/gc/

[75]. "Architectural and Cluster Terminology", technical article
http://e-docs.bea.com/wls/docs81/cluster/planning.html

[76]YuFeng, Network Programming with J2ME Wireless Devices
http://www.wirelessdevnet.com/channels/java/features/j2me_http.phtml

[77] Introduction to Mobility Java Technology, Sun develop network, technical topics
http://developers.sun.com/techtopics/mobility/getstart/

[78] J2ME, small business computing channel
http://www.webopedia.com/TERM/J/J2ME.html

[79] Vikram Goyal, J2ME Tutorial, February 09, 2005
http://today.java.net/pub/a/today/2005/02/09/j2me1.html

Appendix: System Requirements

Minimal System Requirements

Client workstation	
Processor	Pentium III-class, 600 MHz
RAM	128 MB Recommended: 256 MB
Operating System	Windows 2000 Service Pack 3
CD-ROM	Required for installation from CD.
Hard Disk Space	225 MB on the system partition
Microphone	Required
Audio Output Device	Required
Server	
Processor	Pentium IV, 1.8GHz
RAM	512 MB
Operating System	Windows 2000 Service Pack 3
CD-ROM	Required for installation from CD.
Hard Disk Space	10GB

Software Requirements for Development Environment

Category	**Details**
Application server	Apache Tomcat 5.24
Develop tool	1. Eclipse 3.0 2. Microsoft Visual Studio .NET 2003 Professional Edition
LDAP	OpenLDAP 2.09
Build tool	Ant 1.6
Version Control	CVS in Linux

Microsoft Speech Application SDK	1. Microsoft Enterprise Instrumentation 2. Internet Information Services 3. Microsoft Internet Explorer, Version 6.0, Service Pack 1 or later 4. Microsoft .NET Framework 1.1
IE plug in	Microsoft Speech plug in

Appendix: Running the Tomcat Servlet/JSP Container

(1). Download and Install a Java Development Kit (If you don't have JDK)

Set the environment JAVA_HOME

(2). Download and Install the Tomcat 4.0 Binary Distribution

* Download a binary distribution of Tomcat from:

http://jakarta.apache.org/builds/jakarta-tomcat-4.0/nightly/

On a Windows platform, we need:

jakarta-tomcat-4.0-YYYYMMDD.zip

On a Unix platform, we need:

jakarta-tomcat-4.0-YYYYMMDD.zip

* Unpack the binary distribution into a convenient location so that the distribution resides in its own directory (conventionally named "jakarta-tomcat-4.0"). For the purposes of the remainder of this document, the symbolic name "${catalina.home}" is used to refer to the full pathname of the release directory.

(3). Start Up Tomcat 4.0

There are two techniques by which Tomcat 4.0 can be started:

* Via an environment variable:
  - Set an environment variable CATALINA_HOME to the path of the directory into which you have installed Tomcat 4.0.
  - Execute the shell command:

%CATALINA_HOME%\bin\startup	(Windows)
$CATALINA_HOME/bin/startup.sh	(Unix)

* By modifying your current working directory:
  - Execute the following shell commands:

```
cd %CATALINA_HOME%\bin          (Windows)
startup                         (Windows)
cd $CATALINA_HOME/bin           (Unix)
./startup.sh                    (Unix)
```

After startup, the default web applications included with Tomcat 4.0 will be available by browsing:

http://localhost:8080/

Appendix: J2EE Design Pattern

Pattern Name	Description	Use Scenario
Composite View	This pattern is a view using template to manage common page elements. i.e., it uses a layout with sub-views, including a header, footer, left menu, body, and so on [48].	Tile, which uses this pattern, is adopted in our system to divide the page to three parts, including header, footer, and body.
Façade	This pattern coordinates operations between cooperating business objects, unifying application functions into a single, simplified interface for presentation to the calling code. It encapsulates and hides the complexity of classes that must cooperate in specific, possibly complex ways, and isolates its callers from business object implementation changes [48].	There is a SearchMgr class playing the role as the façade.
Front Controller	This pattern provides a centralized controller for managing requests. A front controller receives all incoming client requests, forwards each request to an appropriate request handler, and presents an appropriate response to the client. Point of entry for requests to update model. The controller is responsible for processing requests [48].	Struts' ActionServlet works in the same way described here.
Service Locator	Locate a service or resource needed within an application such as a JNDI resource, JDBC connection, or an EJB component [48].	It's used to retrieve the LDAP server, url, etc., information
View Dispatcher	Generally part of the controller, the view dispatcher is responsible for dispatching a request to a specified view based on the results of the Command processing and/or current state of the model [48].	Struts framework handles this function.
View Helper	A view helper encapsulates the presentation and data access logic portions of a view, thus refining the view and keeping it simpler. Presentation logic concerns formatting data for display on a page, while data access logic involves retrieving data. View helpers are often JSP tags for rendering or representing data and JavaBeans for retrieving data [48].	Struts tag library supports this pattern.
Value	This pattern facilitates data exchange between tiers (usually the	In the code, Person

Object	Web and EJB tiers) by reducing the cost of distributed communication. In one remote call, a single value object can be used to retrieve a set of related data, which then is available locally to the client [48].	JavaBean is used as the Value Object to transfer data between the tiers.

Appendix: Validator

To implement and config the Validation using Struts, the basic steps are listed as follows:

1. The Form Bean must extend ValidatorForm.
2. The <html:errors/> tag must be included on the form's JSP where to display the error message.
3. The validation rules must be defined in Validation.xml file as follows:

```
<form-validation>
    <formset>
      <form name="searchForm">
        <field property="userName" depends="required">
          <msg name="required" key="searchForm.userName"/>
        </field>
      </form>
    </formset>
 </form-validation>
```

The msg element points to the message resource key to use when generating the error message.

4. The format of error message displaying is defined in ApplicationResources.properties file.
5. Lastly, you must enable the ValidatorPlugin in the struts-config.xml file like this:

```
<plug-in
    className="org.apache.struts.validator.ValidatorPlugIn">
        <set-property property="pathnames"
                value="/WEB-INF/validation.xml"/>
</plug-in>
```

Appendix: Tiles

To implement and config the Tiles using Struts, the basic steps are listed as follows:

1. Create a layout.jsp file that contains application's common look and feel:

```
<html>
   <body>
      <table>
        <tr><td><tiles:insert attribute="header"/></td></tr>
        <tr><td><tiles:insert attribute="body"/></td></tr>
        <tr><td><tiles:insert attribute="footer"/></td></tr>
      </table>
   </body>
</html>
```

2. Create JSP files:

Since we set three attributes in the layout file, we need to create three JSP files, which are head.jsp, search_body.jsp and footer.jsp.

3. Define the "Tile" in the tiles-defs.xml config file that looks like this:

```
<tiles-definitions>
   <definition name="Search" path="/layout.jsp " >
      <put name="header" value="/common/header.jsp"/>
      <put name="body"   value="/search_body.jsp "/>
      <put name="footer" value="/common/footer.jsp"/>
   </definition>
<tiles-definitions>
```

4. Setup the TilesPlugin in the struts-config.xml file:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin">
        <set-property
            property="definitions-config"
            value="/WEB-INF/tiles-defs.xml"/>
</plug-in>
```

5. Set up an action mapping in struts-config.xml to point to the search tile:

```
<action path="/Search"
        type="com.bell.corpdir.actions.SearchAction">
        <forward name="success"                path="Search"/>
</action>
```

In struts-config,xml, we define the TilesPlugin, which indicates the special RequestProcessor. The corresponding head, body and footer JSP files will be inserted into the main layout [39]. Therefore the common tiles, such as header and footer, can be reused.

Appendix: Security

To config the security using JAAS, the basic steps are [52]:

1. In order to config JAAS security, the security realm has to be defined in the server.xml. There are different kinds of realms, such as the database realm, LDAP realm, etc. As decided before, the database will be used to store the username and passwords. Then the database security realm is chosen.
    - First, the database should be populated with user data. The following tables should be created:

User table:

Column name	Column type	Size	Constraints
User_name	varch	15	Primary key, not null
User_pass	varch	15	Not null

User role table:

Column name	Column type	Size	Constraints
User_name	varch	15	not null
Role_name	varch	15	Not null

primary key for user_name, role_name

- Example Realm elements are included (commented out) in the default $CATALINA_HOME/conf/server.xml file. Here's an example for using a MySQL database called "authority", configured with the tables described above, and accessed with username "dbuser" and password "dbpass":

```
<Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
   driverName="org.gjt.mm.mysql.Driver"
connectionURL="jdbc:mysql://localhost/authority?user=dbuser&amp;password=dbpass
"
    userTable="users" userNameCol="user_name" userCredCol="user_pass"
  userRoleTable="user_roles" roleNameCol="role_name"/>
```

2. Change the web.xml

- Add the restriction for URLs

```
<!-- Security is active on entire directory -->
 <security-constraint>
  <display-name>Click&Dial Configuration Security Constraint</display-name>
  <web-resource-collection>
        <web-resource-name>Protected Area</web-resource-name>
        <!-- Define the context-relative URL(s) to be protected -->
        <url-pattern>/protected/*.jsp</url-pattern>
        <url-pattern>/protected/*.do</url-pattern>
  </web-resource-collection>
  <auth-constraint>
        <!-- Anyone with one of the listed roles may access this area -->
       <role-name>admin</role-name>
       <role-name>user</role-name>
  </auth-constraint>
 </security-constraint>
```

- Add the login part

```
<!-- Login configuration uses form-based authentication -->
 <login-config>
        <auth-method>FORM</auth-method>
        <realm-name> Click&Dial Configuration Form-Based
                                Authentication Area</realm-name>
        <form-login-config>
              <form-login-page>/logon.jsp</form-login-page>
              <form-error-page>/error.jsp</form-error-page>
        </form-login-config>
 </login-config>
```

- Add the roles

```
<!-- Security roles referenced by this web application -->
 <security-role>
        <description>The role is required to log in to
                                        the administration pages</description>
        <role-name>admin</role-name>
 </security-role>
 <security-role>
        <description>The role that is required to log
                                        into the application</description>
        <role-name>user</role-name>
 </security-role>
```

Appendix: Log4j

Here is an example about how to write codes to use the log4j and configurations [65]:

1. In the java source code:

```
import org.apache.log4j.Logger;

public class Bar {
        static Logger logger = Logger.getLogger(Bar.class);
        public void doIt() {
                logger.debug("Error!");
        }
.....
}
```

2. In configuration file

```
log4j.rootLogger=debug, stdout, ClickDial

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) -
%m%n

log4j.appender.ClickDial=org.apache.log4j.RollingFileAppender
log4j.appender.ClickDial.File=clickdial.log

log4j.appender.ClickDial.MaxFileSize=100KB
# Keep one backup file
log4j.appender.ClickDial.MaxBackupIndex=1

log4j.appender.ClickDial.layout=org.apache.log4j.PatternLayout
log4j.appender.ClickDial.layout.ConversionPattern=%p %t %c - %m%n
```

Then, if it's run, the output log will look like this:

```
INFO [main] (Bar.java:12) - Entering application.
DEBUG [main] (Bar.java:8) - Hello world!
```

INFO [main] (**Bar**.java:15) - Exiting application.

Appendix: ANT

Here is an example of an ANT scripts file [54]:

```
<project name="MyProject" default="dist" basedir=".">
  <description>Click&Dial build file</description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>      <!--define the source directory-->
  <property name="build" location="build"/><!--define the build destination directory-
->
  <property name="dist"  location="dist"/>   <!--define the distribution directory-->

  <!--define the initialization target -->
  <target name="init">
   <!-- Create the time stamp -->
   <tstamp/>
   <!-- Create the build directory structure used by compile -->
   <mkdir dir="${build}"/>
  </target>
  <!--define the compile source code target-->
  <target name="compile" depends="init" description="compile the source " >
   <!-- Compile the java code from ${src} into ${build} -->
   <javac srcdir="${src}" destdir="${build}"/>
  </target>
  <!--define the build war file target-->
  <target name="dist" depends="compile"
      description="generate the distribution" >
```

```
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>
    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/ClickDial-${DSTAMP}.jar" basedir="${build}"/>
  </target>
  <!--define the clean up target: delete useless files and directory-->
  <target name="clean" description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```