Université de Technologie de Compiègne – UTC
HEUDIASYC – Heuristique et Diagnostic des Systèmes Complexes

Voice Activated Information Entry: Technical Aspects

Author: Emerson Cabrera Paraiso
Email: emerson.paraiso@hds.utc.fr

Compiègne – 2003

Contents

1. Introduction 3
2. Automatic Speech Recognition (ASR) 4
   2.1. Accuracy 4
   2.2. Signal Processing 4
   2.3. ASR Engine 4
   2.4. Applications that uses ASR 5
   2.5. Engine Adjustment for Windows Operating System Users 5
   2.6. Speaker Dependency 5
3. Text To Speech Generation 6
4. Others Terms and Concepts 7
   4.1. Utterances 7
   4.2. Grammars 7
   4.3. Speech Recognition and the Telephony 7
   4.4. VoiceXML 8
5. Developing a Speech Application 9
   5.1. Desktop or WEB Application Interface Design 9
      5.1.1. Command and Control Interface Design 10
      5.1.2. Dictation Interface Design 10
   5.2. Telephone Application Interface Design 11
      5.2.1. Command and Control Interface Design 11
      5.2.2. Dictation Interface Design 11
   5.3. Using Voice Tools 12
      5.3.1. Voice Server 12
      5.3.2. Lisp Client Voice Application 13
6. From Voice Oriented Interfaces to Conversational Interfaces 14
   6.1. Personal Agent User Interface Entry Point 15
   6.2. Dialogue Samples 15
   6.3. User Interface and Assistance Basic Structure 15

1. Introduction

This text explores the problem on how to enable voice oriented interfaces giving an overview on Speech Recognition programming. Its aim is to technically describe the use of Speech Recognition.

This document contains personal impressions and definitions about the subjects presented. The orientation of this text is intentionally driven to my work at Université de Techonologie de Compiègne (UTC), which addresses the construction of software agents with voice capabilities.

The use of speech in software development is not new. Many applications already have voice capabilities. Simple inquiries about bank balance, movie schedules, and phone call transfers can already be handled by telephone-speech recognizers.

Voice activated data entry is particularly useful in many situations, such as medical applications, where hands and eyes are unavailable, or in hands-busy or eyes-busy command and control applications. Speech is used to provide more accessibility for the handicap (wheelchairs, robotic aids, etc.) and to create high-tech amenities (intelligent houses, cars, etc.), as well. By adding voice as an input option, one can afford a 'multi-modal' computer environment, freeing users from dependence on the mouse and keyboard.

Recently, over-the-phone applications, i.e., applications that use the telephone as logical media, has spread over several areas of business, such as: transportation, financial, telecommunications, and others. Most people have by now experienced at least one automated phone system, where questions are posed by a computer to callers seeking help or information (Kotelly 03).

Along this text, we intend to show how these applications work; what designers should have to do applications with voice capabilities. Technical aspects are presented such as the use of Automatic Speech Recognition engines and ActiveX/COM components for programming voice interfaces. Figure 1 gives a notion of how a desktop voice program looks like.

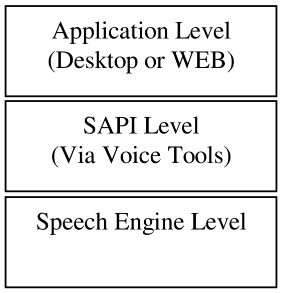

Figure 1. Three level components for desktop application.

In the next sections we are going to study each level, from the lowest to the highest. By the end, we discuss the problems involving these kind of interfaces and what we think will be the future interfaces.

2. Automatic Speech Recognition (ASR)

Automatic Speech Recognition (ASR) is a technology that allows a computer to identify the words that a person speaks into a microphone or a telephone. The goal of ASR research is to allow a computer to recognize with 100% accuracy all the words that are spoken by any person, independent of vocabulary size, noise, speaker characteristics and accent, or channel conditions (CSLU 03).

2.1. Accuracy

Despite several decades of research in this area, accuracy greater than 90% is only attained when the task is constrained in some way. Depending on how the task is constrained, different levels of performance can be attained; for example, recognition of continuous digits over a microphone channel (small vocabulary, no noise) can be greater than 99%. If the system is trained to learn an individual speaker's voice, then much larger vocabularies are possible, although accuracy drops to somewhere between 90% and 95% for commercially-available systems. For large-vocabulary speech recognition of different speakers over different channels, accuracy is no greater than 87%, and processing can take hundreds of times real-time (CSLU 03).

2.2. Signal Processing

The dominant technology used in ASR is called the Hidden Markov Model (HMM). This technology recognizes speech by estimating the likelihood of each phoneme at contiguous, small regions (frames) of the speech signal. Each word in a vocabulary list is specified in terms of its component phonemes. A search procedure, called a Viterbi search, is used to determine the sequence of phonemes with the highest likelihood. This search is constrained to only look for phoneme sequences that correspond to words in the vocabulary list, and the phoneme sequence with the highest total likelihood is identified with the word that was spoken. In standard HMMs, the likelihoods are computed using a Gaussian Mixture Model; in the HMM/ANN framework, these values are computed using an artificial neural network (ANN) (CSLU 03).

2.3. ASR Engine

The process of speech recognition is encapsulated in a software component known as the speech recognition engine. The main function of the speech recognition engine is to process spoken input and to translate it into text that an application may understand.

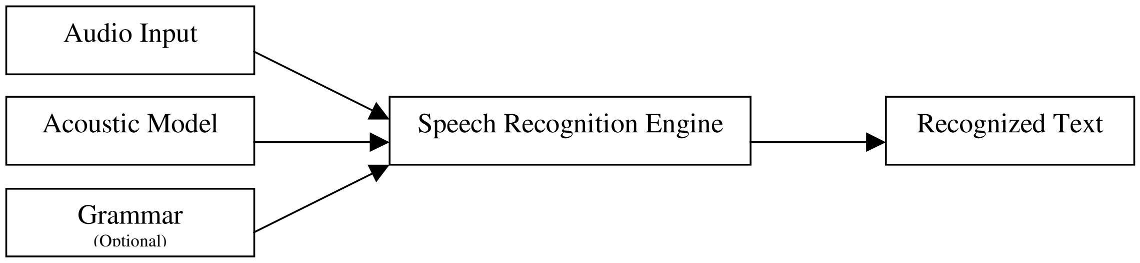

Figure 2. Speech recognition engine configuration.

2.4. Applications that uses ASR

There are two types of applications that use speech recognition:

1) Command and Control Applications: the application interprets the result of the recognition as a command. In this case, the application is a command and control application. An example of a command and control application is one in which the caller says “check balance”, and the application returns the current balance of the caller’s account;

2) Dictation Applications: if an application handles the recognized text simply as text, then it is considered a dictation application. In a dictation application, if the user says “check balance,” the application would not interpret the result, but simply return the text “check balance” and use it in a more complex context, for instance, to feeding a dialogue session.

2.5. Engine Adjustment for Windows Operating System Users

After installing the engine, the user may perform some adjustments (Microsoft 03).

Pronunciation Sensitivity: controls the level of confidence needed for the system to respond to the user command. High sensitivity indicates that the user wants the system to reject any command he utters when is not confident of what was said. This means the system will make fewer recognition errors, but it will also increase the frequency of rejecting what was said. As a result, the user may need to enunciate more slowly and clearly. Low sensitivity indicates that the system will respond to the user command when it has little confidence that it has correctly recognized what was said. This will result in the system making more mistakes in recognizing user commands, but it will very rarely reject a command. This setting affects only command and control applications.

Accuracy vs. Recognition Response Time: controls the trade off between accuracy in recognizing speech and computer processing time needed to generate the recognized speech. Low/Fast indicates that the system will perform limited processing. This results in the recognized text appearing on the screen quickly, but with a low accuracy rate. High/Slow indicates that the system will perform a much larger amount of computation to obtain the best accuracy possible, but at the expense of producing the dictated text much more slowly. This setting affects both command and control applications and dictation applications.

Background Adaptation: controls whether the system adapts to the user voice and speaking manner. When this feature is checked, the system will learn what the user’s voice sounds like (in a particular environment). This information is stored in the user profile and contributes to an improvement in recognition accuracy. It is highly recommended that the user selects this setting.

2.6. Speaker Dependency

Some recognizer engines are speaker dependent, which means they require the user to train the recognizer for his specific voice. Voice desktop applications are usually created for a single user with a much larger vocabulary. In this case voice training is employed to improve accuracy.

Others engines are speaker independent, so they can be used by anyone, without any training by the user. These type of engines are indicated for large-scale or telephony-based systems.

3. Text To Speech Generation

Text to speech (TTS) is the generation or production of synthesized speech from text. This involves breaking down the words into phonemes; analyzing for special handling of text such as numbers, currency amounts, inflection, and punctuation; and generating the digital audio for playback.

The concept of TTS engine is a result of developments in speech synthesis and natural language processing techniques.

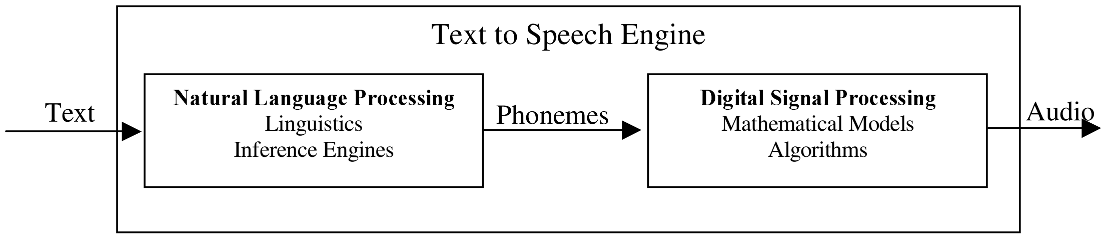

Figure 3. TTS system general diagram.

Figure 3 introduces the functional diagram of a general TTS engine or synthesizer. It comprises a Natural Language Processing module (NLP), capable of producing a phonetic transcription of the text, together with the desired intonation and rhythm (often termed as prosody), and a Digital Signal Processing module (DSP), which transforms the symbolic information it receives into speech (Dutoit 96).

There are two main types of TTS engines (Kotelly 03):

- Formant TTS: those that synthesize the sound, and;
- Concatenative TTS: those that take thousands of small pieces of prerecorded human-speech and concatenate them. These ones are more appropriate since the result is quite near the voice of the person from whom the audio files were recorded.

Many systems may use what is known by *prompts*, which are pre-recorded sentences, such as acknowledgments sentences or error indication sentences.

4. Others Terms and Concepts

Following are some of the basic terms and concepts that are fundamental to speech recognition.

4.1. Utterances

When the user says something, this is known as an utterance. An utterance is any stream of speech between two periods of silence. Utterances are sent to the speech engine to be processed (Kemble 01).

Silence, in speech recognition, is almost as important as what is spoken, because silence delineates the start and end of an utterance.

The speech recognition engine is "listening" for speech input. When the engine detects audio input, in other words, a lack of silence, the beginning of an utterance is signaled. Similarly, when the engine detects a certain amount of silence following the audio, the end of the utterance occurs (Kemble 01).

An utterance can be a single word, or it can contain multiple words (a phrase or a sentence). For example, “checking”, “checking account,” or “I’d like to know the balance of my checking account please” are all examples of possible utterances - things that a caller might say to a banking application.

4.2. Grammars

A grammar uses a particular syntax, or set of rules, to define the words and phrases that can be recognized by the engine. A grammar can be as simple as a list of words, or it can be flexible enough to allow such variability in what can be said that it approaches natural language capability.

Grammars define the domain, or context, within which the recognition engine works. The engine compares the current utterance against the words and phrases in the active grammars. If the user says something that is not in the grammar, the speech engine will not be able to decipher it correctly.

The design of grammars is important to improve accuracy. One challenge is to design grammars to satisfy a few speakers using a large vocabulary or many speakers using a limited vocabulary, or some combination of the two .

4.3. Speech Recognition and the Telephony

A hard infra-structure is required to allow a system to deliver information to a telephone service. This infra-structure is compounded by a Voice Server which is responsible to perform speech recognition and TTS generation; by a voice browser, which creates HTTP requests as necessary, and accesses the target information over the network. The voice browser is analogous to a visual browser, except that speech-enabled requests look for Web pages written in VoiceXML.

The application resides on a Web application server, which contains pages of both visual HTML and VoiceXML code. As each HTTP request is received, information is returned to the requesting server in the form of VoiceXML pages, which the TTS engine reads back to the caller (see figure 4).

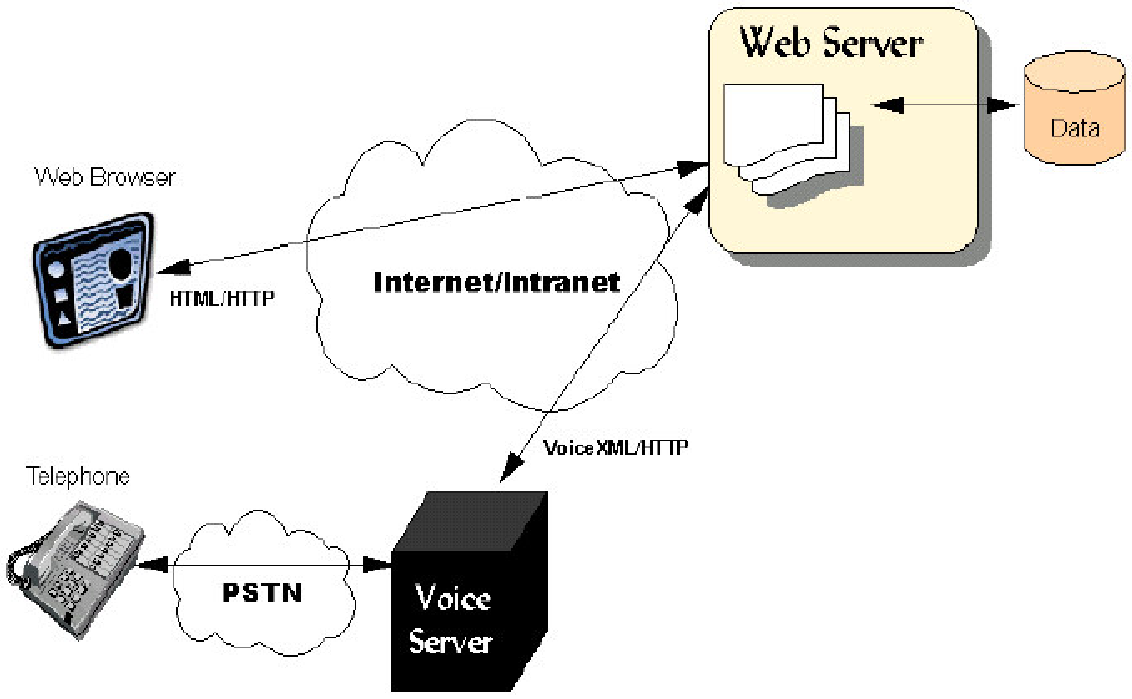

Figure 4. Voice WEB access overview (IBM 01).

4.4. VoiceXML

The Voice Extensible Markup Language or VoiceXML, is an XML-based markup language for distributed voice applications. VoiceXML is defined by an industry forum, the VoiceXML Forum, founded by AT&T, IBM, Lucent and Motorola. VoiceXML was designed to create audio dialogs that feature text-to-speech, digitized as well as prerecorded audio, recognition of both spoken and DTMF (Dual Tone Multi-Frequency) key input, recording of spoken input, telephony, and mixed-initiative conversations. Its goal is to provide voice access and interactive voice response (e.g. by telephone, PDA, or desktop) to Web-based content and applications.

5. Developing a Speech Application

This section describes the implementation of voice applications. For developing a program that is speech enabled for Windows operating system, the programmer will need to interface with the Speech Application Program Interface (SAPI) specially designed for this purpose. Its function is to provide access to the Speech Recognition and TTS engines. This API contains hundreds of functions and variables and is hard to use. To facilitate the engines manipulation, many packages are available, encapsulating and managing this access. Some of them are appropriate for a specific type of application, command/control applications or dictation applications. Voice Tools is one of them.

Voice Tools is a set of ActiveX/COM components developed by Wizzard Software. It eliminates the need for programmers to directly interface with the speech engine at the API level. Voice Tools are intended for use by Windows developers in programming languages that supports COM components. It can also be used in the web development environment using VB Script, Java Script or others scripting languages which support the use of ActiveX/COM components. There is no impact on ability to use the mouse or keyboard when adding speech to the application.

Figure 5 presents a diagram that illustrates the speech flow treatment through the system.

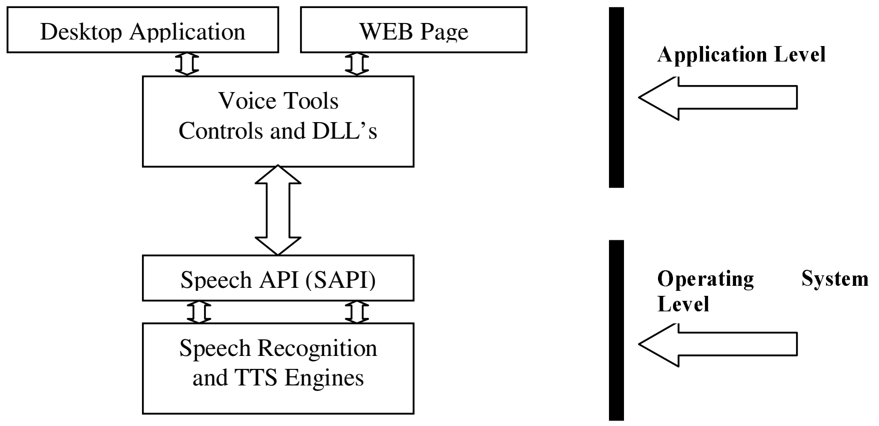

Figure 5. Speech Treatment flow diagram.

Voice Tools will be useful at interface design time. The use of speech may significantly change the user interface. The developer will design her application interface and also will determine the way speech will drive the information entry. Depending on the application style (command/control or dictation) the use of speech will directly interfere with interface design. Let us analyze this impact dividing the interface design into two groups: desktop or WEB application and telephone application interface design. It is important to highlight that Voice Tools is useful only for desktop applications.

5.1. Desktop or WEB Application Interface Design

To better understand let us take a Weather System as an example. In this system four main options are available: Select a City, Satellite Photos, Conversion and exit. The developer may choose to implement a command and control or a dictation speech interface.

5.1.1. Command and Control Interface Design

In this case, each application feature is represented by a command. The main window is shown on figure 6. Each option is represented by a button. Clicking on the button will start the required operation. Since the developer decided to develop a *command and control* speech interface, she will enable her interface to start the operation as soon as the user say, for instance, the text that corresponds to the button caption. If the utterance is recognized by the speech engine, the same operation will be performed as it would happen if the user had clicked the button.

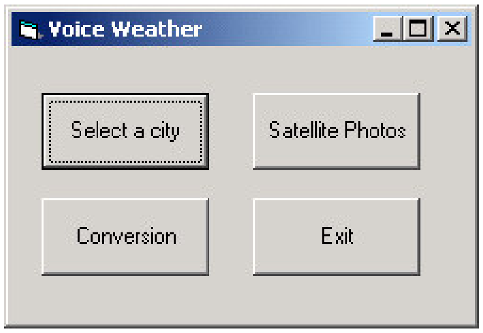

Figure 6. Banking application main interface window.

When the application is running, everything said is caught by the microphone and automatically transferred to the speech recognition engine. The system recognition starts when an utterance is recognized (an utterance may be a word or a group of words). The engine gives back the resulting text. The application must then analyze the text and start the required action, as is done in the extract of the algorithm shown below:

```
Select Case TextRecognizedByTheSpeechEngine
        Case "Select a city"
            Call Exec_SelectCity()
        Case "Satellite Photos"
            Call Exec_Satellite()
        Case "Conversion"
            Call Exec_Conversion()
        Case "Exit"
            Call Exit()
        Case Else
            TextToSpeech = "Invalid Command"
            Call Say_It(TextToSpeech)
End Select
```

This test is done when the speech engine returns something. If the text is not a valid command, then the user may be informed.

This type of speech use does not change anything in the interface, since the user may use the mouse and keyboard and the orientation is made by buttons. In this case the use of speech may be useful for handicapped persons.

5.1.2. Dictation Interface Design

The interface design is harder than the previous one and comprises the definition of a dialogue system. The result could be a conversation just like this one:

(1) SYS: Welcome to Voice Weather. What can I do for you?
(2) USR: Please, I would like to know the weather condition in Paris.
(3) SYS: OK. Today the weather will be nice and the actual temperature is 25 Celsius degrees.
(4) USR: And tomorrow?
(5) SYS: For tomorrow is expected a cloudy day. Do you want to see some Satellites photos?
(6) USR: Yes!
…

where: SYS is the system utterance and USR is the user utterance.

5.2. Telephone Application Interface Design

Telephone applications interfaces require a special design. Since there is no graphical window to implement, the designer will obligatorily implement a "dialogue like" solution.

5.2.1. Command and Control Interface Design

The dialogue system in this case could be a Finite-State or a Question-Answer one (Clark 99). This type of dialogue will drive the user allowing the system to collect the needed information. The result could be a system that produces a dialogue as follows:

(1) SYS: Welcome to Voice Bank. Please enter or say your account number.
(2) USR: Five, six, one, one, two, eight.
(3) SYS: Please, enter your PIN number?
(4) USR: The user will use the touchtone keypad to enter her PIN.
(5) SYS: You're at the main menu. You can say "Account balances" or "Transfer funds" or "Preferences"
…

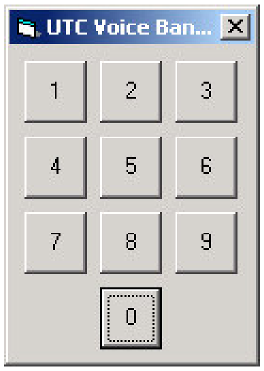

Figure 7. Command Control Voice Banking.

The figure 7 shows an application specially implemented to test this situation. In this kind of application the ability to generate audio is also important, since the user does not have a monitor or display to read what is happening or what she should do next.

5.2.2. Dictation Interface Design

This choice will allow user friendly interfaces, but will demand more effort. The user will be able to speak using natural language as she could do if she were at a bank desk talking to a bank employee. The interface design comprises the definition of a dialogue system; there is no menu options in this situation. The result could be a conversation just like this one:

(1) SYS: Welcome to Voice Bank. What you want to do?
(2) USR: I want to check my account balance.
(3) SYS: OK. Please enter or say your account number.
(4) USR: Five, six, one, one, two, eight.
(5) SYS: Please, enter your PIN number?
(6) USR: *The user will use the touchtone keypad to enter her PIN number.*
(7) SYS: Your account balance is …

5.3. Using Voice Tools

The actual version of Allegro CL does not support ActiveX/COM components directly. Thus, the use of Voice Tools components is impossible in an Allegro Lisp application. A Lisp interface set of functions was developed to allow a programmer to develop command/control applications. This set of functions must be attached to the Lisp project file. Also, a Voice Server application was developed and must be loaded with the Lisp application. The voice server will be responsible for performing the voice treatment and sending its results to the Lisp application.

5.3.1. Voice Server

The voice server is a Visual Basic application that receives the list of commands to be treated from the client. It stores the commands sent by the client and always resends an acknowledgment when a command is recognized. In its first version it is able to treat only button, menu and list commands. It is capable of receiving and generating text to speech, as well. The Voice Server may be used by any program, written in any language.

The voice server, shown on figure 8, must be started before the client application. It works independently of which application has the currently focus.

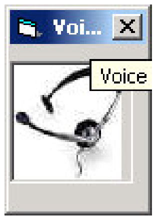

Figure 8. Voice Server

A very simple protocol was implemented to allow the client request register:

```
    <string> ::= ASCII
    <message> ::= <component type>&<caption>
    <component type> ::=  "button" | "menu" | "list" | "tosay" |
"text"
    <caption> ::= <string>
```

Thus, to register a new button called "Open File", the client application must to send the following message to the server:

"button&Open File"

The interaction between the client and the server is done using Sockets (UDP/IP connection) by now. The voice server uses port 9999 to receive messages.

5.3.2. Lisp Client Voice Application

A set of functions was developed to simplify the Voice Server use by a Lisp application. First the client sends a registry request calling `(init-voice)`. After, the client may register all components that will be voice enabled. It can do it by calling one of the functions:

```
(add-voice-to-button caption function-to-execute list-of-args)
```

where: `caption` is the button caption,
`function-to-execute` is the function name to be executed when the button caption were speech, and
`list-of-args` is its arguments.

```
(add-voice-to-menu caption function-to-execute list-of-args)
(add-voice-to-list caption function-to-execute list-of-args)
(say-it text-to-say)
```

To terminate the Voice Server utilization, the client sends a closing solicitation by calling `(finish-voice)`.

See bellow how to register a client and configure a button and a list of elements to be voice enabled.

```
(init-voice) ;; at the begging

(add-voice-to-button "choice" 'exec-choice '(test))
(add-voice-to-button "new user" 'exec-user nil)
(add-voice-to-list "file" 'exec-file nil)
(add-voice-to-list "color" 'exec-color nil)
(say-it "Welcome to the Voice UTC Phone Banking")

(finish-voice) ;; at the end
```

In the example above, if the user says “choice”, the lisp function `exec-choice` will be executed. If the client needs a voice message, it could do it by calling `say-it` passing the text to be speech.

6. From Voice Oriented Interfaces to Conversational Interfaces

Our interest is spoken dialog handling in a personal assistance context. The use of speech is a central point. But, it is important to realize that a speech interface by itself does not solve the problem. To replace the operations of menu selection by using speaking predetermined phrases that perform the equivalent operation, may aggravate the problem, since the user would need to remember a potentially long list of arbitrary commands.

Conversational interfaces, on the other hand, would provide the opportunity for the users to state what they want to do in their own terms, just as they would do to another person, and the system takes care of the complexity.

We are convinced that the utilization of speech dialogue will improve the assistance quality when using Personal Assistant Agents in some specific situations.

A good example could be a Help Desk system achieved by a telephone. As the user does not see menus, buttons or pre-determined options she can explores the application differently. The user will intuitively use the application and will try to obtain different services speaking and waiting for an answer. Without having to type or to click in a fixed configuration interface, the user will demand a service to the application and the application will perform the service or will just warn the user that the requested feature does not exist or is not supported. This will potentially accelerate the application usage, since the user does not need to find a specific feature in a sub-menu or in a “hidden” dialogue box.

Another point to study is the way users are prepared to use this kind of systems. They are not used to “direct speak” to machines, using natural language. Probably they are inclined to sub-estimate the system capacities. This is different to the previously case, where users find it difficul to use the system or have difficulty to perform an operation, since the operation entry point is hidden into deep sub-menus.

Whenever we add voice handling to the dialog, we introduce an additional complex component. The development of a spoken dialog system requires the integration of the various components of spoken language technology, such as speech recognition, natural language processing, dialog modeling, and speech synthesis as we already presented. It seems a good opportunity to use the agent approach here. Moreover, the agent approach will be used here as a tentative of organizing assistance and services. The personal assistant will be an agent, a Conversational agent. A Conversational Agent is an interface agent that is capable of developing natural conversations (it can dialog) with the user, using typed natural language statements, or even spoken statements.

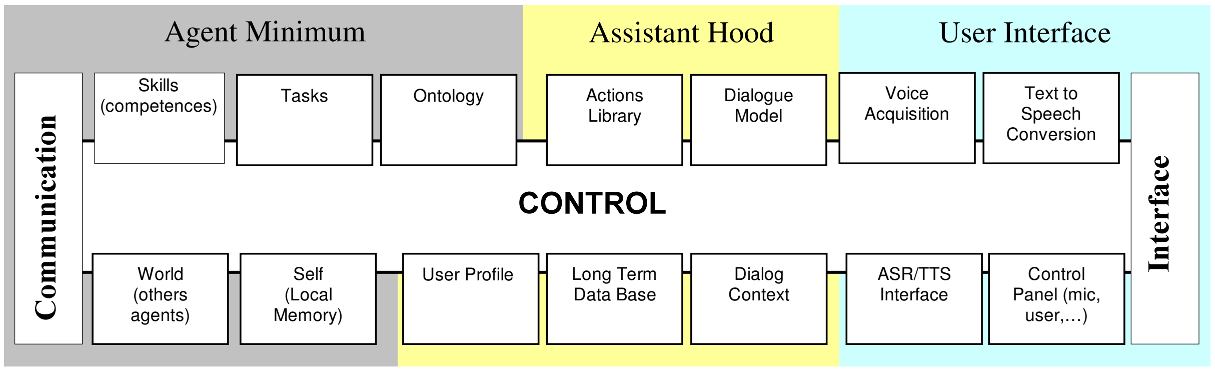

Figure 9. Voice Personal Agent Structure.

A possible personal agent with voice handling structure is shown in Figure 9. The content of each component must be analyzed from many points of view to assure that this structure is sufficient. The voice handling functions is implemented at the *User Interface* component.

6.1. Personal Agent User Interface Entry Point

The personal agent may engage in a dialogue with the user, if a previous user sentence cannot be executed, was misunderstood, or if the agent needs additional information to solve a problem. The personal agent might also start the conversation to inform the user when an event occurs (for instance: electronic message arrival, call for papers deadline, a meeting today, etc.).

6.2. Dialogue Samples

Just for clarifying the usage of the personal agent, find following a table of dialog X actions samples.

Actor	Sentence	Action fired by the PA
User	"Please, open my email account."	Open email client*
User	"Locate and open the latest version of BUCKS.doc"	Send request to Document Manager Service Agent
User	"List all meetings at my office for the next 3 days"	Send request to Agenda Manager Service Agent
PA	"A new electronic message just arrived"	Wait for the user response; without locking
User	"Look for documents on references to Assistant Agents"	Suggest a better criteria, a search engine or send request to WEB search Service Agent
PA	"…"	

* Since the personal agent realizes that the user want to work with her email client, it will open it. The email client could be voice enabled as well. So, the user remains speaking with the system by saying the correct commands to operate the email client: "new message", "send it", "forward it", and so on.

As an action result, a new interface may be attached to allow, for instance, a new email edition or a text document reading.

6.3. User Interface and Assistance Basic Structure

Figure 10 shows the structure of the first level assistance organization. Since the interface is voice-oriented, most modules are specialized in speech treatment. Note that the user may also use traditional graphical elements (buttons, menu items, etc) to operate the system.

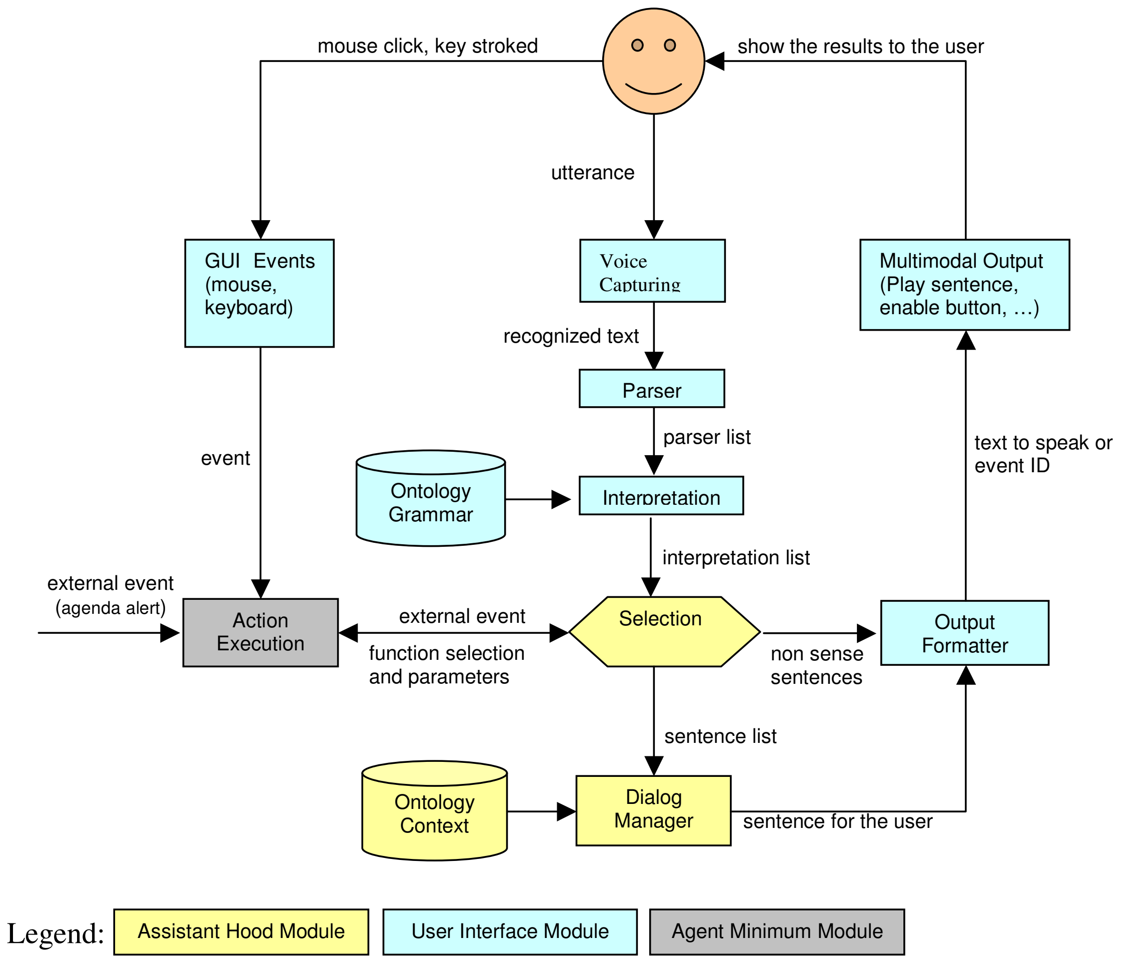

Figure 10. User Interface Basic Structure.

Brief modules description:

- Voice Capturing: voice capturing listener;
- Parser: speech-to-text translation and filtering;
- Interpretation: context matching;
- Action Execution: fire appropriate function to execute the user request (i.e.: may generate a message to a Service Agent), or send an external event to *Selection*;
- Output Formatter: text-to-speech generation and event generation;
- Dialog Manager: generates dialogue planning. Handles context (Assistant Hood);
- Selection: choose which module will treat the solicitation.

The development of this modules will be done in the next steps of this work.

References

(Clark 99) Clark, P.; Thopson, J.; Porter, B. 1999. A Knowledge-Base Approach to Question-Answering. AAAI 99 - Fall Symposium on Question Answering Systems.

(CSLU 03) Center for Spoken Language Understanding. WEB page: http://www.cslu.ogi.edu/.

(Dutoit 96) Dutoit, T. 1996. An Introduction to Text-To-Speech Synthesis. Kluwer Academic Publishers.

(IBM 01) Developing Voice Applications: An IBM White Paper. 2001.

(Kemble 01) Kemble, K. A. 2001. An Introduction to Speech Recognition. Voice Systems Middleware Education - IBM Corporation.

(Kotelly 03) Kotelly, B. 2003. The Art and Business of Speech Recognition: Creating the Noble Voice. Addison-Wesley.

(Microsoft 03) Windows Microsoft Operating System Manual.

List of Abbreviations

- ANN: Artificial Neural Network
- API: Application Program Interface
- ASR: Automatic Speech Recognition
- DSP: Digital Signal Processing
- DTMF: Dual Tone Multi-Frequency
- HMM: Hidden Markov Model
- HTTP: Hyper Text Transfer Protocol
- NLP: Natural Language Processing
- PIN: Personal Information Number
- SAPI: Speech Application Program Interface
- TTS: Text to Speech
- UDP/IP: User Datagram Protocol/Internet Protocol
- VoiceXML: Voice Extensible Markup Language

List of Figures

Figure 1. Three level components for desktop application. 3
Figure 2. Speech recognition engine configuration. 4
Figure 3. TTS system general diagram. 6
Figure 4. Voice WEB access overview (IBM 01). 8
Figure 5. Speech Treatment flow diagram. 9
Figure 6. Banking application main interface window. 10
Figure 7. Command Control Voice Banking. 11
Figure 8. Voice Server 12
Figure 9. Voice Personal Agent Structure. 14
Figure 10. User Interface Basic Structure. 16

Notices