VoiceLink: A Speech Interface For Responsive Media

by

Yi Li

M.A.Sc. in Electrical Engineering, University of Toronto, Canada, 1999
B.Eng. in Electronics Engineering, Tsinghua University, China, 1997

SUBMITTED TO THE PROGRAM IN MEDIA ARTS AND SCIENCES,
SCHOOL OF ARCHITECTURE AND PLANNING,
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN MEDIA TECHNOLOGY

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SEPTEMBER 2002

Signature of Author .....

Program in Media Arts and Sciences
August 9, 2002

Certified by .....

V. Michael Bove Jr.
Principal Research Scientist
Object-Based Media Group, MIT Media Laboratory
Thesis Supervisor

Accepted by .....

Andrew B. Lippman
Chairman, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

VoiceLink: A Speech Interface For Responsive Media

by

Yi Li

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on August 9, 2002 in partial fulfillment of the
requirements for the degree of
Master of Science in Media Technology

Abstract

We developed *VoiceLink*, a speech interface package for responsive media applications. It contains a set of speech interface modules that can interface with various multimedia applications written in Isis, a scripting programming language created at the MIT Media Laboratory. Specifically, we designed two command-and-control voice interfaces, one for *iCom*, a multi-point audio/video communication system, and another for *HyperSoap*, a hyperlinked TV program. The *iCom* module enables users to control an *iCom* station using voice commands while the *HyperSoap* module allows viewers to select objects and access related information by saying objects' names. We also built a speech software library for Isis, which allows users to develop speech aware applications in the Isis programming environment.

We addressed a number of problems when designing *VoiceLink*. In the case of the *iCom* module, visual information is used to seamlessly inform users of voice commands and to provide them with instant feedback and instructions, making the speech interface intuitive, flexible and easy to use for novice users. The major challenge for the *HyperSoap* module is the open vocabulary problem for object selection. In our design, an item list is displayed on the screen upon viewers' request to show them selectable objects. We also created an object name index to model how viewers may call objects spontaneously. Using a combination of item list and name index in the *HyperSoap* module produced fairly robust performance, making the speech interface a useful alternative to traditional pointing devices. The result of user evaluation is encouraging. It showed that a speech based interface for responsive media applications is not only useful but also practical.

Thesis Supervisor: V. Michael Bove Jr.

Title: Principal Research Scientist, MIT Media Laboratory

Thesis Committee

Thesis Supervisor .....

V. Michael Bove Jr.
Principal Research Scientist
MIT Media Laboratory

Thesis Reader .....

Christopher Schmandt
Principal Research Scientist
MIT Media Laboratory

Thesis Reader .....

Deb Kumar Roy
Assistant Professor
MIT Program in Media Arts and Sciences

Acknowledgments

As always, there are many people to thank. Foremost among them is V. Michael Bove Jr., my advisor, whom I thank not only for his wise supervision, continuous encouragement and ready accessibility, but also for affording me the freedom to develop my own research project and pursue the directions that intrigued me most.

I would like to thank my committee members, Christopher Schmandt and Deb Kumar Roy, for teaching me a lot about speech interface design, and for carefully reviewing my thesis and providing useful suggestions.

A number of colleagues and friends at the Media Lab provided me with assistance and encouragement during the course of the development of *VoiceLink*. I cannot help but thank: Stefan Agamanolis for answering me numerous questions about Isis and *iCom*; Surj Patel for helping me get the *HyperSoap* program up and running; Jim McBride and Jacky Mallett for helping me solve many Linux configuration problems; Kwan Hong Lee and Jang Kim for helping me set up the IBM ViaVoice SDK. Also, I would like to thank all the members of the Object-Based Media Group and the Speech Interface Group at the Media Lab for participating in the user evaluation and providing insightful comments and suggestions.

Special thanks go to Michael Rosenblatt, my officemate and a bicycle guru, for bringing so many fun toys (including two cool bikes) to the office.

My gratitude extends to Linda Peterson and Pat Solakoff for reminding me every deadline during my stay at the Media Lab, and to Pat Turner for taking care of ordering all the equipments I needed.

Finally, none of the work I did would have been possible without the unfailing love and continuous encouragement I received from my parents and sister throughout my life.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Many responsive media applications have been developed at the MIT Media Lab using the Isis programming language [1]. Listed below are two examples:

- *iCom*, a multipoint communication portal consisting of several stations that links different workspaces of the MIT Media Lab with Media Lab Europe in Dublin. It serves several purposes, including acting as an ambient porthole that provides a constant sense of awareness, a live interpersonal communication system, and a community messaging center [2]. Figure 1.1 shows an *iCom* station, which is composed of a projection screen and a sitting area where users can control the *iCom* station using a trackball (a big mouse).
- *HyperSoap*, a hyperlinked video program that resembles television serial dramas known as "soap operas", in which many items on the screen are linked to information stored in a database and can be selected by viewers using a device with point-and-click capability (*i.e.*, mouse, laser pointer or touch screen) [3]. Figure 1.2 shows how *HyperSoap* viewers can select objects using a laser pointer.

Figure 1.1 An *iCom* station.

Figure 1.2 Viewer of *HyperSoap* clicks on the blue jeans using a laser pointer.

In these examples, an interface based on pointing devices is used to control and interact with various media objects. Although very reliable, such an approach has several limitations:

- While it is easy to set up an *iCom* projection screen in a public place such as the hallway of an office building, it is not always possible to find a suitable sitting area to place a table for the trackball. This limits the usability of the *iCom* system.
- When selecting objects from *HyperSoap* using a pointing device, it is difficult to click on a small or fast-moving object. Also, once an object has disappeared from the TV screen, it can no longer be accessed.
- As the only means for users to control various media objects, the pointing device limits the interface experience to mouse clicking. As we integrate more responsive media applications into our daily environment, an interface that could enable more natural, transparent, and flexible interaction is desired.

The above problems can be avoided if viewers can interact with the *iCom* system or the *HyperSoap* program through a speech-based interface. For example, a trackball is no longer needed if users can control the *iCom* system reliably using voice commands. With a speech interface, *HyperSoap* viewers can select an object on the screen by simply saying its name, no matter how small it is and how fast it is moving. They can even select

objects not shown on the screen. More importantly, as an alternative to pointing devices, a speech based interface offers a more compelling and enjoyable interaction experience. For example, an interface with voice control capability enables users to interact with an *iCom* station in a relaxed and hands-free manner, engaging them in a responsive environment rather than making them feel like they are just operating yet another computer.

1.2 Goals

In this thesis, we aimed to develop a speech interface package called *VoiceLink*, which can interface with various multimedia applications written in the Isis programming language. The major goal was to overcome the limitations of the pointing devices used in the *iCom* system and the *HyperSoap* program by designing two command-and-control speech interface modules, one for *iCom* and another for *HyperSoap*. With functionalities similar to those of a trackball, the *iCom* module would enable users to control an *iCom* station using voice commands. As an alternative to a laser pointer, the *HyperSoap* module would allow viewers to select objects and access related information by saying objects' names. A second goal was to build a speech software library for Isis, which would allow users to develop speech aware applications in the Isis programming environment without having to know the details of the underlying speech recognition system, thus lessoning the burden of developers.

Compared with pointing devices, a speech based interface has its own limitations. Speech is inherently ambiguous and speech recognition is error prone, making a speech interface less reliable than pointing devices under certain circumstances. For example, while a manually impaired user with a repetitive stress injury may prefer a speech-based interface, a user with heavy accent may find such an interface difficult to use. Therefore, although *VoiceLink* was designed as a stand-alone speech interface, we did not expect it to replace traditional pointing devices completely at the current stage. Rather, the goal was to develop an alternative that can supplement the pointing devices and enhance their functionalities. The speech interface should be able to work interchangeably with a

trackball or a laser pointer to offer greater accessibility for diverse users and usage context.

1.3 Related work

1.3.1 Speech recognition

Significant progress in speech recognition technology has been made during the last several decades. Large vocabulary, speaker-independent, continuous speech recognition is now possible. State of the art systems are generally developed based on statistical modeling and data-driven approaches. With lexicon size of more than 50,000 words, they can achieve recognition rates of more than 90% for cooperative speakers in benign environment. However, recognition of conversational speech involving multiple speakers and poor acoustic environment remains a challenge. Listed below are some examples of leading commercial speech recognition systems:

- Dictation software, including ViaVoice developed by IBM Corp., NaturallySpeaking by Dragon Systems, and Voice Xpress by Lernout & Hauspie. (Lernout & Hauspie acquired Dragon Systems in 2000 and discontinued its own speech recognition software: Voice Xpress.)
- Telephone transaction systems developed by various companies, including AT&T Corp., Nuance Communications Inc., SpeechWorks International Inc., TellMe Networks Inc., and Philips Electronics NV.

In addition to the above commercial products, several academic institutions also have developed speech recognition systems. Among them are the Sphinx system [4] developed at Carnegie Mellon University and the SUMMIT system developed at MIT [5].

1.3.2 Speech interface

Speech interface is an active research area. State of the art speech recognition and understanding systems have made speech aware applications practical. However,

designing a good voice interface for a given application remains a challenge. A number of approaches for speech interface design have been explored [6], including:

- Vocabulary-driven command-and-control systems, which constrain what users can say and explore the constraint to produce robust performance.
- Natural language processing based voice interface systems, which accept and react to users' spontaneous speech.
- Multi-modal interfaces, which combine speech, gesture and other means of human communications to make the system more flexible and efficient.
- User-derived interfaces, which learn and adapt to users' behavior.

Lee [7] developed IMPROMPTU, an Internet Protocol based audio platform for mobile communications and networked audio applications. A command-and-control voice interface was implemented on IMPROMPTU client to allow users to switch applications and control each application. Other examples of vocabulary-driven command-and-control systems include voice control interfaces for cars and many of the dialog-tree based telephone transaction systems currently in use.

The Spoken Language Systems Group at MIT Laboratory for Computer Science developed GALAXY [8], an architecture for integrating speech technologies to create natural language understanding based conversational spoken language systems. With the GALAXY architecture they have developed a wide variety of telephone information retrieval systems, including: JUPITER - A weather information system; MERCURY - An airline flight planning system; PEGASUS - An airline flight status system; VOYAGER - A city guide and urban navigation system [9].

Schmandt *et al*. built "Put That there" [10], one of the first multi-modal systems, which combined speech and manual pointing to manipulate objects. Oviatt used a combination of speech and pen-based gesture to perform mutual disambiguation [11]. The result showed that although speech recognition alone performed poorly for accented English speakers, their multi-modal recognition rates did not differ from those of native English speakers.

The systems described above are interface centered. Even for natural language processing based interfaces, a set of predefined rules or grammars are used to parse users' input. An alternative approach is to develop user-derived interfaces. Good *et al.* [12] built an interface to accommodate novice users' behavior. Through careful observation and analysis of the actual behavior of many users, a mail interface unusable by novices evolved into one that allows novices to do useful work within minutes. More recently, Roy [13] proposed Adaptive Spoken Interfaces, which learn individual user's speech patterns, word choices, and associated semantics.

1.3.3 Industry standards

The SALT (Speech Application Language Tags) Forum [14] released "Speech Application Language Tags Specification Version 1.0" in July 2002. It allows developers to add speech "tags" to Web applications written in XML (Extensible Markup Language) and HTML (Hypertext Markup Language), making it possible to create multi-modal programs that can be controlled by both voice and traditional input methods. The SALT specification is also designed for applications that don't have a visual user interface, such as those accessed by telephone. Founding members of the SALT Forum include Microsoft Corp., SpeechWorks International Inc., Cisco Systems Inc., Intel Corp. and Philips Electronics NV.

A rival effort is under way to develop a standard for speech interfaces based on a technology called VoiceXML [15]. This effort is led by a group of companies including IBM Corp., Motorola Inc., AT&T Corp. and Lucent Technologies Inc. First announced in early 1999, VoiceXML originally was designed to allow applications to be accessed by telephone. Efforts are under way to add the capability to voice-enabled applications that are accessed using the Web.

1.4 Challenges and approaches

We encountered a number of problems during the development of *VoiceLink*. In the case of the *iCom* module, the challenge is to make the voice interface intuitive and flexible so that novice users in public places could easily understand how it works with minimal training. In our design, we utilized the *iCom* projection screen to inform users of voice commands and provide them with instant visual feedback and instructions. Many voice commands are seamlessly incorporated into the screen display to eliminate the need for users to remember them. To reduce false alarms, a press-to-talk button was implemented to allow users to activate and deactivate the speech interface as needed so that normal conversation between users at two locations will not be incorrectly taken as voice commands by the speech interface. Users could also toggle the state of the speech interface by saying a pair of keywords.

The major challenge for the *HyperSoap* module is the open vocabulary problem for object selection. When watching *HyperSoap*, people do not know what items on the screen are selectable, and in many cases they do not know how to call the items that they want to select. Moreover, different viewers may refer to the same item using different names, making it very difficult to assign a unique name to each object. To overcome this problem, we used a combination of item list and name index. An item list containing the names of selectable items is displayed on the screen upon viewers' request to show what objects are selectable and how to call them. A name index is created to model how viewers may call objects spontaneously. It contains a number of synonyms for each selectable object, allowing viewers to select items in a more natural and flexible manner. Ambiguity is another problem we must address: a viewer may refer to different items using a common name and they do not know how to distinguish them when they want to select one of them by speaking. We solved this problem using a combination of explicit on-screen confirmation and implicit disambiguation based on timelines.

In our system, we used the IBM ViaVoice SDK software [16] for speech recognition. It has a large vocabulary, speaker adaptive speech recognition engine. It also provides a

Software Developer's Kit (SDK) that allows users to develop speech aware applications using a set of Application Programming Interfaces (APIs), which are designed for use with the C programming language. We implemented *VoiceLink* using a combination of C and Isis under the Linux operating system. In our implementation, we installed the IBM ViaVoice speech recognition engine on a speech application server and fed recognition results to an Isis station running responsive media applications.

1.5 Thesis outline

The rest of the thesis is organized as follows: Chapter 2 and Chapter 3 describe the speech interface design for the *iCom* module and the *HyperSoap* module respectively; Chapter 4 presents user evaluation results; Chapter 5 describes the Isis speech software library; Chapter 6 concludes this thesis with discussions on future work.

Chapter 2

Speech interface design for *iCom*

VoiceLink went through two iterations of design and test. Several changes to the initial design were made based on the lessons we learned from user evaluation. This chapter presents the final design of the *iCom* speech interface module. Section 2.1 provides a brief description of the *iCom* system. Major design considerations are discussed in Section 2.2. A number of approaches for voice model adaptation are proposed in Section 2.3. The system architecture is described in Section 2.4.

2.1 Brief description of *iCom*

The *iCom* system connects several sites at the MIT Media Lab and Media Lab Europe 24 hours a day. Its normal mode is background, providing continuous ambient awareness among all sites, but at any time it can be transformed into a foreground mode for ad-hoc tele-meetings or casual interaction, without the need to dial telephones or wait for connections to be established. Echo-canceling speaker/microphones enable full duplex speech transmission. *iCom* also functions as a bulletin board for community messages or announcements, sent via email. Message titles are listed in chronological order with varying size to reflect the age and popularity of a posting. The screen projections at each site are synchronized so that people at different sites see exactly the same projection. Figure 2.1 shows an *iCom* station at the MIT Media Lab and a few screen projections for different activities.

Users control an *iCom* station using a trackball: clicking the windows changes their arrangement, allowing the display to be customized for a particular activity. Specifically, the left button of the trackball is labeled as “Select” and the right button is labeled as “Dismiss”. Left clicking on a window enlarges it while right clicking shrinks it. Left clicking on a message title causes its full text to be displayed. Right clicking on the

displayed text closes the message display. Audio at each site can be turned on or off by left clicking or right clicking on its corresponding indicator box at the bottom of the screen.

(a) **(b)**

(c) **(d)**

Figure 2.1 Illustration of *iCom* screen projections for different activities.

(a) An *iCom* station at the MIT Media Lab; (b) A typical *iCom* screen projection in the background mode; (c) *iCom* in foreground mode: users at two sites are having a chat; (d) An email message is being displayed.

2.2 The speech interface

The goal of the design is to make the speech interface as intuitive and as easy to use as possible. Although the functionality of the interface is fairly simple, it is still a challenge

to design a robust interface that is going to be used by many novice users in a public place.

2.2.1 Functionalities of the *iCom* speech interface

The speech interface for *iCom* should allow users to control an *iCom* station the same way they do with the trackball. There are four major functions for voice control of *iCom*:

- Window selection: selecting (enlarging/dismissing) a window.
- Message selection: displaying the text of a selected message, scrolling up/down the text page for a long message, and dismissing the message after reading it.
- Audio selection: turning on/off audio at a selected site.
- Feedback and instructions: providing users with feedback and instructions when necessary.

2.2.2 Design of the *iCom* speech interface

We faced three major design considerations for the *iCom* speech interface:

- Identifying voice commands.
- How to inform users of voice commands.
- How to reduce false alarms.

Identifying voice commands

The first step to designing a command-and-control speech interface is identifying a set of voice commands that best match the requirements and expectations of the users. Two questions need be answered when deciding what commands are to be included into the vocabulary.

The first question is whether to support a more natural way of saying something instead of specifying restrictive commands (for example, “please turn off the microphone” versus “microphone off”)? Rather than defining a grammar file to enable the speech interface to

accept more natural speech inputs, we specified voice commands as a list of words and short phrases for the following reasons: Unlike multi-step transactions that are typical in telephone based information retrieval applications, the task of voice control of *iCom* is relatively simple - it is essentially to allow users to select an object, such as a window or a message, on the projection screen. A command-and-control voice interface with a word/phrase based vocabulary is sufficient for this task and should produce reasonably robust performance. Moreover, in our design, we informed users of many voice commands using visual information displayed on the *iCom* screen. So a grammar based parsing algorithm will provide very limited additional benefits, while significantly increasing the complexity of the system.

The second question is whether to support synonyms, or more than one way of saying a command? The vocabulary can be as restrictive or as flexible as the application needs to be. A large vocabulary containing many synonyms will make the interface more intuitive. But there is a trade-off of recognition speed and accuracy versus the size of the vocabulary. In our design, we provided synonyms to a number of frequently used commands to make the interface more flexible (for example, users can say "close" or "close message" to dismiss the message being displayed on the *iCom* screen). The increase in vocabulary size due to synonyms did not slow down the recognition speed or affect the recognition accuracy because the vocabulary is fairly small - there are only about 50 commands.

Showing users what to say

Unlike grammar based voice interfaces that accept more natural user inputs, the *iCom* command-and-control speech interface only accepts a set of predefined voice commands. First time users of the interface don't know these commands beforehand. It is also difficult for frequent users to remember a lot of commands. Since the *iCom* speech interface is to be used by a large number of users, including many novices, in an open lab environment, its usefulness largely depends on whether it can inform users of voice commands during the course of interactions without extensive training.

In the case of voice interfaces designed for personal devices such as Personal Digital Assistants (PDAs), cell phones or cars, users have to learn and remember voice commands through manuals and repeated use, which is a burden for them. In telephone based speech portals, a list of menu items is usually recited using a text-to-speech engine or a prerecorded message at each step in the dialog tree. Feedback also takes the form of speech output. The system response time is slow since speech output takes place over a period of time [17]. These problems are avoided in our case because we utilized the *iCom* projection screen to show users what they can say and provide them with instant visual feedback and instructions. In our design, we incorporated many voice commands into the *iCom* screen display seamlessly. For example, each window on the screen is labeled with a unique name, and each message title is numbered. These window labels and message numbers are actually voice commands for window selection and message selection. A visual instruction message containing all the voice commands will also be displayed on the screen upon users' request, allowing them to quickly browse through the command set. Such a design not only enables users to understand how the interface works with little training, but also eliminates the need for them to remember a lot of commands.

Reducing false alarms

A voice interface should provide proper feedback to help users cope with recognition errors. In our design, if the speech engine fails to recognize a user's speech input twice, a textbox will appear on the screen, prompting users to read instructions for help. (The instruction message will be displayed when users say "instruction".) Since the speech engine tries to recognize any input to its microphone, loud noise or users' normal conversation will trigger the feedback message (when the speech engine cannot recognize the input) or cause an unintended action by the speech interface (when the speech engine recognizes the input and takes it as a voice command). Frequent false alarms like this are annoying and confusing to users.

To reduce the chance of false alarm, the speech interface should be able to distinguish voice commands and users' normal conversation automatically. But no existing techniques are reliable enough for this task. In our system, we implemented two features

that allow users to activate or deactivate the speech interface as needed: one is "press-to-talk button"; the other is "keyword-trigger".

The press-to-talk button is a microphone button displayed at the bottom of the *iCom* screen. Users can activate/deactivate the speech interface by clicking on the button using the trackball. The speech engine is always running, but the interface responds to users' voice commands only when it is active, and neglects users' speech after it is deactivated. The microphone button also functions as an indicator, showing a label "Mic is on" or "Mic is off" depending on the state of the speech interface. When users click on the button to activate the interface, a message box appears on the *iCom* screen briefly, prompting them to issue voice commands and reminding them that they can see instructions for help. This approach is reliable and can effectively reduce false alarms caused by users' conversation and other noises. (To further reduce interference, the volume of the *iCom* audio output could be turned down while the speech interface is active.) The limitation is that users have to use the trackball to toggle the state of the speech interface. If we can access the speech interface through a handheld device such as a PDA or a cell phone, however, the trackball is no longer needed since the press-to-talk button could be implemented on the handheld device.

With keyword-trigger, users could activate/deactivate the speech interface by saying a pair of keywords, in our case, "microphone" for activation and "microphone off" for deactivation. Keyword-trigger enables users to control an *iCom* station completely hands-free, eliminating the need for the trackball. However, the approach itself is prone to false alarm (this did not happen in our user testing though). Also, having to know and remember the two keywords is an extra burden for users. In our final design, the two features can work interchangeably. Users could choose either of them based on their preferences.

2.2.3 The interaction flow and the command set

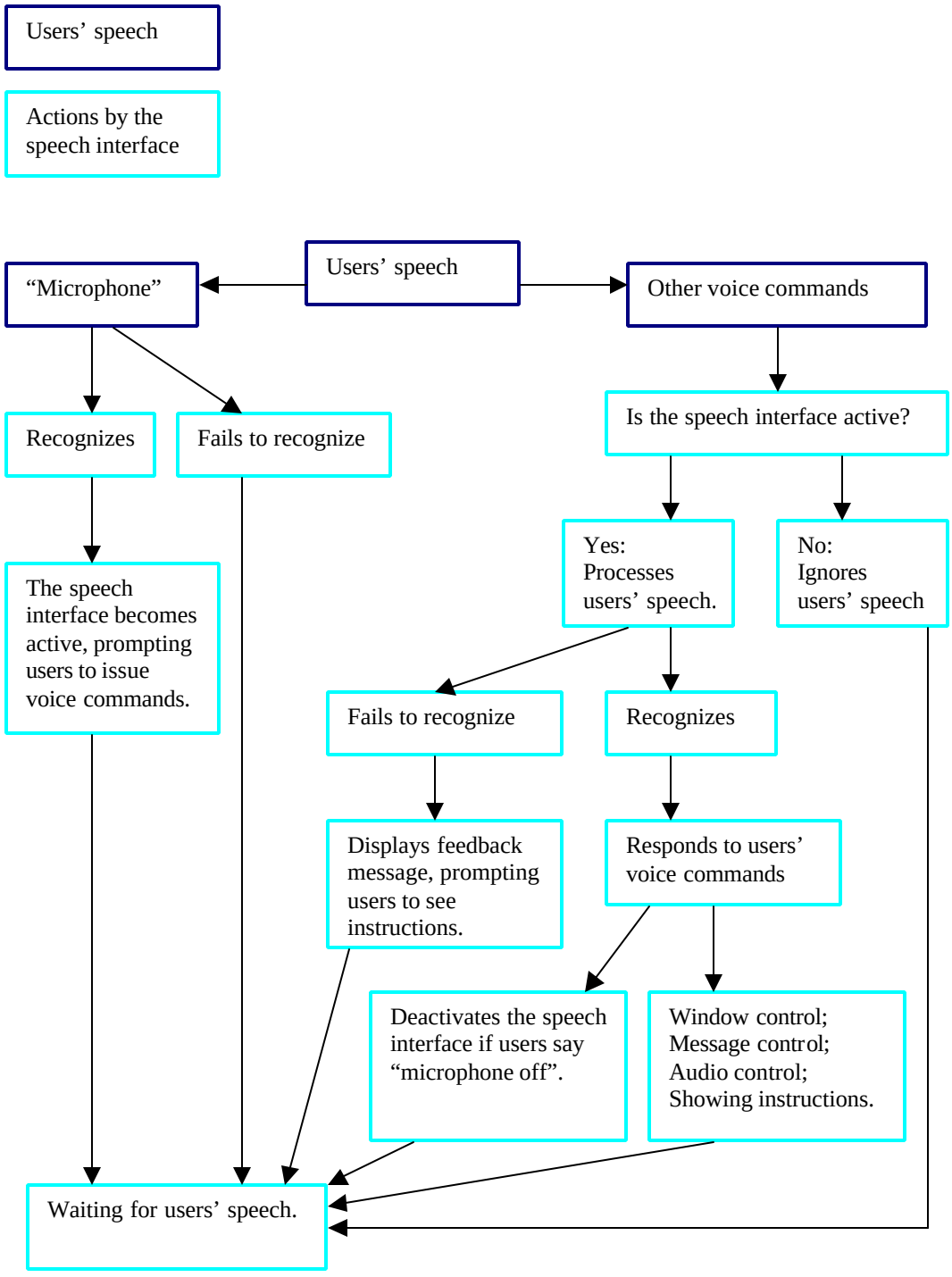


Figure 2.2 The interaction flow of the *iCom* speech interface.

Figure 2.2 depicts the interaction flow of the *iCom* speech interface. Voice commands for the *iCom* speech interface are listed in Appendix A. Many commands consist of two or three words. A multi-word command can be defined using structured grammar, but we simply define it as a phrase since the IBM ViaVoice SDK accepts such a multi-word phrase as one entry in a vocabulary. A benefit of using multi-word phrases as commands is that it can reduce unintended actions by the speech interface in noisy environment without increasing the rejection rate. Commands for window selection such as "garden" and "cube" are actually the names of the open workspaces in different parts of the MIT Media Lab, where *iCom* stations are located. Windows showing images of those workspaces are labeled with their corresponding names. Commands for message selection takes the form "message" + message number, such as "message eleven". By default, the maximal message number is thirty, because it is very rare that more than thirty message titles appear on the *iCom* screen at the same time. There are about 80 voice commands (including the 30 message selection commands) for the *iCom* speech interface. The exact number depends on how many stations are connected to the *iCom* system.

2.2.4 The new *iCom* screen display

Figure 2.3 shows the modified *iCom* screen display that is tailored to the speech interface. Figure 2.4 shows the modified *iCom* screen projections in different states. We tried to keep the change in the appearance of the original *iCom* screen projection at a minimum. However, the following changes are necessary:

- A message number, starting from one, is shown in front of each message title.
- Window labels are always shown. (In the original *iCom* system, a window label is displayed for a few seconds after users click on a window.)
- A microphone button is displayed at the bottom of the *iCom* screen.
- Message boxes for feedback and instructions are displayed when necessary.
- Two tags, labeled as "page up" and "page down" respectively, are shown when long messages are being displayed, allowing users to scroll the message text by saying "page up" or "page down".

In Figure 2.3 and Figure 2.4, no images are shown in the windows on the *iCom* screen because we didn't install the speech interface on a real *iCom* station.

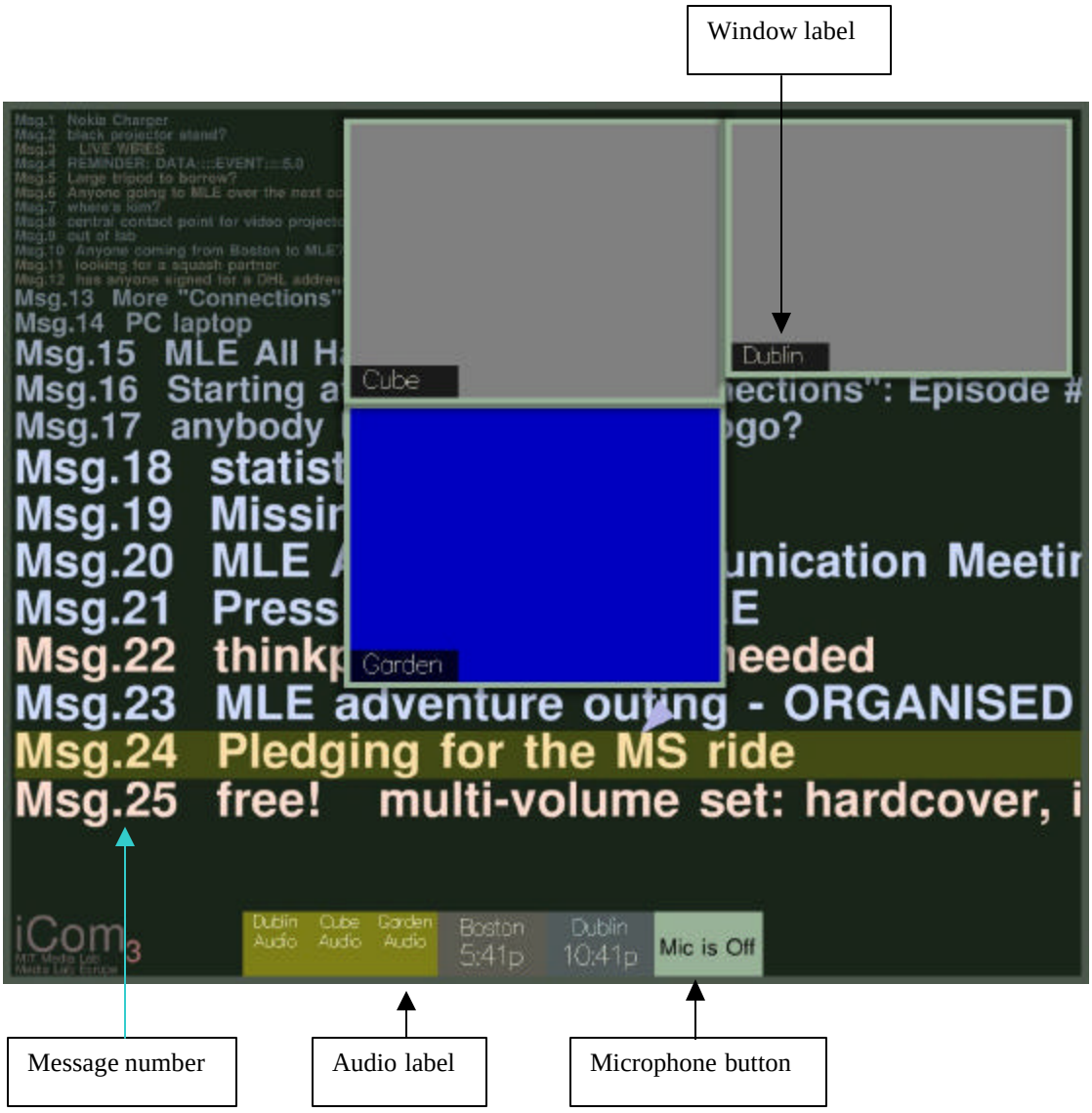
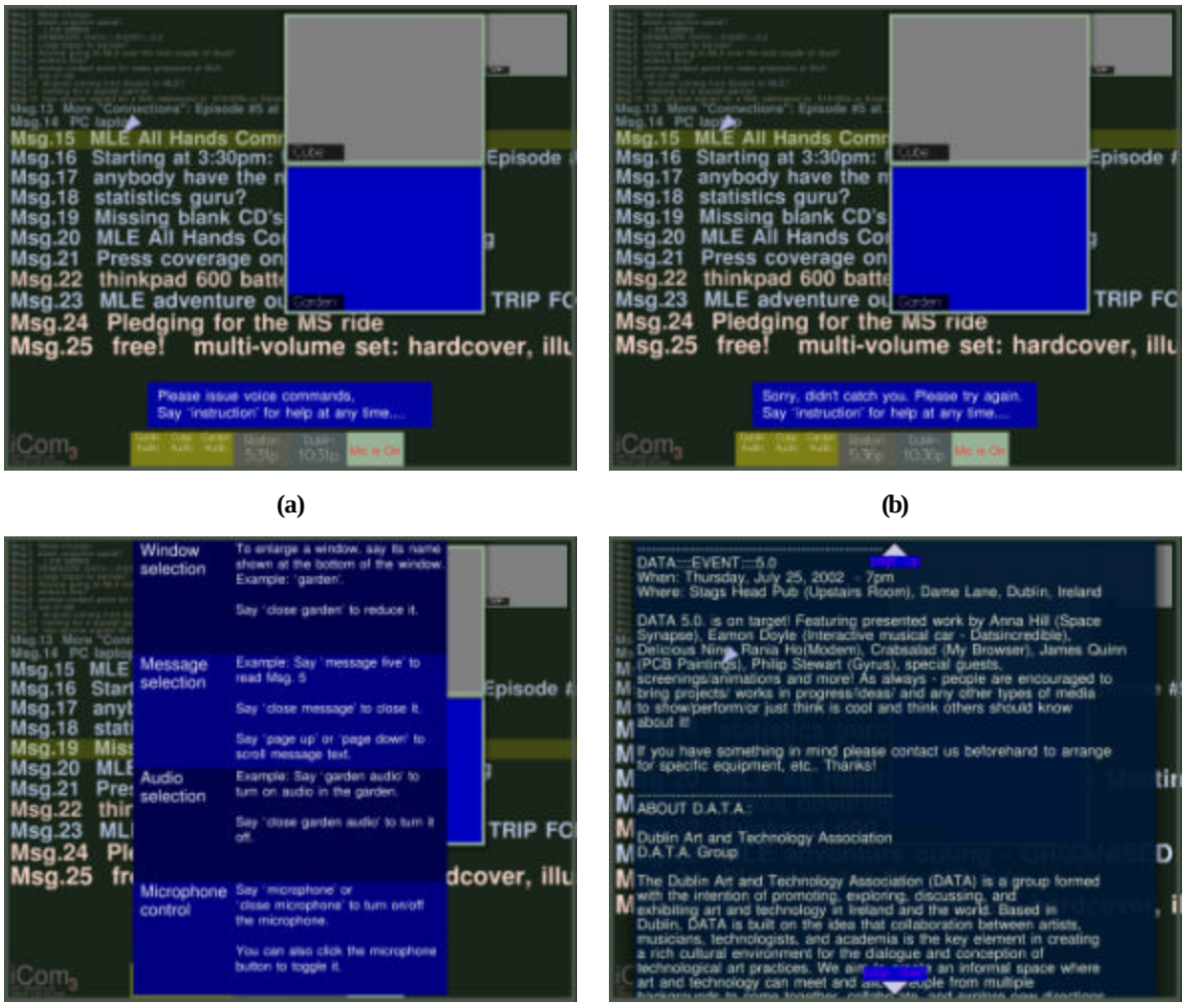


Figure 2.3 The modified *iCom* screen display tailored to the speech interface.

Several changes are made to the original *iCom* screen display: Message numbers are shown in front of message titles; A microphone button is displayed to indicate the state of the speech interface; Each window is labeled with a unique name, which is shown all the time.



(a) **(b)**

(c) **(d)**

Figure 2.4 The modified *iCom* screen projections in different states.

(a) When users click on the microphone button to activate the speech interface, a text message in blue box appears on the screen briefly, prompting them to issue voice commands; (b) When the speech engine fails to recognize users' speech, a feedback message in blue box is shown briefly, reminding users to read instructions for help; (c) An instruction box containing all the voice commands is displayed upon users' request; (d) When a long message is being displayed, two light-blue tags labeled as "page up" and "page down" are placed beneath/above the up/down arrows respectively, allowing users to scroll the message text by saying "page up" or "page down".

2.3 Voice model

The IBM ViaVoice speech recognition engine is a speaker adaptive recognizer. With the default general voice model, it can produce decent recognition results for users with a variety of voice characteristics. But recognition performance can be further improved if each user creates his/her own voice model by completing an enrollment program and uses this speaker specific model for speech recognition.

In our implementation, we used the general model for all users. Right now, there is no good mechanism for the speech interface to switch voice models adaptively. In the future, however, several approaches for voice model adaptation can be explored. For example, a drop-down user ID list corresponding to different voice models can be displayed on the screen, and a new user ID will be added to the list when a new voice model is created. Users could select their own voice models from the list when necessary. In a restricted environment with a limited number of frequent users, such as the Media Lab, face recognition or speaker identification algorithms may be employed to determine a user's ID so that the speech interface could choose the corresponding voice model automatically. Another approach is to access the speech interface through a handheld device, as already mentioned. In this case, each user ID is associated with a device ID. When a device tries to access the speech interface, the speech interface will know which voice model to use.

2.4 System architecture

We implemented the speech interface using a combination of Isis and the C programming language under the Linux operating system, since *iCom* is written in Isis while the IBM ViaVoice SDK is designed for use with C. A socket is used for communication between the *iCom* process and the ViaVoice SDK process. Each *iCom* station has a speaker/microphone for audio communication, which can be used for the speech interface. A separate microphone also could be used to make the system more flexible.

Two types of system architectures can be employed for the *iCom* speech interface: one is centralized speech server architecture; the other is localized speech interface architecture. In the first approach, a single speech server is used to process voice commands from all *iCom* stations. The speech input is digitized locally at each *iCom* station and is sent to the central speech server for recognition. Obviously, this approach saves computing resources since only one speech engine is needed. But its major limitations are complexity and slow response time. Because only one speech client can access the speech engine at a time, some polling or queuing protocols are needed so that two or more clients can share the speech engine. Heavy network traffic will slow down the speech interface's response time, and loss of packets will reduce recognition rates.

We chose the localized speech interface architecture in our implementation for its simplicity. In this architecture, each *iCom* station runs its own speech interface. Users' voice commands are directed to the local speech engine for processing. A problem with this approach is that the IBM ViaVoice speech recognition engine cannot be installed on the machine running the *iCom* process, because the Creative Sound Blaster Live PCI sound card used in our current system does not work properly if both ALSA (the Advanced Linux Sound Architecture used for the Isis audio utilities) and the ViaVoice speech engine are present in the system. Therefore, an extra machine is needed at each *iCom* site. This problem can be avoided by using another commercially available sound card: the Creative Labs Ensonic Audio PCI card, which works well for both ALSA and the ViaVoice speech engine at the same time.

In our experiment, we planed to install the speech interface at only one *iCom* site, so only one microphone button is displayed. If the speech interface were to be installed at several *iCom* sites, extra microphone buttons are needed to indicate the state of the local speech interfaces, and they can be placed above the audio indicators of their corresponding sites.

Chapter 3

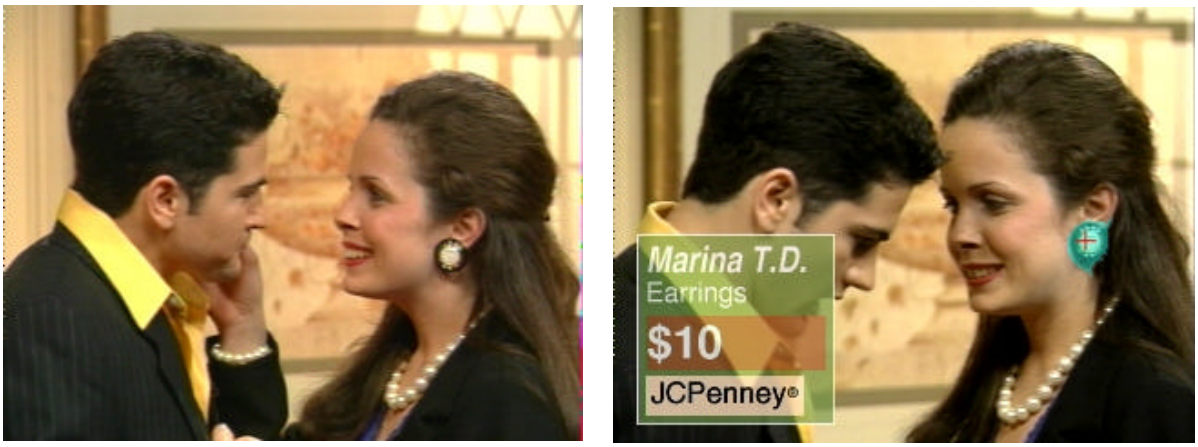
Speech interface design for *HyperSoap*

As in the case of the *iCom* module, we made several improvements to the *HyperSoap* module during the course of user testing. This chapter presents the final design of the *HyperSoap* speech interface. Section 3.1 discusses speech-enabled interactions in hyperlinked TV programs. Section 3.2 explains the major problems we must solve. Section 3.3 describes the details of the speech interface design.

3.1 Hyperlinked video and speech interface

New techniques in multimedia signal processing have made it possible to produce truly interactive TV shows such as *HyperSoap*, a hyperlinked video program produced by the Object-Based Media Group at the MIT Media Lab. In *HyperSoap*, many objects are made selectable through an interface based on pointing devices, and the user's interactions with these objects modify the presentation of the video. Using a laser pointer or a mouse, *HyperSoap* viewers can click on clothing, furniture, and other items on the screen to see information about how they can be purchased, as shown in Figure 3.1.

Another example of hyperlinked video program is *An Interactive Dinner At Julia's* [3], which is an interactive cooking show. Starting with a dinner party at Julia's house, viewers can click on entrees and decorative items at the dinner table and be shown video clips in which Julia creates them. Selecting ingredients and cooking utensils generates text boxes with relevant details. Icons are used to indicate the "path" viewers have traveled through in the show, allowing them to navigate among the video clips. Figure 3.2 shows a frame from *An Interactive Dinner At Julia's*, in which a textbox of the selected item (highlighted with green mask) is displayed. Also, viewers can switch to another video clip by clicking on the icon shown at the top-left corner of the screen.



(a) **(b)**

Figure 3.1 A pair of frames from *HyperSoap*.

(a) A frame during normal playback; (b) When the viewer clicks on the earring, it is highlighted with a green mask, and a window pops up, showing its brand, price and retailer.

Figure 3.2 A frame from *An Interactive Dinner At Julia's.*

Although very effective, the existing interface for hyperlinked video programs, which is based on pointing devices, has several limitations: (1) It is difficult for viewers to click on a small or fast-moving object. (2) Rapid change in scene makes it difficult to select objects that appear on the screen only for a short period of time. Once they move out of the screen, they can no longer be accessed. (3) Object selection is based on position

information. This implies that, to produce a show like *HyperSoap*, we have to identify and track selectable regions in every frame, which is a difficult process.

We could overcome the above problems by incorporating a speech interface into hyperlinked video programs, which enables viewers to select objects by saying their names. With such a speech interface, viewers can easily select any hyperlinked items no matter how small they are or how fast they are moving. They can even access items not shown on the screen. Moreover, object segmentation and tracking is no longer needed for the production of hyperlinked video programs. (The segmentation/tracking process is still necessary if highlighting a selected object is a desired feature.)

The concept can be extended to other types of interactive programs as well. For example, a speech interface can be embedded in a role-playing computer game, in which players can use voice commands to control their corresponding roles' actions. With a speech interface, basketball fans can retrieve a player's statistics by saying his name when watching a game on TV. (In this case, object segmentation and tracking is not only very difficult but also unnecessary.) So a speech based interface is well suited for certain types of interactive TV programs, and if properly designed, it could not only enhance traditional interface experiences but also enable new forms of interactions. However, little research has been done in this area in part because truly interactive TV shows do not exist until recently.

3.2 The challenge

Although the functionality of the *HyperSoap* speech interface is very simple: allowing viewers to select hyperlinked objects by speaking, it poses two difficulties:

- The open vocabulary problem for object selection.
- The ambiguity (or imprecision) problem for object selection.

When watching *HyperSoap*, people do not know what items on the screen are selectable, and in many cases they do not know what names or access terms they can use to select

the desired items. Moreover, different viewers may refer to the same item using different names, making it hard to assign a unique name to each object. So identifying a proper vocabulary for *HyperSoap* is extremely difficult. To make the system more accessible to untutored users, we must provide a number of access terms (or synonyms) for each selectable item. However, many synonyms are shared by two or more items. When users try to make object selection using one of those terms, ambiguity arises.

Actually, open vocabulary and ambiguity are two common problems for human-computer interface design. In many computer applications, users must enter correct words for the desired objects or actions. To increase usability and accessibility, the system must recognize terms that are chosen spontaneously by untutored users, and should be able to resolve ambiguities when necessary. Furnas *et al*. [18] studied these problems extensively and concluded that: "There is no one good access term for most objects. … Even the best possible name is not very useful." In their study, they analyzed spontaneous word choice for objects or actions in five application domains and found surprisingly large variability in word usage. For example, the probability that two typists will use the same verb in describing an editing operation is less than one in fourteen; that two cooks will use the same keyword for a recipe is less than one in five. In all five cases, the probability that two people favored the same term is less than 0.20. Their simulations show that, the popular approach in which access is via a single word chosen by the system designer will result in 80-90 percent failure rates in many common applications. To achieve really good performance, many synonyms are needed and should be collected from real users.

3.3 Design of the *HyperSoap* speech interface

To address the open vocabulary problem, we used a combination of item list and name index: An item list containing the names of selectable items is displayed on the screen upon viewers' request to show them what to say; A name index containing several synonyms for each item is created to model how viewers may call an object

spontaneously. We also addressed the problem of ambiguity using a combination of explicit on-screen confirmation and implicit disambiguation based on timelines.

3.3.1 Item list

The basic idea is to use visual information to inform users of selectable items. An item list may be displayed upon users' request, or it could be displayed automatically when it is appropriate to do so. In either case, it will appear on the screen for a few seconds. The item list not only indicates what objects are selectable but also shows users what to say.

Unlike the case of the *iCom* speech interface, where an instruction box containing all the voice commands could be shown to users, it is impractical to display a list of all selectable items in *HyperSoap*, because users simply don't have enough time to browse a long list while the video is constantly playing. Therefore, we grouped selectable items into 5 categories, each forming a small item list such as lady's item, men's items and furnishing.

We implemented the item list in two variations. In the first one, the item list is either displayed upon users request (when users say "item list") or triggered when the speech engine fails to recognize users' speech. In either case, items in the list are automatically matched to what are shown on the screen. For example, when the scene contains a man standing in front of a desk, men's items and items on the desk will be displayed in the item list, as shown in Figure 3.3. This approach has two limitations. First, in *HyperSoap*, there are always a lot of hyperlinked items shown on the screen, making the item list long, and therefore difficult to read for viewers. Second, the item list is triggered by noise once in a while, which is distracting and confusing to viewers. (In our initial design, when the item list is turned on, the corresponding objects are highlighted to give viewers a better sense of what objects are hyperlinked. But some viewers felt that such a feature is annoying and unnecessary. So we didn't incorporate it into our final design.)

However, the design described above may work well for shows containing a small number of hyperlinked objects, such as *An Interactive Dinner At Julia's*. The item list in this show will not be difficult to read since it contains only a few items at any time. In fact, in this particular case, we could have the item list displayed all the time without distracting users because there is no rapid change in scene in the program. Items in the list could be updated automatically when viewers switch to another video clip. To help viewers better associate item names in the list with objects shown on the screen, we could highlight the hyperlinked objects briefly when the item list is updated. An alternative to showing the item list is to display a nametag around each selectable object, since the position information for each hyperlinked object is available.

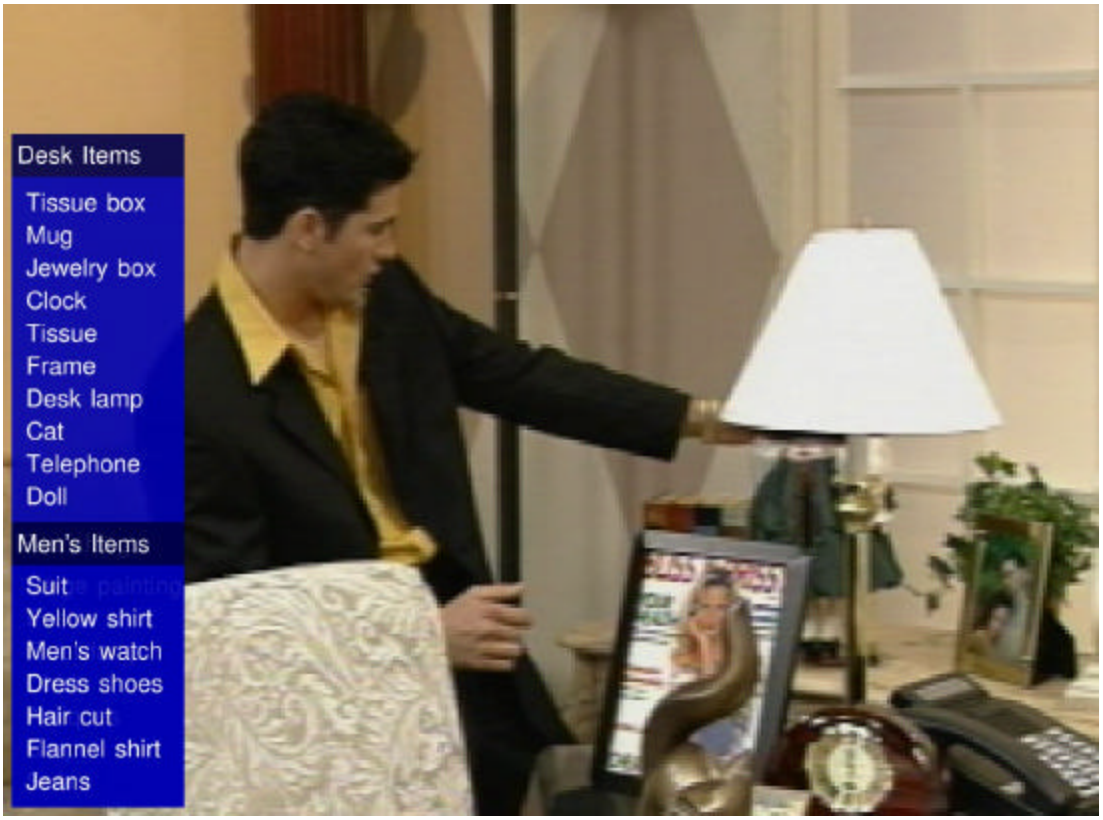


Figure 3.3 An item list containing Desk items and Men's items is displayed.

In the second variation, we implemented a two-tier item list. In this mode, a category list is always shown at the bottom-left corner of the screen. Users can choose to see items in a particular category by saying the category's name. Also, when the speech engine fails

to recognize users' speech, the system will display a feedback message, prompting users to see item list for hyperlinked objects. Most users felt that this design is better than the previous one for the *HyperSoap* program, since users have better control over which item list to see. Figure 3.4 shows the two-tier item list.

Figure 3.4 A frame containing the two-tier item list.

A category list is always shown at the bottom-left corner. Viewers can choose to see items in a particular category by saying the category's name, such as "men's items".

3.3.2 Name index

To make hyperlinked objects more accessible to viewers, we created a name index to model how viewers may call those objects spontaneously. It contains a number of synonyms for each hyperlinked object, allowing viewers to select items without having to see the item list first. As suggested in [18], a large number of synonyms are needed for a really effective name index. However, in the case of *HyperSoap*, creating an object name

index of moderate size is sufficient because, unlike abstract concepts or actions that are difficult to describe using common terms, most of the selectable items in *HyperSoap* are ordinary objects in our daily life, such as jacket, shirt and shoes, each having only one or two widely used access terms.

We first initialized the name index using the object names shown in the item list. Then we collected synonyms for each object name from real users to make the name index more "knowledgeable". In each user testing session, we documented the terms users used to select various objects, and added new terms to the index. We also asked users to provide extra synonyms if they can. The resulting name index is shown in Appendix B: the left column in the table shows the category names; the middle column shows the names used in the item list; the right column shows the synonyms for selectable objects.

3.3.3 Resolving ambiguity

It can be seen that several terms in Appendix B are shared by two or more items. For example, the word "lamp" is shared by "table lamp" in Desk items and "floor lamp" in Furnishing. Ambiguity will arise if viewers say "lamp". Note that this is not only an ambiguity to the system but also an ambiguity to the viewers since viewers may want to choose one of the lamps but don't know how to distinguish the two by speaking. This, however, is not a problem for the interface based on pointing devices since users could always click on the desired object.

Initially, when ambiguity occurs, we simply displayed information about the item that we think users are most likely referring to. This approach didn't produce consistent results. So we addressed the ambiguity problem using a combination of explicit on-screen confirmation and implicit disambiguation based on timelines. If the items involved in an ambiguity situation appear on the screen simultaneously, we will ask for users' confirmation explicitly by displaying a set of distinguishable names for all the relevant items on the screen and prompting users to choose one. Figure 3.5 shows such a situation.

Figure 3.5 Explicit disambiguation: on-screen confirmation.

A viewer says "lamp", resulting in an ambiguity. The system displays both the "table lamp" and the "floor lamp", asking for the viewer's confirmation.

If the items involved in an ambiguity situation appear in different parts of the show, we can resolve the ambiguity implicitly based on timelines: we simply choose the item that is shown on the current screen, assuming users are referring to the visible item. For example, the term "photo" is shared by two items: a photo on the table (in Table items) that appears in the first half of the show, and a photo on the bookcase (in Bookcase items) that appears in the second half of the show. If a viewer says "photo" in the first half of the show, information about the photo on the table will be shown.

3.3.4 The interaction flow

Figure 3.6 depicts the interaction flow of the *HyperSoap* speech interface. Figure 3.7 shows four frames of the *HyperSoap* program in different states.

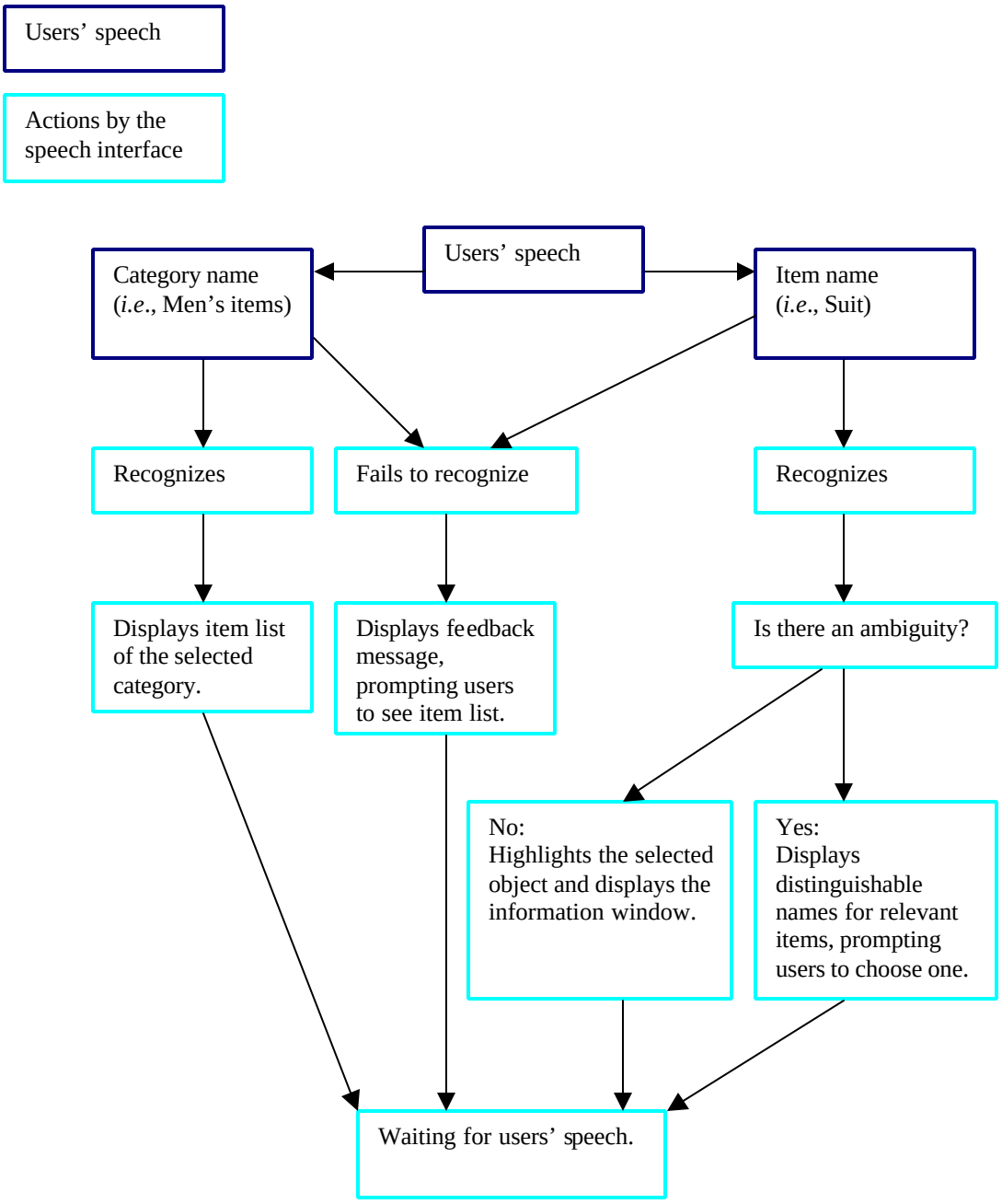


Figure 3.6 The interaction flow of the *HyperSoap* speech interface.

(a) **(b)**

(c) (d)

Figure 3.7 Four frames of the *HyperSoap* program in different states.

(a) A normal frame; (b) The photo is selected; (c) A feedback message box is displayed when the speech engine fails to recognize the viewer's speech, prompting the viewer to see item list; (d) The viewer says "picture", resulting in an ambiguity. The system displays the names for two relevant items: "photo" and "painting", asking the viewer to select one.

3.3.5 Other issues

In addition to the open vocabulary problem and the ambiguity problem, the following two issues also need to be considered.

Dynamic vocabulary

The IBM ViaVoice SDK supports dynamic vocabulary management: it allows multiple vocabularies to be active at the same time. This feature is very useful for improving the recognition rate, because instead of using a single large vocabulary for an application, we can divide it into a set of smaller ones, and activate/deactivate them as needed so that the actual vocabulary size is small. We used a single vocabulary for the *HyperSoap* program since the size of the vocabulary is moderate. (The running time for *HyperSoap* is about 2 minutes. There are 45 hyperlinked items in the show, and the total number of words/phrases in the vocabulary, including the synonyms and the category names, is around 90.) For a longer program, say, a 30 minute show, with a large number of hyperlinked items, we can create a set of small vocabularies by segmenting the show into a series of consecutive intervals, either with equal length (2 minute, for example) or corresponding to different shots, each containing a small number of selectable items that form a small vocabulary.

Interference of audio

In a normal TV-watching setting, *HyperSoap*'s audio will interfere with viewers' speech. This will result in poor recognition performance. A practical approach to solve this problem is to use a high quality directional microphone with echo-cancellation capability. But a more sophisticated method involving the separation of TV audio from viewers' speech also could be employed in the future.

Chapter 4

User evaluation

This chapter presents user evaluation results for *VoiceLink*. Section 4.1 describes the evaluation procedures. Section 4.2 summarizes the observations we made during user evaluation. Section 4.3 discusses the lessons we learned from users' feedback and several improvements we made based on their suggestions.

4.1 Subjects, tasks and procedures.

Fifteen people participated in the user evaluation. They represent a range of different voice characteristics, language skills and prior experiences in using speech recognition software. Among the subjects are 3 female speakers and 5 non-native but fluent English speakers. Two native English speakers have British accents. Several subjects are experienced and frequent users of speech recognition systems, while the others have little or no experience in using speech recognition software. Thirteen subjects are Media Lab students or faculty members who are familiar with the *iCom* system and the *HyperSoap* program; the other two are students of other departments at MIT, who have never seen the demonstrations of *iCom* and *HyperSoap* before.

The evaluation consists of two rounds. Seven people performed user testing in the first round. We made several improvements to the initial design based on their feedback, and tested the system with the remaining subjects in the second round. All the user testing sessions were held in an office. The noise level in the office varied from session to session: sometimes it was quiet and sometimes it was very noisy due to a busy surrounding environment. In a few sessions, background audio/music was also played. Both the *iCom* module and the *HyperSoap* module are tested on an IBM workstation to evaluate their voice control capabilities. In the future, we plan to install *VoiceLink* on a

real *iCom* station to evaluate its usefulness in real application settings during extended periods of use.

The procedures for a testing session are as follows. It takes about 20 to 30 minutes for each subject to complete the evaluation.

- An orientation of the *iCom* system and the *HyperSoap* program is given to users who have never used them before. After completing the tasks listed below using the speech interface, they were also asked to complete the same tasks using mouse/trackball for comparison.
- Test the *iCom* speech interface module using a general voice model, completing the following tasks.
    - Learn how the interface works and find out valid voice commands by using the system.
    - Select three different windows, enlarging each of them to its maximal size and reducing it to its minimal size.
    - Read at least five messages.
- Test the *HyperSoap* speech interface module using the general voice model, completing the following tasks
    - Learn how the interface works by using the system.
    - Select at least 10 objects while watching *HyperSoap*.
- Repeat the above two tests using speaker-specific voice models, and compare the results with those obtained using the general voice model. To create a speaker-specific voice model, a user need to complete the IBM ViaVoice user enrollment program, which takes about 10 minutes. (Only two native English speakers and two non-native English speakers performed user enrollment, because we found that the general voice model worked fairly well for most users.)
- Data gathering for the name index used in *HyperSoap*: each subject was asked to provide synonyms for object names.
- Finally, subjects were interviewed briefly about the effectiveness and usefulness of *VoiceLink*, its features, and its overall performance.

4.2 Observations

VoiceLink performed very well in quiet environment, and is robust under the presence of light noise. All the users were able to learn how to use the system very quickly and completed the tasks without difficulties. Using the general voice model, the speech engine could accurately recognize valid voice commands for most of the users, although it had problems recognizing some of the commands issued by a couple of users with heavy accents. In those cases, using speaker-specific voice models resulted in significant increase in recognition rates.

The response time of the speech interface is comparable to that of the mouse or the trackball. For most voice commands, there is no noticeable delay in system response due to the time needed for speech recognition. Only a little delay was observed for message selection using voice commands.

There were four types of errors, which are shown below. Some of the errors are identical from a user's point of view. We differentiate them here for clarity.

- Rejection: The speech engine failed to recognize valid voice commands. This happened occasionally to native speakers due to the interference of noise. Rejection rates were higher for accented speakers, but they could still interact with the system and finish the required tasks smoothly. Using the general voice model for all users, the overall recognition rate for valid voice commands is above 80% in normal office environment. There is no significant difference in recognition rate due to gender.
- Replacement: The speech engine incorrectly recognized a valid voice command as another command with similar pronunciation. This happened occasionally to users with strong accents. For example, when using the general voice model for one user, the speech interface always replaced the word "shirt" with "chair", both of which are selectable items in *HyperSoap*.
- Out-of-vocabulary: *iCom* users used invalid voice commands; *HyperSoap* users tried to pick selectable items using names not included in the object name index,

or tried to pick items that are not selectable. Out-of-vocabulary is the major source of errors for both the *iCom* module and the *HyperSoap* module. It, however, didn't result in serious user frustration or confusion, because after a few failed attempts, most users were able to know what to say by reading the instruction message for *iCom* or the item list for *HyperSoap*.

- False alarm caused by noise: The feedback message or unintended actions were triggered by noise. This happened occasionally under the presence of light noise and occurred quite often under loud noise (for example, when loud music was being played in the office during user testing). Frequent false alarms of this kind were annoying and confusing to users.

Replacement errors and unintended actions can be reduced by raising the rejection threshold, which is a speech engine parameter that can be adjusted using the ViaVoice SDK. However, the threshold should not be set too high, otherwise, rejection rate will increase. (The rejection threshold is essentially the confidence level for speech recognition results. We used the system default value, zero, in our experiment, so that any recognition results will be accepted.)

4.3 Feedback

Many users provided insightful comments and suggestions on the design of *VoiceLink*, leading to a number of improvements to the system. The following two sections describe user feedbacks on the *iCom* module and the *HyperSoap* module respectively.

4.3.1 The case for *iCom*

Most users felt that the *iCom* speech interface is intuitive and easy to use, and is effective for controlling the *iCom* station. They said that it is very helpful to incorporate voice commands into the *iCom* screen display and to provide an instruction message containing all the voice commands for quick browsing.

Many users stated that the speech interface is a useful feature for the *iCom* system, and they would like to use it on a real *iCom* station for the following two reasons:

- The speech interface enables hand-free control of *iCom*, making the system more convenient to use under certain circumstances. For example, when several users are sitting in front of an *iCom* station, they can use voice commands to control the system when the trackball is out of reach.
- The speech interface offers users a better interaction experience. It makes them feel that they are interacting with a responsive environment in a natural and relaxed manner, rather than operating a computer system.

However, a few users thought that the *iCom* speech interface is unnecessary because the trackball works perfectly well while the speech interface is not robust enough at the current stage, especially for accented users.

We learned many valuable lessons through user testing, and made the following improvements to our initial design of the *iCom* speech interface:

- Supporting synonyms: allowing more than one way of saying a command. Initially, we defined a concise command set without providing any redundant commands (one command for one function). After several user evaluation sessions, we added a number of synonyms frequently mentioned by the users to make the interface more intuitive and flexible. This also improved the consistency of the command format. For example, we only defined "close" as the command for closing message display initially, but after using the "close garden" command to close a window, many users tried to close the message display by saying "close message", assuming a "verb + object" command format. So using "close message" as a synonym for "close" results in a better match between voice commands and users' expectations.
- Adding the Keyword-trigger feature. It allows users to activate/deactivate the interface by saying a pair of keywords, instead of having to click on the press-to-talk button using the trackball. This feature enables users to control an *iCom* station completely hands-free.

- Improvement on instruction message: we replaced wordy descriptions with simple examples to make the instruction message more informative.
- Reducing the frequency of feedback. Initially, a feedback message is displayed each time the speech engine fails to recognize the speech input. However, noise often triggers the feedback message, which is confusing and annoying to users. In our final design, we cut the feedback frequency in half: the feedback message is displayed after the speech engine fails to recognize the speech input twice.

Some users also suggested that we should modify how windows are managed on the *iCom* screen. For example, they said that it would be better if a window could be reduced all the way down to its minimal size when users want to close it. We didn't make any change to window management because we want to keep the original *iCom* system design intact.

For a couple of users, visual feedback failed to capture their attention - they kept saying invalid commands without noticing the feedback message on the screen. This suggests that proper auditory cues might be used in conjunction with visual feedback to better assist users to understand the speech interface.

4.3.2 The case for *HyperSoap*

Most users enjoyed the interaction experience. They felt that the *HyperSoap* speech interface worked fairly well and the item list was very helpful. They were able to select the desired objects most of the time. Overall, they thought that the speech interface is a useful feature to the *HyperSoap* program, and it makes the interaction more seamless. However, one user in the first round of user testing said that the speech interface was not very effective because he followed the item list instead of watching the video. The major disadvantage of the speech interface, as some users mentioned, is the ambiguity problem, which does not arise at all when pointing devices are used.

We made the following changes to the *HyperSoap* speech interface during the course of user evaluation:

- Implementing the two-tier item list, which is described in detail in Section 3.3.1.
- Improving the object name index. We collected synonyms for the object name index to make it more "knowledgeable", as described in Section 3.3.2 This improved the hit rate for users' spontaneous speech, making the object selection process more natural and flexible.
- Providing ambiguity resolution. Initially, the speech interface did not have disambiguation capability. When users said a name that is shared by more than two items, the speech interface picked one of them randomly. But quite often, the randomly chosen item was not the intended one. So we added the disambiguation capability to the system, as discussed in Section 3.3.3.

Some users suggested that when they selected an item not shown on the screen, a small picture of the item should be displayed alongside the information window to give them a better sense of what they actually selected. Some users also suggested that we should allow them to browse through the video clip (fast forward/reverse or jumping to a particular point) using voice commands. These features could be implemented and tested in the future.

Chapter 5

Speech software library for Isis

The goal was to lesson the burden of speech interface developers by allowing users to develop Isis based speech aware applications in the Isis programming environment without having to know the details of the IBM ViaVoice SDK. The Isis programming language and the IBM ViaVoice SDK are briefly introduced in Section 5.1 and Section 5.2 respectively. The speech software library is described in Section 5.3.

5.1 The Isis programming language

Isis is a programming language created at the MIT Media Lab in 1995 by Stefan Agamanolis. It is specially tailored to support the development of demanding multimedia applications. Isis is very flexible and can operate on a variety of platforms, from high power workstations and servers to set-top boxes and handheld devices. It is designed to be accessible to a wide variety of users of different levels of expertise. Its small yet complete syntax lessens the burden on programming novices while still allowing experienced programmers to take full advantage of their skills. Isis also provides an efficient mechanism for extending functionality by accessing software libraries written in other languages such as C. Many of the projects being developed at the Media Lab use Isis as the development tool because of its flexibility and simplicity.

5.2 IBM ViaVoice SDK and speech aware applications

The IBM ViaVoice Software Developers Kit (SDK) includes a set of application programming interfaces (APIs) known as the Speech Manager API, or SMAPI for short, which enables an application to access the speech recognition engine. The ViaVoice speech recognition engine supports U.S. English, six European, and three Asian

languages. Multiple languages can be installed on one system, and ViaVoice allows the user to switch between them as needed. The ViaVoice SDK, by default, runs with the general office domain in the selected language. This general office domain contains more than 20,000 words representative of the office environment. The SMAPI is designed for use with the C language, but any language that supports C function calls can access the ViaVoice SDK library.

The ViaVoice SDK has several features that are very useful for developing speech aware applications:

- It allows multiple vocabularies to be active at the same time.
- It allows users to add/remove words to/from a vocabulary dynamically at runtime.
- It also allows multiple concurrent connections to the speech engine, even from within the same application.

There is a starter set of less than 20 SMAPI calls that one can use to develop a full-function speech aware application, which can handle the following tasks:

- Establishing a recognition session
- Defining and enabling vocabularies
- Directing the engine to process speech
- Processing recognized commands
- Disconnecting from the engine

In addition to the starter set, ViaVoice SDK includes many other SMAPI calls that provide more capabilities, such as session sharing and querying system parameters (including task ID, user ID, enrollment ID, and rejection threshold).

5.3 The Isis speech software library

To build the speech software library, we wrote a voice interface routine in C using the basic ViaVoice SMAPI calls and bind it into Isis. It could handle all the basic tasks needed for a command-and-control speech application. Programmers can access this

routine (thus the ViaVoice speech engine) in the Isis programming environment. We created several Isis functions that allow programmers to change the routine's behavior as needed:

- Defining a vocabulary. Developers can define an application specific vocabulary consisting of a list of words and/or phrases as an Isis list, and pass it to the voice interface routine. Currently, we don't support grammar based speech aware applications.
- Adding/removing words to/from a vocabulary. Developers can add/remove words to/from a vocabulary dynamically. This feature is useful when a developer does not know all the possible items in the vocabulary at the time of application design. For example, in a telephone dialer application, the program can load new dialers' names into the vocabulary at run time.
- Specifying a user ID. Developers can pass a user ID to the voice interface routine. This is essentially to allow the speech engine to use a user specific voice model.
- Turning on/off the microphone. Developers can pass a flag (a True/False value in Isis) to the voice interface routine to turn on /off the microphone as needed.
- Accepting speech recognition results. When the speech engine recognizes users' speech, it outputs the corresponding string. If it fails to recognize the speech, it sends out an empty string. The voice interface routine can write the recognition results to a file or send them to an Isis process through a socket, depending on which method developers choose to use.
- Adjusting the rejection threshold. As already mentioned in Section 4.2, the rejection threshold is basically the confidence level for speech recognition results. Raising the threshold can reduce false alarms caused by noise.

To use the software library, the IBM ViaVoice SDK should be installed on a machine running Isis, and at least one ViaVoice user account has to be created. Please refer to the Isis website [19] for detailed documentation about the speech software library.

The functions described above allow Isis programmers to build a very basic yet full-function speech aware application. More features, such as grammar definition, session

sharing for multiple speech applications, and dynamic vocabulary management could be included into the library to allow users to develop more complex applications.

Chapter 6
Conclusions

This chapter concludes the thesis with a summary of the *VoiceLink* design in Section 6.1 and discussions on future work in Section 6.2.

6.1 Summary

In this thesis, we developed *VoiceLink*, a speech interface package, which can interface with various multimedia applications written in the Isis programming language. It contains two command-and-control speech interface modules, one for the *iCom* system and another for the *HyperSoap* program. With functionalities similar to those of a trackball, the *iCom* module enables users to control an *iCom* station using voice commands. As an alternative to a laser pointer, the *HyperSoap* module allows viewers to select objects and access related information by saying objects' names. We also built a speech software library for Isis, which allows users to develop speech aware applications in the Isis programming environment without having to know the details of the underlying speech recognition system, thus lessoning the burden of developers.

We encountered a number of problems during the development of *VoiceLink*. In the case of the *iCom* module, the challenge is to build a robust and easy-to-use speech interface that could be used by novice users in public places with minimal training. In our design, visual information is displayed on the *iCom* projection screen to show users what to say and provide them with instant feedback and instructions. Through such a design, we not only inform users of many voice commands seamlessly but also eliminate the need for users to remember those commands. To reduce false alarms, a press-to-talk button was implemented to allow users to activate and deactivate the speech interface as needed, so that normal conversation between users at two locations will not be incorrectly taken as voice commands by the speech interface.

The major challenge for the *HyperSoap* module is the open vocabulary problem for object selection. We overcame this problem using a combination of item list and name index. An item list containing the names of selectable items is displayed on the screen upon viewers' request to show them what objects are hyperlinked and how to call them. A name index is created to model how viewers may call objects spontaneously. It contains a number of synonyms for each selectable object, allowing viewers to select items in a more natural and flexible manner. We also addressed the problem of ambiguity using a combination of explicit on-screen confirmation and implicit disambiguation based on timelines.

VoiceLink overcomes several limitations of traditional pointing devices and produced robust performance. The result of user evaluation showed that a speech based interface for responsive media applications is not only useful but also feasible, and has great potential to offer better interface experiences than traditional pointing devices. Due to the limitations of speech recognition, however, the *VoiceLink* speech interface is still less reliable than pointing devices. Therefore, we should not expect that *VoiceLink* could replace traditional pointing devices completely at the current stage. Rather we should allow the two types of interface modals to function interchangeably to offer greater accessibility for diverse users and usage context.

6.2 Future work

A number of problems should be addressed in the future to further improve the performance of *VoiceLink*.

- In addition to visual feedback, auditory cues may be employed to help users better understand the system. For example, a sound alert scheme or a text-to-speech engine could be used to indicate that the speech engine cannot recognize users' speech and prompt users to read instructions.
- The *HyperSoap* module is just one feature of speech interface for interactive TV. Other features also could be incorporated into a speech interface for interactive

TV programs. Voice enabled channel switching and a TV program guide/reminder driven by a text-to-speech engine are two such examples.

- Currently, the Isis speech software library only supports vocabulary based speech aware applications. Extension should be made to allow users to define simple grammars. Text-to-speech capability also could be incorporated into the library.
- While the general voice model produces robust recognition performance for most of the users, it does not work very well for heavy accented users. In those cases, speech recognition rates may be improved by using speaker specific voice models. A number of approaches for voice model adaptation can be explored, which are discussed in Section 2.3.
- Although *VoiceLink* is designed to function as a stand-alone interface, it can be incorporated into a multi-modal architecture. For example, speech recognition may be combined with gesture recognition to enable better interaction experiences and more robust and flexible control of various responsive media applications.
- In addition to *iCom* and *HyperSoap*, speech interfaces may be developed for other responsive media applications. *Reflection of Presence* [20], for example, is another program in which users can control and interact with various media objects using speech. We also should explore new approaches for the production of interactive TV programs tailored to speech-enabled interactions.
- Recently, IBM discontinued its offering of the ViaVoice SDK for Linux software. A good alternative is the Sphinx speech recognition system [4], which produces comparable performance as that of ViaVoice. It also has an API that allows users to develop speech aware applications. More importantly, it is open-source software. So future development of speech interfaces for Isis applications could be based on Sphinx.
- Currently, users can select objects in *HyperSoap* only by saying their names as isolated words or phrases. In the future, a keyword-spotting algorithm [21] could be used to extract object names from users' casual conversations. For example, the system could spot the word "shirt" from a speech input such as "I like the shirt". Such a keyword spotting capability will lead to more transparent and engaging interactions. Furthermore, we could incorporate a speech understanding

engine into the system so that the speech interface would be able to distinguish two different inputs such as “I like the shirt” and “I don’t like the shirt”, and react to them differently.

Appendices

A. Voice commands for the *iCom* speech interface

Functions	Commands	Behavior
Window control	Window label, or "open" + window label. Example: "garden", "cube", "open garden".	Enlarges the selected window.
	"close" + window label. Example: "close garden".	Shrinks the selected window.
Message control	"message" + message number. Example: "message one".	Displays the selected message.
	"page up", "page down".	Scrolls up/down message text for long messages that have more than one page.
	"next", or "next message", "previous", or "previous message".	Displays the next message or the previous message.
	"close" or "close message".	Closes the message display.
Audio control	Audio label, or "open"+ audio label. Example: "garden audio", "open garden audio".	Turns on the audio at the selected location.
	"close" + audio label. Example: "close garden audio".	Turns off the audio.
Instruction	"instruction".	Shows instructions on the screen.
	"close instruction".	Dismisses instructions.
Interface control	"microphone", or "microphone on".	Activates the speech interface.
	"close microphone" or "microphone off".	Deactivates the speech interface. (Users also can toggle the state of the speech interface by clicking on the microphone button using the trackball.)

B. Object name index for *HyperSoap*

Categories	Object names	Synonyms
Lady's items	Blouse	Shirt, Blue shirt, Lady's shirt
	Hair salon	Hair, Lady's hair, Hair style
	Earrings	
	Bracelet	
	Necklace	Pearl necklace
	Lady's watch	Watch, Wristwatch, Swatch
	High Heels	Shoes, Lady's shoes
	Jacket	Skirt
	Ring	
	Pantyhose	Stockings, Leg
Men's items	Suit	Coat
	Hair cut	Hair, Men's hair
	Yellow shirt	Shirt, Men's shirt
	Men's watch	Watch, Wristwatch
	Shoes	Dress shoes, Men's shoes
	Flannel Shirt	Shirt
	Jeans	
Table items	Tissue box	Box
	Mug	Cup, Coffee mug
	Jewelry box	Box
	Clock	Desk clock, Table clock
	Tissue	Napkin
	Picture Frame	Frame
	Digital Image	Picture, Photo
	Desk lamp	Lamp, Table lamp
	Sculpture	Cat
	Telephone	Phone
	Magic Frame	Frame
	Doll	
Bookcase items	Bookcase	Bookshelf
	Teddy bear	Bear, Teddy
	Globe	
	Being digital	Book
	Perl 5	Book, Computer programming
	Photo	Picture
	Plants	
	Painting	Print, Picture

Furnishing	Lamp	Floor lamp
	Chair	
	Antique	Pillar, Column
	Carpet	Rug
	Sofa	Couch
	Print	Painting, Large painting, Picture
	Poster	Painting, Print
	Framed Print	Painting, Small painting, Picture, Print

Bibliography

[1] S. Agamanolis, "Isis, Cabbage, and Viper: New Tools and Strategies for Designing Responsive Media", PhD Thesis, MIT Media Lab, June 2001.

[2] The iCom website. http://www.medialabeurope.org/hc/projects/iCom/

[3] V. M. Bove, Jr., J. Dakss, E. Chalom, and S. Agamanolis, "Hyperlinked television research at the MIT Media Laboratory," *IBM Systems Journal*, Vol. 39, No. 3-4, 2000.

[4] The Sphinx speech recognition system. http://www.speech.cs.cmu.edu/sphinx/

[5] J. Glass, J. Chang, and M. McCandless, "A Probabilistic Framework for Feature-Based Speech Recognition", *Proc. ICSLP 96*, pp. 2277-2280, Philadelphia, PA, October 1996.

[6] R. Rosenfeld, X. Zhu, et al, "Towards a universal speech interface", *Proceedings of the International Conference on Spoken Language Processing*, Beijing, China, 2000.

[7] K. H. Lee, "IMPROMPTU: Audio Applications for Mobile IP", Master of Science Thesis, MIT Media Lab, September 2001.

[8] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, "GALAXY-II: A Reference Architecture for Conversational System Development", *Proc. ICSLP 98*, Sydney, Australia, November 1998.

[9] The MIT Spoken Language Systems Group. http://www.sls.lcs.mit.edu/sls/research/

[10] C. Schmandt and E. A. Hulteen, "The Intelligent Voice Interactive Interface", *Proceedings, Human Factors in Computer Systems, National Bureau of Standards/ACM*, Gaithersburg, MD, 1982.

[11] S. Oviatt, "Mutual disambiguation of recognition errors in a multimodal architecture", *Proceedings of the ACM CHI 99*, Pittsburgh, USA, pp. 576-583.

[12] M. D. Good, J. A. Whiteside, D. R. Wixon, and S. J. Jones, "Building a user-derived interface", *Communications of the ACM*, October 1984, Vol. 27, No. 10. pp. 1032-1043.

[13] D. K. Roy, "Learning from Sights and Sounds: A Computational Model", Ph.D. Thesis, MIT Media Laboratory, September 1999.

[14] The SALT Forum. http://www.saltforum.org.

[15] VoiceXML. http://www.voicexml.org/.

[16] The IBM ViaVoice SDK. http://www-3.ibm.com/software/speech/dev/

[17] C. Schmandt, *Voice Communication with Computers Conversational Systems*, Van Nostrand Reinhold, 1994.

[18] G.W. Furnas, T.K. Landauer, L.M. Gomez, and S.T. Dumais, "The vocabulary problem in human-system communications", *Communications of the Association for Computing Machinery*, 30(11): 964-972, 1987.

[19] The Isis web site. http://web.media.mit.edu/~stefan/isis/

[20] S. Agamanolis, A. Westner, and V. M. Bove, Jr., "Reflection of Presence: Toward More Natural and Responsive Telecollaboration", *Proc. SPIE Multimedia Networks*, 3228A, 1997.

[21] T. Burianek, "Building a Speech Understanding System Using Word Spotting Techniques", Master of Engineering Thesis, MIT Department of Electrical Engineering and Computer Science, July 2000.