HYPERVOICE: GROUPWARE BY TELEPHONE

by

Paul Resnick
B.S. Mathematics, Univ. of Michigan (1985)
S.M. EECS, MIT (1988)

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE
at the
Massachusetts Institute of Technology
September 1992

Signature redacted

Signature of Author ______________________________
Department of EECS
September, 1992

Signature redacted

Certified by ______________________________
Thomas Malone
Patrick J. McGovern Professor of Information System
Sloan School of Management

Signature redacted

[illegible]

Accepted by ______________________________
Campbell L. Searle
Departmental Committee on Graduate Studies

HyperVoice: Groupware by Telephone

Paul Resnick

ABSTRACT

Many useful group communication and coordination applications appear to require large investments in portable and desktop computers. Instead, why not leverage the huge installed base of touch-tone telephones? This thesis presents user interface improvements and software tools that enhance the viability of phone-based groupware.

Skip and Scan is a new user interface style for audio documents that gives listeners some of the same control that readers gain from eye gaze shifts. In laboratory experiments, subjects selected options from Skip and Scan menus faster than from other menus. More generally, Skip and Scan begins the process of developing authoring and listening skills that harness the power of random access audio, a process that is likely to continue long after telephone sets in their current form become obsolete.

HyperVoice is an application generator for group communication and coordination applications. It includes a high-level language based on abstractions that even non-programmers can understand. From specifications in the high-level language, the HyperVoice interpreter automatically generates user interfaces that conform to the Skip and Scan guidelines. Field trials totaling more than 7000 phone calls demonstrate that HyperVoice can generate group communication applications that are both usable and useful.

Thesis Supervisor:	Dr. Thomas W. Malone
Title:	Patrick J. McGovern Professor of Information Systems, Sloan School of Management

ACKNOWLEDGMENTS

Tom Malone, my thesis advisor, taught me to trust my intuitions, even when they differed from his. He taught me the importance of presenting ideas in a clear and compelling way, and tried to pass on his skill at doing so. Many of his ideas also found their way into the thesis, including the value of semi-structured messages and the speculation that future generations may not rely on written communication, even for complex ideas.

The other committee members, Hal Abelson, Randy Davis, and Chris Schmandt, all shaped the research and its presentation, each in their own way and perhaps more than they realize.

Collaborators at GTE Labs contributed to the audio menu experiments and more. Bob Virzi was the person I called first to get a reality check on any new idea about Skip and Scan or the automatic generation of Skip and Scan interfaces. I only regret that I couldn't find him a parking space at MIT. Don Ottens ran all the subjects for the experiments and also helped with the data analysis. Sandra Teare and Kate Dobroth recorded prompts.

Mel King got me started on this project with his plans for a community information center.

Ben Shneiderman first suggested the hypertext analogy that led to the name HyperVoice.

Pamela Samuelson and Richard Stallman recorded arguments for the intellectual property discussion and Jessica Balaban recorded prompts. Richard Mullins and Kim Johnson helped design the Mandela applications and Elisa Hill recorded the prompts. Charlie Welch administers and publicizes the Boston Peace and Justice Event Calendar. My mother, Lauren Resnick, suggested the teachers' curriculum line and Vicki Bill, Mary Leer and Leslie Clark made it happen. Gwendolyn Lee did all the fundraising and publicity for U-TALK and Kathy Peck recorded the prompts. Thanks also to everyone who called, especially Ron Newman.

Thanks to my colleagues at the Center for Coordination Science and elsewhere at MIT, Barry Arons, Geoff Bock, Amy Bruckman, Erik Brynjolfsson, Kevin Crowston, Marc Davis, Christopher Fry, Bob Halperin, Debby Hindus Troy Jordan, Kum-Yew Lai, Jintae Lee, Wanda Orlikowski, Brian Pentland, Mike Plusch, Mike Travers, and Joanne Yates. Jolene Galegher and Mark Ackerman deserve special mention for helping me to analyze the results of the teachers' curriculum line study. Mark's daughter Rebecca provided comic relief on trips to the Friendly Eating Place.

Outside MIT, Eric Aboaf, Deborah Servi, Dan Ellis, and Sabrina Birner contributed to the presentation of the ideas in the thesis; thanks especially to Deborah, who helped with crucial last minute editing. Mary McCartney constructed the 12x12 Latin square as a mathematical diversion one rainy afternoon. Many others played important support roles, especially Lydia Tuden, Jeffrey Alan (Charlie Brown Duke) Stock, Erik Ekwurzel, Pradeep Dalal, all my music and soccer buddies, and the cooks and guests at Ben & Eve's.

My parents encouraged and advised me along the way and even thought I was doing something interesting. While I was in graduate school, my big brother Jeremy did neighborhood political organizing, wrote software, taught in a high school, married Pamina Ewing, and fathered two beautiful daughters, Sarah and Aliza. Sarah can already write her own name and read books.

Thanks to all of you. I couldn't have done it without you and wouldn't have wanted to.

TABLE OF CONTENTS

1 INTRODUCTION

Groupware applications help people to share information, by collecting it from some of them, then distributing it to others and arranging it in ways that they find useful. For example, groupware applications can support distribution of announcements, issue discussion, collection of commonly asked questions and distribution of answers, matching (e.g., buyers and sellers), and project status reporting. Telephones have largely been overlooked as a platform for groupware applications, but they offer several attractive features:

Networking. A single network connects nearly all telephones worldwide. By contrast, many groupware applications that run on workstations require all of them to be on the same local area network. Users already know how to dial telephone numbers to make telephone network connections. By contrast, connecting computers through modems for the first time is often a several hour ordeal.

Audio. Telephones do not require writing, typing, or reading skills. In addition, spoken voice carries nuances that text cannot.

Huge installed base. Most offices and homes have touch-tone telephones. Travelers can expect to find touch-tone telephones at all but the most remote destinations, and many places en route, including hotel rooms, airports, airplanes, street corners, and, with mobile phones, streets. For many groups that span organizational boundaries, telephones and fax machines may be the only shared communication devices. Telephones are also the only available communication device commonly available in residences.

This thesis addresses three interpretations of the question, "How can we build usable telephone-based groupware applications?"

The first interpretation emphasizes the word 'usable'. *Wait and wilt* characterizes the interface style in most existing applications: listeners wilt while waiting through the recitation of long sequences of options. Research on ways to improve usability led to the *Skip and Scan* interface style: by frequently skipping, listeners can scan the contents of audio documents.

The second interpretation emphasizes the word 'groupware', asking how multiple authors can contribute information in a format that lets a computer program route it to the appropriate listeners and arrange it in helpful ways. This led to the development of structured input techniques that help callers to segment voice recordings and annotate them with symbolic information such as typed-in dates.

The third interpretation emphasizes the word 'build', asking what software tools can help application developers, who may be neither human factors experts nor expert programmers, to build usable groupware applications. This led to the development of an application generator, *HyperVoice*.

1 SAMPLE APPLICATION: EVENT CALENDAR

Consider a concrete example of a phone-based groupware application, the Boston Peace and Justice event calendar. It allowed callers to listen to announcements of upcoming events and to add new event announcements from any touch-tone phone. A menu asked callers to select one of two categories, ongoing activities or upcoming events. Callers could then press touch-tone buttons to interrupt and skip back and forth between announcements in a category, or press a different button to add a new announcement.

Announcements were segmented into six fields to encourage people to include important information about events, including a headline, the date and time, location, sponsoring organization, contact telephone number, and details. All the information was recorded. A moderator listened to all the new announcements and used cut and paste operations, also by phone, to sort them by event date and erase them after they were no longer relevant.

Later implementations of event calendar applications included typed-in dates, so that these sorting and filtering operations could be accomplished automatically.

The event calendar was first publicized, through flyers and word of mouth, during the Gulf War (January-March, 1991). During the war, the service handled as many as eighty calls per day. The emcee at a city-wide rally described it as the best source of up-to-date information about events. It was even featured in a National Public Radio story about how the student anti-war movement was well-organized, if not popular. As of January 1992, the system had handled well over 5000 calls, with more than 300 recorded announcements from an estimated 40 different people.

2 METHODS

This project employed several research methods, including iterative design, field trials, laboratory experiments, and analysis from first principles. Iterative design and pilot testing of complete applications led to user interface improvements and evolution of the HyperVoice application generator. Field trials confirmed the feasibility of phone-based groupware and suggested context variables that will influence the adoption of particular applications. Laboratory experiments, conducted in collaboration with Robert Virzi at GTE Laboratories, compared the usability of three interface styles for telephone menus. Analysis from first principles yielded a set of design guidelines for multi-author audio documents that are consistent with the empirical results from the field trials and laboratory experiments.

3 SOURCES OF POWER

This research draws on two sources of power. First, a little structure in voice messages can go a long way. Second, the same high-level abstractions are helpful in the design, programming and use of applications.

3.1 STRUCTURED MESSAGES

Structured messages can help contributors to add information, listeners to scan it, and the computer to sort and arrange it. A structured message can include recorded fields (as in the event calendar) and symbolic fields, such as dates, numbers, and pointers to other objects. Field names remind contributors of important information to include and the division of messages into segments allows them to correct mistakes without throwing away long recordings. Listeners can skip through the fields at their own pace instead of waiting through long recordings. Symbolic fields represent knowledge about the contents of voice recordings in a format that a computer program can manipulate. For example, the computer can automatically filter and sort recorded information based on the associated date or numeric fields.

3.2 CLOSING THE ABSTRACTION GAPS

This thesis presents four building blocks for phone-based groupware applications:

- *Lists* present ordered sequences of information objects. The objects can contain links to other lists, which callers can follow. Callers can also initiate the addition of new objects from some lists.
- *Menus* are special cases of lists, where the information objects serve only as prompts to inform callers of links to other lists.
- *Forms* allow contributors to fill in the contents of new objects that they are adding.
- *Login processes* restrict access to applications, determine what information callers will hear first, and determine what privileges they will have for adding information.

Design, programming and use of applications all rely on these abstractions. They are easy to understand, so that even non-programmers can participate in designing applications.

HyperVoice reifies the abstractions as programming language primitives. Listeners and contributors treat lists, menus, and forms as metaphors to aid navigation and information entry.

Moreover, reliance on the *same* abstractions allows designers, programmers, and users to speak the same language. Designers can put themselves in users' shoes and hence anticipate usability problems. Programmers can more faithfully implement designs. The audio documents that programmers create consistently will match users' navigation metaphors.

4 RESULTS

4.1 PROOF OF FEASIBILITY

The Boston Peace and Justice event hotline described above provides the strongest evidence that it is possible to make phone-based applications that are both usable and useful in supporting coordination activities. Several other field trials supported group discussions, including an MIT-wide opinion forum that handled more than 1000 calls, with more than 150 recorded comments over the course of two months. A group of elementary school teachers used another application to hear sample lesson plans and to record success stories and comments about a new math curriculum they were using. These field trials validate the overall concept of supporting coordination with phone-based communication applications.

4.2 CONTEXTUAL FACTORS

Some field trials were more successful than others. Contrasts between the applications and user populations suggested a set of context variables that will influence the adoption of particular applications. For example, phone-based groupware applications are more likely to succeed when there is time-critical information, when the expressiveness of voice is valuable, and when users have weak keyboarding or reading skills. Some of the negative factors are a need for anonymous contributions of information, a poor distribution of costs and benefits among users, and a need to remember large information chunks.

4.3 'SKIP AND SCAN' USER INTERFACE STYLE

The user interface research in this project culminated in the articulation of the *Skip and Scan* interface style. In a Skip and Scan interaction, callers listen to short fragments of speech and then interrupt, pressing buttons to skip to other speech fragments. By frequently skipping, callers hear only a small fraction of the contents of an audio document without spending a lot of time deciding which buttons to press, just as frequent eye gaze shifts allow readers to visually scan a printed page. The Skip and Scan slogan led to the development of new audio menu styles and to a set of more general audio document design guidelines.

4.3.1 Audio Menu Styles

Skip and Scan audio menus allow callers to skip at their own pace through hearing the options in the menus, rather than waiting for the computer to automatically advance after each option is recited. This additional user control can lead to faster selection times because callers can often reject a menu option before the end of its recitation. Laboratory experiments compared the current standard menu style with two menu styles that allow callers to skip and scan. Overall, subjects preferred one of the Skip and Scan styles, called the 3-button, and made selections faster with it after the first few trials.

4.3.2 Audio Document Design Guidelines

A set of design guidelines lists properties of audio documents that will help listeners to scan. Properties such as short recordings and progressive disclosure help listeners to know when it is safe to skip the rest of a speech segment. Regularities, orderings, and metaphors help callers to predict what button presses are available and what their effects will be without hearing prompts. The guidelines can help designers to evaluate alternative designs without implementing them and conducting behavioral tests. They also identify the properties that multi-author documents need to preserve as authors add new information to them.

4.4 'HYPERVOICE' APPLICATION GENERATOR

While the Skip and Scan guidelines help designers to *evaluate* audio documents, HyperVoice helps designers to *generate* audio documents that conform to the guidelines. Previous software tools express phone-based applications in terms of state-machine abstractions: programmers specify *nodes* that contain speech segments, and *transitions* between segments labeled by the telephone buttons that initiate them. By contrast, HyperVoice introduces a new abstraction layer, called the *application layer*, based on the four primitives described above, login processes, lists, menus, and forms. Programmers specify a few parameters that determine how information will be arranged in lists, initial values for information entry, and user interface choices such as which menu style to use. An interpreter then automatically generates a state-machine layer representation of the application.

Automatic generation of state-machine layer programs offers three advantages over direct specification of state-machine programs:

Faster development. Application layer programs are shorter and easier to debug.

Participatory design. Application-layer programs use the same abstractions that are useful in informal descriptions of designs.

Consistent, good user interfaces In generating a state-machine layer program, HyperVoice automatically determines dialogue sequencing and the text of prompts. A state-machine programmer may accidentally write a prompt that tells the caller to press the wrong button. With HyperVoice, automatic generation and placement of prompts insure that they match the available actions.

Naturally, HyperVoice cannot guarantee that all the document design guidelines will be followed. For example, a contributor can still record a long, rambling message. Instead, HyperVoice is a catalyst for collaboration among listeners, contributors, and application developers. It provides convenient ways to elicit information from contributors and format it for listeners, so that developers can create audio documents that expand gracefully.

5 IMPLICATIONS

In the short-term, the Skip and Scan interface style, if widely adopted, could increase user satisfaction with phone-based applications. Automatic generation of user interfaces from high-level abstractions could speed up development of applications, reduce the programming and human factors skills required of developers, and enhance the quality of the interfaces. The addition of structure to voice messages could enhance the utility of voice mail systems and integrate well with other technologies such as fax and email. Finally, the field trials suggest that there is a large and largely untapped opportunity to use telephones as a platform for many-to-many communication applications.

In the longer term, the skip and scan guidelines are a first attempt to define the elements of style for audio documents. Digital storage and random access have ushered in a new era in audio, but as a society, we do not yet have the authoring or listening skills relevant to the new era. As people gain more experience authoring and listening to audio documents, their utility will increase, but no one can predict yet by how much. Right now, even the best audio document is much harder to scan than a well-designed visual document, but visual document design has a several thousand year head start.

6 OUTLINE OF THE THESIS

Chapter 2 presents application designs and field trials of telephone bulletin-board applications. The application designs identified functionality requirements for the application generator. Pilot tests of the applications led to iterative improvement of the user interfaces that HyperVoice generates. Results from the field trials, both positive and negative, suggested a set of context variables that will influence the success or failure of applications.

Chapter 3 takes the user interface insights gleaned from the field trials as inspiration for a more general analysis of audio document user interfaces. It first sets out the interactions that Skip and Scan audio documents should afford, based on an analogy to visual scanning of paper documents. Then, it presents the properties of the documents that will facilitate such interactions.

Chapter 4 presents the HyperVoice application generator. It describes the features of the application layer language and interpreter that make it easy to specify the functionality required for the application designs, and to generate user interfaces that satisfy the Skip and Scan document properties. While Chapter 2 shows that lists, menus, forms, and login procedures are good design abstractions and Chapter 3 shows that they are good metaphors to help users scan audio documents, Chapter 4 treats those abstractions as programming primitives. Appendices A through C present more technical details about the application generator and sample programs written in the application layer language.

Finally, chapter 5 presents the results of the two laboratory experiments comparing user performance with the three audio menu styles. Appendix D presents technical details of the second experiment.

2 Application Designs and Field Trials

Telephone bulletin boards are audio documents that allow remote entry and selective retrieval of recorded information. If configured appropriately, they can support a wide range of communication and coordination activities, including discussion of issues, publicity of announcements, answering commonly asked questions, matching, and status reporting. This chapter describes prototype HyperVoice applications designed to support these activities. Several of the prototypes have been implemented and tested in field trials.

The application designs and field trials contributed to this research in five ways.

Proof of concept. Three successful field trials confirmed that telephone bulletin boards can support cooperative work and play. Without prior training, callers listened to and recorded messages. In all, the field trials included more than 7000 calls, with more than 700 messages added.

Design abstractions. A few primitive components, lists, menus, forms, and login procedures, are sufficient to describe all the application designs. Several non-programmers understood the design abstractions well enough to generate and evaluate alternative designs.

User interface refinement. Observation of people using the prototypes, during pilot testing and field trials, led to many user interface improvements.

Software tool refinement. The HyperVoice application generator evolved to handle all the variations in functionality required for the applications. The software tools also evolved to incorporate the user interface refinements.

Success factors. The field trials identified variables that determine whether telephone bulletin board applications will be effective in different contexts.

This chapter first describes each of the application designs and field trials. The designs describe how the four primitives (lists, menus, forms, and login procedures) are instantiated and combined. The field trial descriptions focus on the role non-programmers played in the design processes, on user interface lessons and on factors that contributed to the success or failure of the application. The next sections then summarize the functional requirements of the application designs, the user interface insights, and the success factors.

1 APPLICATION DESIGNS AND FIELD TRIALS

The field trials took place over the course of two and a half years, as summarized in Figure 2.1. The text presents them chronologically, except for the MIT U-TALK issue discussion, which directly follows the other two issue discussion applications even though it occurred much later. This section also includes designs for several more generic applications that have not yet been tested in field trials.

Application Name	Participatory Design?	# of calls	# of messages added	Start of Trial	duration of trial
Joke Collector	No	~40	~10	January, 1990	1 day
Issue Discussion 1: class discussion	No	~20	~10	February, 1990	4 days
Issue Discussion 2: Intellectual property	No	~150	3	April, 1990	3 days
Issue Discussion 3: MIT U-TALK	No	1030+	152+	April, 1992	2 months +
Mandela Task Tracking	Yes	1	0	June, 1990	7 days
Mandela Events Calendar and Volunteer signup	Yes	1378	more than 200	June, 1990	10 days
Peace and Justice Events Hotline	No	4578+	more than 300	January, 1991	1 year +
Teachers' Curriculum Line	Yes	72 by head teacher; 66 by others	57 by head teacher; 10 by others	October, 1991	6 weeks

Figure 2.1 A summary of the usage statistics for the field trials. The Peace and Justice Event Hotline and MIT U-TALK were still in operation as of June, 1992.

1.1 JOKE COLLECTOR

1.1.1 Application Design

The first and simplest application lets people share jokes. There is no login procedure. Callers first select a category of jokes from a navigation menu. Any caller can add a joke to any category, but only the system administrator can add new categories.

1.1.2 Field Trial

Visitors to an MIT AI Lab open house in January, 1990 tried out this application. A number of people recorded jokes, some of them funny.

User Interface Results

While observing callers interact with the system, I identified three drawbacks in the user interface for this application. First, it emphasized a metaphor of controlling a compact disc jukebox, with each joke a track and some tracks containing links, that, if followed, caused other discs to be loaded onto the player. This metaphor failed because people were unfamiliar with the source domain: some had never experienced skipping between songs on a CD player, and no one had ever used a jukebox with links between songs and other discs (no such devices exist). The metaphor was abandoned in later user interfaces.

Second, time-dependent options did not accommodate individual differences. Like a CD player, the system automatically advanced between tracks, but that meant that a link that was available from one track was suddenly no longer available when the system advanced to the next track. One six or seven year old child who tried the system did not make decisions fast enough and got hopelessly lost. This lesson influenced the later choice of which menu styles to experiment with, as described in Chapter 4, section 5.2.2.

Third, the application began with a tutorial that explained the discs and tracks metaphor and let people practice moving around. People seemed not to like that: they wanted to get right to the jokes. During the tutorial, one person said aloud, "So much text." Later prototypes interspersed the tutorial with the application, introducing commands as they were needed.

1.2 ISSUE DISCUSSIONS

Issue discussion applications allow callers to record comments and opinions. Other callers can then listen to those comments and record responses or unrelated comments.

1.2.1 Application Design

A simplistic design for this application uses the joke collector as a model, except that people record comments instead of jokes. Comments are grouped together into lists, one for each topic. Each comment contains two fields, one for a headline, similar to the subject line of an electronic mail message, and one for the contents. As with a radio talk show, if callers want to indicate that a comment is responding to someone else's comment, they use standard rhetorical tricks, such as summarizing the other comment.

A more sophisticated design uses pointers between objects to keep track of which comments are responding to which other comments. The system creates a list of responses each time a comment is added. Then, when listening to a comment, a caller can press a button to go to the list of responses. There can even be responses to responses. Thus, the tree of comments grows over time solely through operations performed by telephone. Chapter 4 will discuss how HyperVoice plays back the headline fields of comments in order to automatically generate meaningful descriptions of the response lists.

Two features help moderators maintain order in issue discussion applications. First, each new comment is added to a master list in addition to the current list of comments. This linearizes the comments so that the moderator can hear all the new ones that have been added without having to traverse the entire tree of comments. From the master list, the moderator can follow a link from any comment to the list to which it was originally added. Second, the moderator can move comments to more appropriate places or delete them altogether, using cut and paste operations.

1.2.2 Field Trials

Three versions of this more sophisticated design were prototyped and tested in field trials. The first was a discussion of the key issues in a business case study for a class at MIT. The second was a forum on intellectual property rights for software user interfaces. The last was an opinion forum for the MIT community, hereafter referred to as U-TALK, which spells out the five-digit phone number people called to access the forum.

1.2.2.1 Class Discussion

The first field trial was an adjunct to a class discussion in a class for MBA students. Following the regular format of the course, students were given a business case to read and analyze, and a set of questions for which to prepare answers. Each of the questions, however, was also recorded as a discussion topic in a HyperVoice application. Students were invited to record comments in response to the questions and to respond to other students' comments in the days leading up to the class during which the case was discussed. Students were briefed on the concept a week beforehand, including a verbal explanation of the interface style. A copy of Figure 2.2 was also handed out in class.

User Interface Results

The user interface for this application adopted a spatial metaphor. Four of the buttons on the telephone keypad were assigned as arrow keys, as shown in Figure 2.2. Callers started at the top left and moved down in order to hear the first question, then right to hear the other questions. Each question had a list of comments that responded to it. From the question, a caller could move down to hear the first comment, and then right to hear the others, and back up to the question. Similarly, each comment had a list of response comments below it. At the end of the comments at one level, the caller could move right again in order to start recording a new comment.

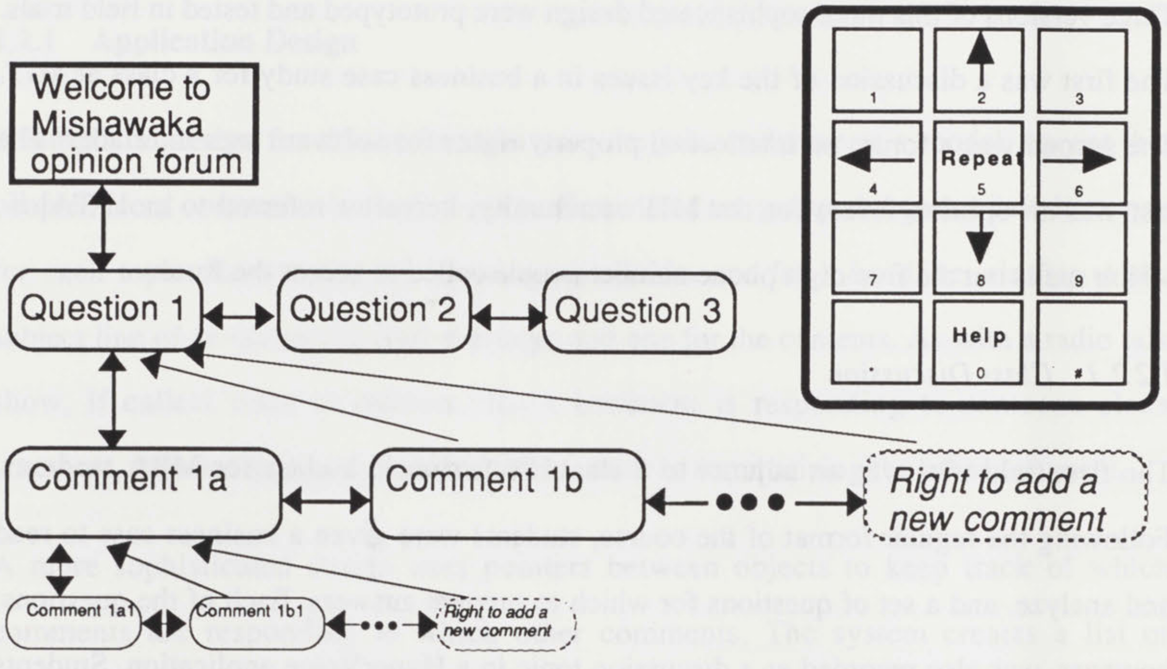


Figure 2.2 The spatial layout of comments for the class discussion field trial.

Despite some technical and user interface difficulties, a majority of the class called and a number of them recorded comments, including some responses to others' comments. One interesting result was that callers recorded at very different volumes, with non-native English speakers tending to speak more softly. Some spoke so softly that it was difficult to comprehend, and it was frequently necessary to adjust the distance of the telephone from one's ear while listening. In a normal telephone conversation, there would be ample opportunity to ask a quiet speaker to speak louder, but the delayed feedback of asynchronous communication eliminated that opportunity.

Following this field trial, I designed two possible technological fixes to the volume inequality problem, but did not implement either. Both fixes are premised on setting a desired average decibel level for the messages, but allowing variations in volume within a message, to allow for emphasis. In one case, the system would merely notify callers if a recording's average decibel level fell out of the desired range, on either the loud or the soft side. The caller could then choose to re-record or ignore the request. In the other case, the system would automatically normalize the average volume of the messages. This method would probably have been effective, but would have been difficult to implement with the hardware and software platform I used[1].

1.2.2.2 Intellectual Property

The second issue discussion was on the topic of whether intellectual property rights should apply to software user interfaces. The application followed a similar structure to the one just described, but responses were tagged as either agreeing or disagreeing with a comment. The comments were then laid out spatially in two-sided lists, with agreeing comments on the right side and disagreeing ones on the left (Figure 2.3). Thus, if a caller went down to hear responses to a comment, the system would inform the caller of how many agreeing and how many disagreeing responses there were, and the caller could then go left or right to hear one set or the other.

[1] The software supplied with the Watson card, from Natural Microsystems, used a proprietary compression format.

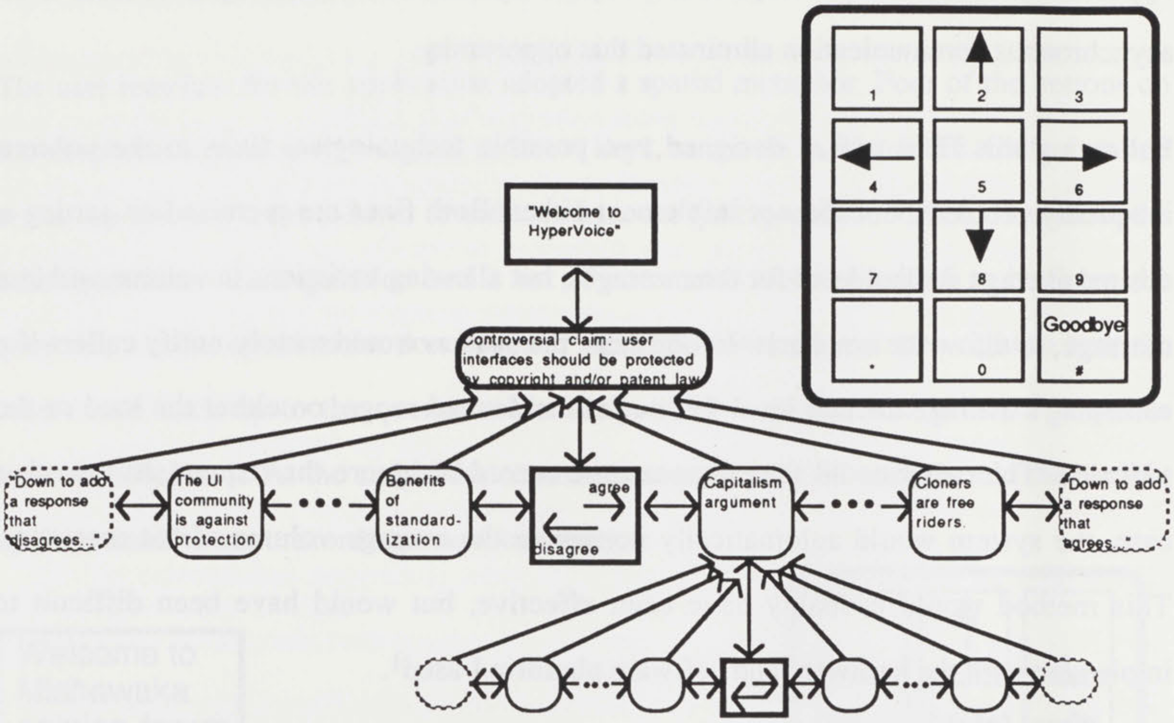

"Welcome to HyperVoice"
Controversial claim: user interfaces should be protected by copyright and/or patent law
"Down to add a response that disagrees..."
The UI community is against protection
Benefits of standard-ization
agree
disagree
Capitalism argument
Cloners are free riders.
"Down to add a response that agrees..."
1
2
3
4
5
6
7
8
9
*
0
#
Goodbye

Figure 2.3 The CHI '90 opinion forum on intellectual property for software user interfaces.

The application was seeded with eight comments, four agreeing and four disagreeing with the claim that intellectual property rights should be extended to software user interfaces. These seed comments were prepared from an article on the topic that laid out both opposing viewpoints [Samuelson 1989]. Richard Stallman, a well-known opponent of intellectual property rights on user interfaces, also called in before the conference and recorded several comments.

A telephone was set up at a booth at the Interactive Experience at the ACM CHI '90 conference on Human Factors in Computing Systems. A caller who walked up to the booth found a standard telephone on the table, and an enlarged version of Figure 2.3 on the display board. In addition, a one-page sheet described the system and the user interface. Although I sometimes staffed the booth, I carefully avoided saying anything about how to use the system until after people had tried it out.

Over the course of three days, perhaps several hundred people tried the system. A number of them made verbal comments that I jotted down and many filled out paper response surveys that were next to the phone. Unfortunately, the exact usage cannot be determined because a system crash destroyed the on-line data logs.

User Interface Results

Again, difficulties with system crashes and the user interface limited the usability of the system, but several important lessons emerged that influenced the course of this project. First, many conference attendees were not patient enough to follow the user-interface even as far as hearing one comment on the topic. It required at least three button presses, down, down and then either right or left, just to hear the first comment on the topic. There were many other distractions in the exhibit room, including some noisy exhibits that may have interfered with people attending to the prompts. This underscored the importance of a quick-start in telephone interfaces, especially when callers are unfamiliar with the application.

Second, pauses are needed in telephone interfaces, both to give callers time to make decisions, and to cue them that they should press buttons. While much of the Skip and Scan interface style depends on callers interrupting frequently, callers are not used to doing that initially, so judicious use of pauses is important. One user specifically said that she could not tell when it was "my turn to press a button."

Third, many people did not immediately understand the spatial layout of arguments. In particular, up and down for going to responses and returning from responses did not seem to be a natural mapping. Because the network is tree-structured, going up and then down does not return one to the same location. Even right and left, which have worked well in other field trials, were confused in this one, because right meant "give me another comment" if the caller was listening to agreeing responses, but meant "go back to the previous comment" if the caller was listening to disagreeing responses. Hence, the layout of right for agree and left for disagree gave the left and right cursor keys inconsistent meanings. It was not used in later field trials.

Success Factors

Finally, even those callers who succeeded in navigating through the arguments were, for the most part, unwilling to record comments. When I encouraged them to do so, many said that the topic was too controversial and their opinions not well enough formed. The lack of anonymity of recorded voice was an important factor limiting usage of other applications as well.

1.2.2.3 U-TALK

A third issue discussion was on the topic of academic honesty at MIT, a topic of frequent letters to the editor and editorials in the student newspaper during academic year 1991-1992. The application design was similar to that for the class discussion described above, but the spatial metaphor for moving up and down was abandoned. Instead, **2** moved to the list of responses about a comment and *****, acting as an escape key, exited the responses, as shown in Figure 2.5 below. As well as keys for moving forward and backward by a comment, the system provided a "smart fast forward" button, which skipped ahead to a meaningful boundary in the playback of the current message (i.e., from the headline to the contents, to the date the comment was added, and then through the prompts for what commands were available).

New comments were added at the beginning of each list rather than at the end. That meant that a repeat caller could hear the newest items in each list without wading through all the older ones. One drawback of this design is that comments in a list appear in the reverse order of their recording, which can be confusing if the comments refer to each other. An alternative to this reverse ordering would be for the computer to keep track of what comments each registered user has listened to, then begin playback of lists with the first new comments. This has not been implemented in HyperVoice.

This application was publicized through advertisements around campus and in the largest circulation student newspaper (Figure 2.4) that described the topic and suggested the overall idea of the application, but did not provide any details on how to use the system. Previous field trials suggested that such details are more confusing than helpful before using an application. In addition, business-card sized flyers (see Figure 2.5) were distributed in mailboxes in some of the dormitories on campus. The business cards did contain a diagram of the telephone keypad layout.

Success Factors: Expressiveness and Anonymity

Before the official announcement of the service, several undergraduates expressed trepidation about recording their opinions publicly. A total of 46 recordings did appear on the system, however, over 10 days. Some of the callers attempted to disguise their voices. About half of the recordings were serious or semi-serious arguments about the topic, including the recitation of one poem extolling the virtues of trust. The other half used the system as a means of self-expression unrelated to the topic. Most of the unrelated messages were recorded on April 1 (April Fool's Day) including several musical interludes, part of a Monty Python skit, heavy breathing, and one person criticizing the others for not taking the system seriously. Callers clearly exploited the expressiveness of voice and sound, though not in exactly the ways I had expected.

After two weeks, a new version was introduced. It solicited comments on several topics simultaneously, and these topics were less serious than the first. Callers were asked to say their best and worst experiences at MIT, their best and worst classes, the best and worst books they had read, and so on. Over the following six weeks, more than 100 additional messages were recorded. Many of the messages were of the fun variety rather than

What do you think about cheating?

You Call, You Listen...

U-TALK

x8-8255

Ideas

Current topic:

Academic honesty at MIT

Similar to the Lobby 7 flame sheets, U-TALK serves as a medium for discussion of controversial issues. You will be able to listen to opinions on a topic and record your own ideas. All recordings will be publicly available for other callers to listen to. System usage will also be analyzed for research purposes. Future topic suggestions or comments can be directed to Gwendolyn Lee at wendy@athena.mit.edu.

U-TALK: Just Call It!

Sponsored by: Center for Coordination Science and
Center for Educational Computing Initiatives

Figure 2.4 An ad that appeared in The Tech, an MIT student newspaper, on March 31, 1992.

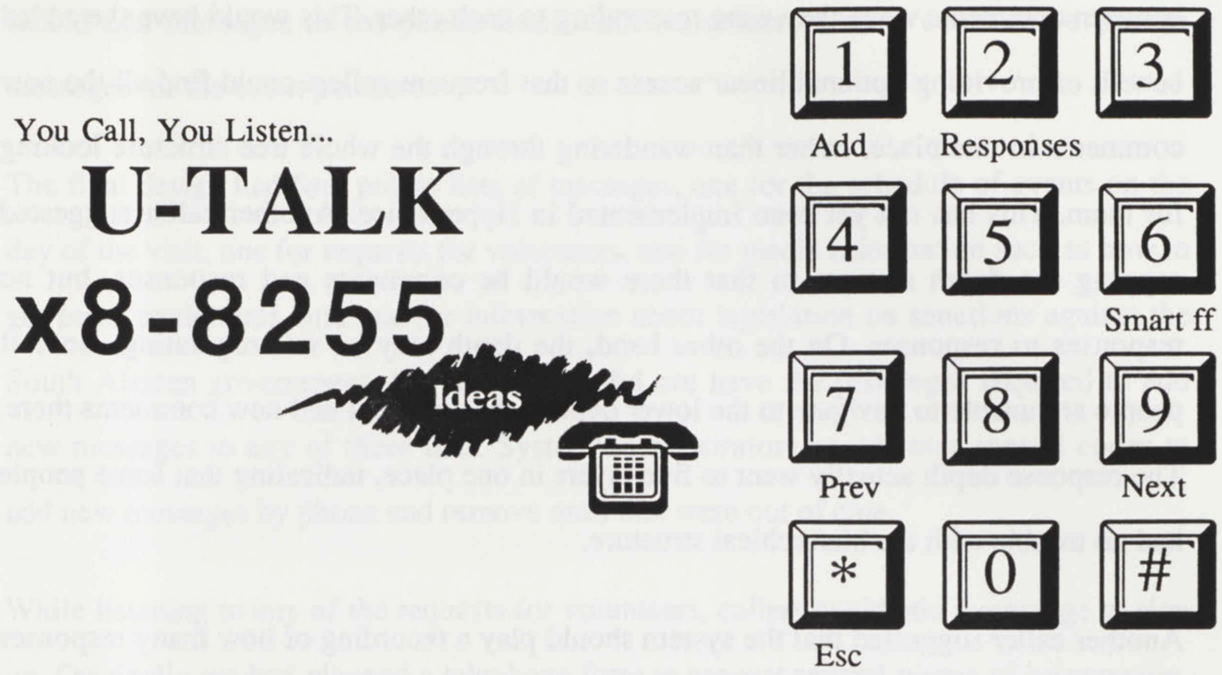


Figure 2.5 Both sides of the business cards that were distributed to student mailboxes in some dormitories and fraternities.

serious opinions. Some of the messages fell in between: in response to the question about their worst experiences at MIT, two women shouted in unison, "Everything!" It is not clear yet whether people will be willing to conduct more serious discussions via this medium as it becomes more familiar.

User Interface Results

A couple of people mentioned that it was difficult to tell where the new comments were when they called back. Despite the reverse chronological ordering within each list, in the tree as a whole the new comments were mixed with the older comments. This is what necessitated the linear master list for moderators of the opinion fora.

Several people recorded suggestions about the user interface for U-TALK in response to a question that I recorded on the system. A couple of people suggested that the tree structure of responses was confusing. One suggested using a threading mechanism that would put all the comments on a topic in one list, but with additional connections between comments when they were responding to each other. This would have the added benefit of providing optional linear access so that frequent callers could find all the new comments in one place, rather than wandering through the whole tree structure looking for them. This has not yet been implemented in HyperVoice. Another caller suggested capping the depth at two, so that there would be comments and responses, but no responses to responses. On the other hand, the depth may be self-regulating, since if people are unable to navigate to the lower depths, they will not add new comments there. The response depth actually went to five layers in one place, indicating that some people had no trouble with the hierarchical structure.

Another caller suggested that the system should play a recording of how many responses there are at the beginning of each list. As it was, callers pressed **2** to select the responses to a comment, then listened to a header for the response list before hearing the recording of how many comments were in the list. This violated the more general principle of progressive disclosure, that audio documents should place more important information before less important information.

1.3 Mandela Public Information Line

1.3.1 Application Design

In June, 1990, a HyperVoice application provided information about Nelson Mandela's visit to Boston and allowed people to leave messages to volunteer as workers at one of the events he was attending. This application was designed in cooperation with a non-programmer, the operations coordinator for the visit. I told him that our building blocks were menus, lists and messages, with access restrictions on listening to and adding messages to some lists. Most of our decisions revolved around message structure and grouping messages into lists. It was clear from the start that only the event planners would add messages to the public lists, while volunteer workers could leave private messages for the event planners.

The final design had four public lists of messages, one for the schedule of events on the day of the visit, one for requests for volunteers, one for media information such as how to get press credentials, and one for information about legislation on sanctions against the South African government. Normal callers did not have the privileges required to add new messages to any of these lists. System administrators could enter special codes to add new messages by phone and remove ones that were out of date.

While listening to any of the requests for volunteers, callers could add a message to sign up. Originally we had planned a telephone form to request several pieces of information, including name address, phone number, and social security number (requested by the U.S. State Department for security checks!) I was not yet confident of the user interface for telephone forms, however, so we opted for an unstructured, answering-machine style message. It also would have been helpful to have a separate list of volunteer signups associated with each message requesting volunteers. At that time, however, the HyperVoice application generator did not yet have facilities for automatically creating a response list each time a new request for volunteers was added (a special purpose hack had been used in the issue discussion applications). As a result, a single list of volunteer signups was kept, and callers were asked to indicate in their messages the tasks for which they were available. System administrators could enter another special code to go to the list of volunteer signups, in order to transcribe the messages.

1.3.2 Field Trial

Success Factors

The telephone number for the public information line appeared on approximately 15,000 flyers that were distributed on the streets seven to fifteen days before the visit. The system was especially effective at handling last-minute changes to event plans. A couple of times, I found that even people working at the volunteer headquarters office had information that was less recent than the information on the telephone system, because printed versions of updates were delayed and not always widely distributed. Another advantage of the phone system was that each caller could access information about any of the events without having to get information about all of the events. With mass distribution of information on paper, everyone has to be given all of the information that might interest anyone (e.g., the telephone book). The immediate publication and selective access, then, made the phone application valuable relative to other available media.

When the telephone began ringing off the hook, a second telephone line was added, which was also in constant use during the last week. Unfortunately, we did not have more equipment available to handle more telephone lines. During the last ten days, the computers answered nearly 1400 calls. Over 200 people who recorded their names and addresses were contacted by mail to inform them of where and when to arrive as volunteers.

User Interface Results and Improvements

One interesting user interface result of this field trial was the identification of the need for pause and rewind buttons whenever messages will be transcribed. The people who transcribed the names and addresses recorded by volunteers initially asked for automatic transcription of the messages, which is beyond the state of the art of speech recognition systems. The transcriptionists were still grateful, though, for the addition of a pause button to the system midway through the field trial.

1.4 MANDELA EVENT PLANNING STATUS REPORTS

1.4.1 Application Design

A status reporting application helps a group of people to keep track of subtasks that are involved in a larger joint project. One version of this application was developed for the committee that handled the operational aspects of Nelson Mandela's visit. Again, the application proceeded through several design iterations with a non-programmer, the head of the committee, using the abstractions that form the core of the HyperVoice language: menus, lists of structured messages, and access privileges.

The original design provided one list of status reports for each major task. Several design iterations went into deciding what the major tasks were. Eventually, we converged on one list for each event or other activity for which an identifiable person was responsible. Committee members would then be responsible for posting status reports about their events or tasks.

The next issue was who would have access to the status reports. Clearly, the overall operations coordinator needed access to all the status reports, but what about the other committee members? We decided that all of the status reports would be public to all of the committee members. One reason was simplicity: login procedures could be avoided. Another reason was that part of the purpose of the system was to help identify inter-dependencies among the various events and tasks. Rather than relying on the overall operations coordinator to track all those interdependencies, individual task coordinators could listen to others' status reports and help to keep track of them.

The third area of frequent re-design was the menu structure for navigating to the appropriate list of status reports. The eventual design had a top-level menu with five

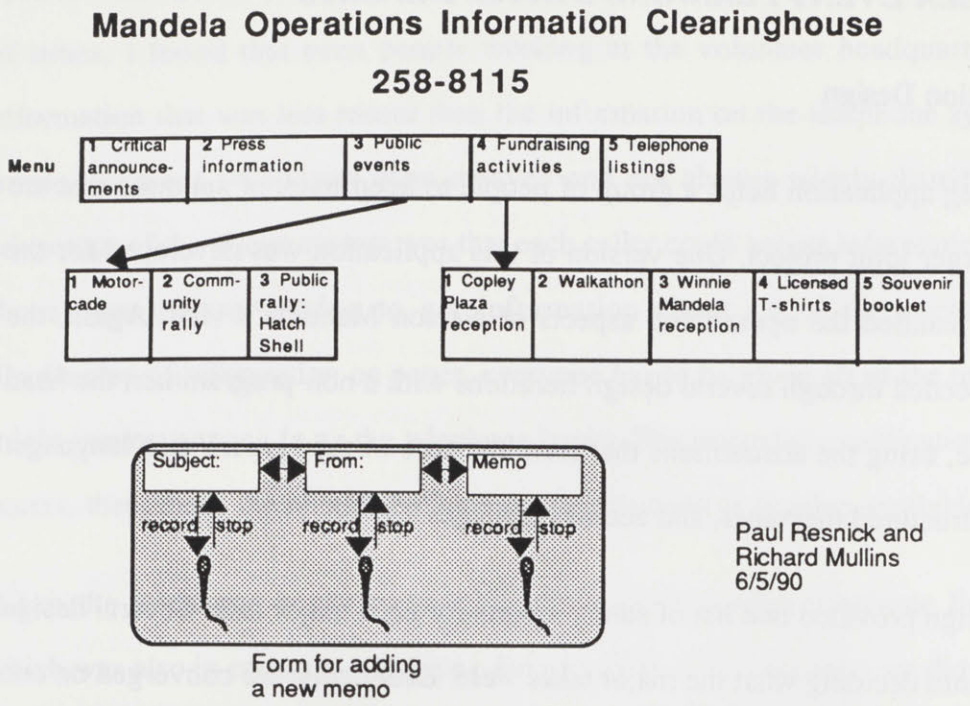


Figure 2.6. The diagram describing the Mandela status reporting application that was handed out to members of the operations committee.

options (see Figure 2.6). Two of those options led to second-level menus, which in turn led to lists of status reports (memos in the terminology of the figure.)

1.4.2 Field Trial

Success Factors

This application was not successful. In fact, only one person other than the operations coordinator called it even once. Follow-up telephone calls to the committee members indicated several reasons. First, the system was ready too late in the process. Communication and coordination structures, however inadequate, had already been established. Second, the system was not properly introduced. It was mentioned during the middle of a lengthy meeting, and committee members were asked to talk to the operations coordinator after the meeting in order to pick up instructions. Several people could not remember the system being introduced at all. Third, there was the classic incentive distribution problem pointed out by Grudin [Grudin 1989]. The work required to post status reports would have fallen on the event coordinators, while most of them perceived that the benefits would all accrue to the overall operations coordinator. Finally, it appeared to me that some of the event coordinators wanted to retain as much control over their events as possible. As a result, they did not want to open their plans to others' scrutiny.

1.5 EVENT CALENDARS

1.5.1 Application Design

The most successful long-term field trial has been an event calendar for political activists in the Boston area. The version that was field tested provided two lists, one for long-term announcements and one for upcoming events. A top-level menu asked callers to select one of those two categories. Any caller could add a new announcement to either list.

Announcements were segmented into six fields to encourage people to include important information about events, including a headline, the date and time, location, sponsoring organization, contact telephone number, and details. All the information was recorded. A moderator listened to all the new announcements and used cut and paste operations to sort them by event date and erase them after they were no longer relevant.

A more sophisticated design requires callers to type in rather than record event dates, so that the announcements can be sorted and filtered automatically. An event calendar that used typed-in dates was included in the teachers' curriculum line, described later in this chapter.

1.5.2 Field Trial

The Boston Peace and Justice Event Calendar was first publicized during the Gulf War, by passing out flyers at political events around Boston and sending flyers to organizations that were likely to sponsor events. Later, it was publicized mainly by word of mouth and occasional publication in newsletters of political organizations.

1.5.2.1 Success Factors

During the Gulf War, the emcee at a city-wide rally described it as the best source of up-to-date information about events. It was even featured in a National Public Radio story about how the student anti-war movement was well-organized, if not popular. Most of the feedback about the system has been positive, including many comments about how nice it is to hear many different voices on the system. After the Gulf War, a local activist installed the system in his home and continued to maintain and publicize it. As of January, 1992 the system had handled well over 5000 calls.

1.5.2.2 User Interface Results

This field trial confirmed two important user interface assumptions underlying this research. First, callers will learn how to interrupt and skip recordings. Analysis of the log of key presses indicates that more than 90% of callers between February 1 and April 5, 1991 (1798 calls in all) interrupted at least one announcement. Second, people can fill in telephone forms without any prior training. While a few people had trouble with the concept of semi-structured input (they recorded the same information in several fields), most of the announcements followed the intended structure quite well.

Despite the overall acceptability of the user interface to the peace events hotline, observation of people trying to add event announcements led to later modifications in the user interface for telephone forms. In particular, later prototypes have much longer pauses between the navigation prompts. Without the pauses, callers never had a period of silence in which to gather their thoughts before starting to record. Moreover, novice callers had trouble getting an overview of what actions they could take in a telephone form, because the actions were described too quickly and there were too many of them.

1.6 QUESTIONS AND ANSWERS: ANSWER GARDEN

1.6.1 Application Design

A question and answer application provides public access to the answers to frequently asked questions [Ackerman and Malone 1990]. When the expertise needed to answer questions is scarce and many requests are for similar information, this can be quite helpful. A caller navigates through a tree of diagnostic questions in order to select the topic on which he has a question. The caller then scans through the existing list of answers on that topic. If he does not find the answer he needs, he records a new question, which is routed to the expert responsible for that topic. When the expert next calls in, she listens to the question and adds a new answer to the list for that topic.

This application requires several new functions. First, access controls are needed so that each expert hears just the questions that are directed to her. Second, when a non-expert caller listens to a topic and asks to add a new question, he adds it to a list other than the one he is currently listening to. The new question is added to the private mailbox of the expert who is responsible for that topic. Third, experts can restructure the tree of diagnostic questions over the phone, if they find that callers are missing existing answers because of misleading categorizations of answers into topics. This design has not been field tested. A somewhat different design was field tested as part of the teachers' curriculum line, described below.

1.7 TEACHERS' CURRICULUM LINE

The teachers' curriculum line combined elements of several of the applications described above, including general discussion, an event calendar, and questions and answers. The application was used for two months by a group of 38 elementary school teachers in the Pittsburgh area who were implementing a new math curriculum in their classrooms. The teachers were spread throughout the city, usually with two or three in each participating school. The math curriculum was developed by one of the teachers, hereafter referred to as the head teacher, in conjunction with researchers at the University of Pittsburgh. The original goal for the application was to reduce the demands on the head teacher's time, by broadcasting her answers to commonly asked questions, and by encouraging the teachers to communicate with each other rather than only with her.

1.7.1 Application Design

The head teacher and a couple of the other researchers at the University of Pittsburgh, none of them computer programmers, participated in designing the eventual application. The design process began with a phone call to the event calendar application described in section 2.5. This gave them a sense of the possibilities. I then described most of the design choices by analogy to elements of that application.

First, I asked them to identify the structure of messages that would be recorded. I pointed out that event announcements were the only kind of messages in the event calendar, and said what the fields of event announcements were. The head teacher identified lesson plans as the key message class, as she already prepared paper calendars with a few lines of text suggesting a lesson plan for each school day in the coming month. In the initial meeting she identified several fields that lesson plans should contain. Over the course of three months, she and the other researchers on the project experimented with several structures for lesson plan messages, and wrote out scripts for several lesson plans. One problem was that some lesson plans needed to contain fields that were irrelevant in other lesson plans. As a result, they planned to make fields that could contain one of two kinds of information. Instead, I suggested that they make two separate fields, but leave out irrelevant fields from any particular lesson plan. In all HyperVoice applications, field names for empty fields are automatically omitted during playback of messages, so there is no penalty to listeners for having additional fields that are rarely used.

The designers also identified several other message classes that the system should accommodate, including questions, answers, success stories, and meeting announcements. In addition, I requested that there be a space for people to make comments or report bugs about the system, which necessitated a comment message class.

Next, the information objects had to be grouped into lists, with menus for navigating among the lists. Initially, the head teacher suggested one list of lesson plans for kindergarten and first grade and another for second and third grade. After experimenting with a prototype implementation, she realized that there should be four separate lists, one for each grade. Because different teachers were on slightly different schedules, the head teacher asked that the questions be arranged by lesson plan rather than chronologically. Each new lesson plan had an attached list of questions. In this way, a teacher who got to lesson 35 a week later than the first teacher could still hear the first teacher's questions about that lesson plan. In addition to the lesson plans and attached lists of questions, there was one list of success stories, one list of meeting announcements sorted by meeting date, and one list of comments and bug reports. These last three lists were shared among teachers from all four grades. Figure 2.7 shows the final design.

First Grade Lessons
    Lesson 25
        Question 25a
            Answer 25a1
            Answer 25a2
            ....
        Question 25b
        ...
    Lesson 26
    ....
Second Grade Lessons
Third Grade Lessons
Fourth Grade Lessons
Success Stories
Meeting Announcements
Comments and Bug Reports

Figure 2.7. The final design of the teachers' curriculum line. Indentation shows hierarchical nesting. Thus, to hear the questions about Lesson 25, a teacher would select first grade lessons, then select lesson 25.

Third, there were decisions about whether questions would be made public and about who would be allowed to answer the questions and add new lesson plans. The question and answer application described above keeps the questions private and only allows the experts to add answers. The head teacher decided instead that both the questions and answers should be public. Thus, each new lesson plan generated a list of questions and each new question had its own list of answers. The head teacher agreed to answer every question, but any teacher could also respond to any other teacher's question. On the other hand, the head teacher decided that only she and the other researchers would add sample lesson plans.

Since new questions might be attached to any of the lesson plans, and up to 40 lesson plans might be active at a time, ten for each grade level, the head teacher expressed a concern about how she would find the new questions. To help her, each new question was added to both a master list and the list for a particular lesson plan. The head teacher could access the master list while other teachers accessed questions from particular lesson plans. The master list needed a different presentation format so that the head teacher could find out which lesson plan each question was associated with. A field was added to each question indicating the associated lesson plan. This field was played back in the master list, but not when teachers accessed questions by selecting a lesson plan, as it would have been redundant there.

Finally, we decided to include a login operation. This served several purposes. First, it meant that only teachers participating in the curriculum project would have access to the system. Second, the name of the person who recorded each question, answer, comment, or meeting announcement was automatically attached to each recording. Third, it made it possible for me to track usage of the system by each teacher. Teachers who did not want their names attached to messages or who did not want their usage tracked were all told a common user id, 1111, to be used for anonymous login. The head teacher was able to add and remove eligible users over the phone.

This design process, then, illustrates the abstractions used in discussing alternative designs. Discussions centered around questions of what the object types would be, how those objects should be collected into lists, who should be given read and write access to different lists, and whether or not to include a login procedure. These abstractions will form the core of the HyperVoice programming language in chapter 4.

1.7.2 Field Trial

The actual usage differed quite a bit from the planned usage. Almost half the teachers never called. Those who did call listened to the lesson plans that the head teacher posted and listened to the meeting announcements. At least two teachers were observed using the sample lesson plans in their classrooms. A few of them recorded success stories or congratulations to other teachers on wedding plans, but only two questions were recorded in two months. This application, then, gave the head teacher the additional task of recording lesson plans without reducing the number of repeat questions that she handled. After two months, she stopped recording new lesson plans.

1.7.2.1 Success Factors

A number of plausible explanations can be made of the usage patterns, and the truth probably lies in some combination of them. First, the system was introduced six weeks into the school year, after communication patterns were already set. While the 38 teachers were distributed among many schools, there were usually two or three in each school, and the whole group met for one day a month, so that they may not have felt the need for greater communication. Second, teachers are not used to getting very much feedback from peers, so that asking a question about teaching puts them in an unfamiliar and vulnerable position. Thus, the lack of anonymity of recorded voice appeared to be an important factor. Third, at approximately ten minutes each, the recorded lesson plans were too long to remember without transcription, which would have been tedious. Not only did this reduce the utility of the lesson plans, but it may have reduced the willingness of teachers to ask questions. They may have been worried about asking questions without listening to the whole lesson plan, for fear that the question was already answered, but not interested in listening to a whole lesson plan. Finally, the head teacher was initially uncomfortable recording the lesson plans, so she read them verbatim from a prepared script. That reduced the expressiveness of her voice, thus squandering a potential benefit of the medium.

1.7.2.2 User Interface Results

The field trial also uncovered two user interface problems. First, as already mentioned, some of the recordings were too long. Second, the head teacher and other frequent callers complained that they had to press too many buttons to get to some of the information. This application used audio menus with positional selection. To select the fifth option in the menu, callers had to advance five times and then press the select button. The head teacher called every day and checked for new success stories and comments, the fifth and seventh items in the main menu. Hence, she had to press the advance key a lot of times to perform those routine operations. In addition, new success stories and comments were added at the ends of those lists, so that the head teacher had to skip over all of the old ones every day in order to check for new ones.

1.8 MATCHING

One generic coordination task is to match people or things based on information about them. An example of such a process would be a dating service, where people are matched based on both symbolic information that they provide (gender, age, sexual orientation) and on unstructured information (recordings they make.) In fact, this is one commercial application of telephone bulletin boards that has already taken off, usually in conjunction with printed personal ads in newspapers.

Personal ads, however, are not the only matching services that a telephone bulletin board can provide. They can be used for classified ads more generally, to match buyers and sellers. Below I present a prototype application for matching conference attendees who want to share a hotel room during the conference.

1.8.1 Application Design

A caller makes two menu selections, to indicate gender and whether or not the caller smokes. It is assumed that callers who do not match on these characteristics will be incompatible roommates, so that it will be better for them not even to hear listings that do not match on these two characteristics. Suppose that the caller is a male non-smoker. The caller then browses through a list of messages from other male non-smokers. If one or more people seem compatible, the caller jots down their phone numbers or email addresses and contacts them. If not, he adds a new message to the list, in the hopes that some future caller will contact him.

Each listing includes separate fields for arrival and departure dates and hotel preference. In addition, there is a field in which the caller can record any other information that he thinks is relevant, such as how loudly he snores. Arrival and departure dates are typed in over the phone rather than recorded. Typing in the dates makes it possible to sort the listings in order of arrival date, which may help other callers to find appropriate listings more quickly.

Once people have found matches, they will want to remove their listings. A sophisticated version of this application would have callers enter a security code as part of the listing, then enter the same code again to remove it. The HyperVoice application generator does not yet allow specification of this functionality. A simpler design that HyperVoice can implement requires a system administrator to remove listings at the request of callers, after checking to make sure that the original voice sounds the same as that of the person requesting its removal.

This application has not yet been field tested.

1.9 TASK TRACKING

The simple status reporting application that was developed for the Mandela operations planning committee misses opportunities that an on-line database can provide. There are a number of different ways that someone might want to access the status reports. Sometimes, the status reports should be sorted by people responsible for completing them. At other times, it may be helpful to get a list of tasks by their priority, or by their due dates. Presentations of lists can provide pre-defined queries, so that the tasks and status reports can be accessed in the way most appropriate to the task at hand. Ad hoc queries could make this status reporting application even more valuable. The future research section at the end of Chapter 4 discusses ideas for implementing ad hoc queries.

1.9.1 Application Design

This application includes one master list of status reports and one master list of tasks. Project members only access the information about tasks for which they are responsible, while project coordinators can access all of the information. More than one person can be responsible for a particular task.

Consider a single project member, John Doe. After John logs in, the system presents him with a menu of options. That menu includes hearing all tasks for which he is responsible, sorted either 1) by date due or 2) by priority, 3) all status reports that he has recorded, or 4) all status reports for tasks for which he is responsible. If John selects either of the first two options, he hears a list of tasks. If John then selects one of those tasks, he hears all of the status reports associated with the task. This menu structure is summarized in Figure 2.8.

John can also add a new task. In that case, a new list of status reports for that task will automatically be created. John will type in a due date and a priority value using touch-tones. In addition, John will select the people responsible for the task from a menu (a picklist) of all the people on the project. The selected person objects are linked in as values to a field of the task object that keeps track of who is responsible for the task. The date due, priority, and person responsible fields, containing symbolic information rather than recordings, are what make possible the various sortings of the tasks and status reports. The headline and description fields, on the other hand, contain informal descriptions of exactly what the tasks are.

Once John has selected a task, he can also add a new status report. The system fills in a

- **John Doe's tasks sorted by date due**
  - *Task 1*
    - Status report 1a (recorded by John)
    - Status report 1b (recorded by someone else)
  - *Task 2*
    - Status report 2a (recorded by someone else)
    - Status report 2b (recorded by John)
  - *Task 3*
- **John Doe's tasks sorted by priority**
  - *Task 2*
    - Status report 2a (recorded by someone else)
    - Status report 2b (recorded by John)
  - *Task 1*
    - Status report 1a
    - Status report 1b
  - *Task 3*
- **All Status reports recorded by John Doe**
  - *Status report 1a (recorded by John)*
  - *Status report 2b (recorded by John)*
  - *Status report x: (recorded by John about a task he is not responsible for)*
- **All Status reports for tasks John Doe is responsible for**
  - *Status report 1a (recorded by John)*
  - *Status report 1b (recorded by someone else)*
  - *Status report 2a (recorded by someone else)*
  - *Status report 2b (recorded by John)*

Figure 2.8 The information network presented to John Doe. Indentation indicates hierarchical nesting, so that the top-level menu consists of the four options in bold.

field with a pointer to the object representing John, in order to note that he is the one who recorded the status report. That is what facilitates the top-level menu option of selecting all the status reports that John has recorded. The system also fills in a pointer to the task object to which this status report is attached. This facilitates the top-level menu option of hearing all the status reports for tasks for which John is responsible: a filter can look at each status report to find the associated task, then look in the task object to see if John is one of the people responsible for it.

The project leader may choose to access the tasks and status reports by people responsible or may choose to access them all together. The Project Leader has four initial options. The first three are similar to the first three options seen by project members: tasks by date or by priority, and status reports by author. The difference is that no filters are specified, so that all tasks or status reports are selected rather than a subset being selected. The project leader can also access information relevant to any one member of the project, by selecting that person from a menu of all the people on the project. After selecting a person, the project leader has all the same options that person has immediately after logging in.

This application design has not been field tested.

2 FUNCTIONAL REQUIREMENTS

All of the application designs build on list, menu, form, and login procedure abstractions. Reflecting different functional requirements for the applications, however, there is considerable variability among instantiations of those abstractions. As described below, instantiations can vary in the arrangement of information in lists, in who adds information where, and in maintenance operations available to system administrators. Table 2.9 summarizes the functions needed in the most complicated applications described in this chapter.

2.1 INFORMATION ARRANGEMENT

The objects in a list can be sorted based on the contents of the objects (e.g., the dates of the events). The same objects can appear in different lists, sorted in different ways or with different fields of each object played back.

2.2 INFORMATION ADDITION

Several features can help to structure the addition of information objects in ways that aid future callers.

Restricted access. Only callers with high enough privileges may initiate addition of new information from some lists. This can insure the quality of information in those lists.

Addition to other lists. Sometimes, the new information object should be added to other lists in addition to, or instead of the current one. This makes it possible for callers to add information objects without knowing all of the lists to which they should be added.

Creation of response lists. In some applications, the addition of a new information object automatically generates a new, empty list. This creates a location where future callers can add related information.

Needed Functions	U-TALK	Curriculum Line	Answer Garden	Generic task tracker
Information Arrangement				
Sort on contents of objects				√
Multiple presentations of same object	√	√		√
Adding Information Objects				
to current list	√	√		√
to other lists	√	√	√	√
Automatic creation of response lists	√	√	√	√
Access controls	√	√	√	√
Maintenance Operations				
Master list for moderator	√	√		
Move, delete information	√	√	√	√
Add new menus and lists			√	
Add new users		√	√	

Table 2.9 The functions required in the more complex application designs.

2.3 Maintenance Operations

A system administrator may need the following features that are not available to normal callers:

- Access to master lists that arrange information objects chronologically.
- Moving and deleting information objects.
- Creating new menus and lists. As more information is added, the system administrator may need to rearrange the information space.
- Registering new users. If there is a login procedure, the system administrator may want to register new users over the phone.

3 USER INTERFACE INSIGHTS

The pilot tests and field trials identified a number of desirable user interface properties, summarized below, often because their absence led to difficulties in particular field trials. Chapter 3 presents a more complete analysis, from first principles, of the interaction styles and audio document properties that will be best for users.

- Quick start
    - No initial tutorial; navigation prompts in context.
- Pauses indicate when to press buttons.
- Prompts not too fast.
- Short recordings.
- Progressive disclosure.
- Normalized volumes.
- Pause and rewind buttons.
- Metaphor that predicts available actions

- Lists and forms are helpful. Keypad layout with **next** to the right of **previous** is helpful.
    - Two-sided list is confusing.
    - CD jukebox is confusing.
    - Hierarchy is helpful for some people, confusing to others. Keypad layout with **go to parent** above **go to child** adds to the help or to the confusion.
- Available actions fixed until next button press.
- Few keystrokes to get to frequently accessed information.
- Find changes since last call quickly.

4 SUCCESS FACTORS

Previous research on CSCW applications and analysis of the differences between text and voice suggest a number of context variables that will influence the value of HyperVoice applications. Below I discuss both positive and negative factors, called 'green flags' and 'red flags' respectively. Observations from both the successful and unsuccessful field trials are consistent with these success factors.

4.1 GREEN FLAGS

Time-critical information. Once entered over the phone, information is immediately available to other callers. For applications with time-critical information, HyperVoice will be more useful than communication systems such as mass mailings that have longer delays to publication.

Need for access from home or while traveling. Applications that benefit from entry or retrieval of information from remote sites will be more likely to succeed, since competing technologies cannot match the telephone's widespread accessibility.

Need for expressiveness of voice. Voice is more expressive than text; through tone, pitch, and speed, a speaker can convey much information not conveyed by a text transcription. Studies of teleconferencing indicate that voice is the single most important channel to include for collaborative tasks [Ochsman and Chapanis 1974] and that voice is a better annotation medium for the more complex, controversial, and social aspects of collaborative tasks [Chalfonte, et al. 1991].

Users have weak composition, keyboarding, or reading skills. Unlike text-based systems, phone-based systems do not require these skills.

Opportunity to create a 'honeymoon period'. Most communication systems have increasing returns to adoption: the utility of the system to each user increases as the number of other users increases [Arthur 1987, Markus 1990]. To achieve critical mass, these systems need an initial honeymoon period in which users adopt based on expectations of future value, when others have adopted, rather than on the current utility of the system [Fichman and Kemerer 1992]. Any opportunity to create a honeymoon period, through sponsorship by a powerful person or through a big splash introduction of the system, will improve its chances of success.

4.2 RED FLAGS

Well-entrenched communication patterns. Changes from existing patterns are disruptive and people may not perceive the opportunity for better communication patterns.

Poor distribution of costs and benefits. The distribution of incentives is a well-known problem for CSCW systems [Grudin 1989]. While the benefits to the group as a whole may outweigh the costs, the benefits may accrue to some individuals and the costs to others, who may then refuse to participate.

Need for anonymity. A person's voice is more easily identified than a person's writing style. Research indicates that the anonymity of bulletin boards can improve participation from shy people [Chaiklin and Schrum 1990] and that anonymous suggestions can enhance brainstorming sessions [Nunamaker, et al. 1991]. Using digital signal processing algorithms, it is possible to disguise voices, but only at the cost of losing the expressiveness and some of the intelligibility.

Naturally textual information. If during a face to face meeting, when all media are available, the information would be communicated as text or graphics, then voice will be a poor medium to carry that information in a computer-mediated communication system.

Need to scan large information chunks. If there is no way to break information chunks into small pieces, then it will take longer to listen to a message than for a good reader to read or scan a written version of it. If, on the other hand, the information divides naturally into small, meaningful segments, then a consistent set of telephone buttons may allow callers to accomplish the fast changes of attention that eye gaze shifts accomplish in visual scanning. This is the idea behind the Skip and Scan interface style, as described in the next chapter.

Need to remember large information chunks. The presentation of information by phone is ephemeral. If callers need to take information with them, they will have to engage in the tedious process of transcription.

5 RELATED RESEARCH

The first published use of structured voice messages was the PhoneSlave project. The PhoneSlave, acting as an answering machine, conducted conversations with callers to elicit the several pieces of information it considered essential to good phone messages [Schmandt and Arons 1984]. The system asked each caller a series of questions ("Who's calling please", "What is this in reference to?", "At what number can he reach you?", etc.) After playing a question, it recorded whatever the caller said, until a long pause was detected, then went on to the next question. This thesis generalizes the structured message idea to information objects other than personal phone messages and then exploits that structure in the design of more complex applications.

The fundamental idea of supporting a range of cooperative work activities through clever processing and presentation of semi-structured information objects is not new to this thesis. It was best stated by Malone et al [Malone, et al. 1988]. That paper, discussing the

Information Lens system, presented designs for a number of applications based on processing structured electronic mail messages. In the Object Lens and OVAL systems, this idea was extended to include information objects other than messages [Lai, et al. 1988, Malone, et al. 1992], which enables data modeling of the kind found in the status reporting application design. The application designs in this chapter apply the idea to a new medium: phone-based applications. The field trials show that such designs can be successfully implemented.

6 CONCLUSION

The most important result in this chapter is a validation that it is possible to create usable and useful telephone bulletin boards. The set of application designs illustrate that telephone bulletin boards can support five generic communication and coordination tasks, summarized in Table 2.10. In field trials of issue discussion and announcement applications, callers were able and willing to add structured messages and other callers were interested in hearing what those callers had to say. One of the applications, the Boston Peace and Justice Event hotline, was used beyond its initial trial phase, and is now administered outside of MIT.

Communication or Coordination Task	HyperVoice application
Issue Discussion	Class discussion Intellectual Property opinion forum U-TALK Teachers' curriculum line
Matching	Roommate finder
Announcements	Mandela public Peace and justice event hotline Teachers' curriculum line
Questions and Answers	Answer Garden Teachers' curriculum line
Task Tracking	Mandela event planning status reports Generic status reporter

Table 2.10. Five generic communication and coordination tasks, and the application designs that supported them.

The differential success rates of the field trials suggested a set of task and context properties that will influence the success of telephone bulletin board applications. Applications are more likely to be adopted when they contain information that is time-critical, when users benefit from remote access, when the expressiveness of voice carries valuable information, when users have weak writing and reading skills, and when there is an opportunity to create a 'honeymoon period'. Applications are less likely to succeed when there are well-entrenched communication patterns, costs and benefits are poorly distributed, there is a need for anonymity, and there is a need to scan or remember large information chunks.

The application designs confirm that expandable lists, menus, forms, and login procedures are sufficient to describe a wide range of applications. The fact that non-programmers participated in designing three of the applications indicates that these abstractions are accessible to non-programmers. Taken together, the application designs determine the functional requirements for the HyperVoice application generator of Chapter 4.

Pilot tests and field trials led to iterative refinements in the user interfaces that the HyperVoice application generator creates. They inspired the laboratory experiments with different menu styles reported in Chapter 5. They also inspired a more general analysis from first principles that identifies and categorizes interaction style properties and document properties that make it easy to retrieve and add information in audio documents and, more generally, hypermedia systems. The next chapter presents that more general analysis.

The field trials [illegible] suggested [illegible]

[illegible] when there are well-entrenched communication patterns, costs and benefits are evenly distributed, there is a need for anonymity, and there is a need to scan or remember large information chunks.

The application designs confirm that expandable lists, menus, forms, and logic procedures are sufficient to describe a wide range of applications. The fact that non-programmers participated in designing three of the applications indicates that these abstractions are accessible to non-programmers. Taken together, the application designs determine the functional requirements for the HyperVoice application generator of Chapter 4.

Pilot tests and field trials led to iterative refinements in the user interfaces that the HyperVoice application generator creates. They inspired the laboratory experiments with different menu styles reported in Chapter 5. They also inspired a more general analysis from first principles that identifies and categorizes audio style parameters and document properties that make it easy to retrieve and add information in audio documents and, more generally, hypermedia systems. The next chapter presents that more general analysis.

3 Skip and Scan

Text is linear, but many texts are not read in a linear manner. Instead, eye gaze shifts rapidly on a single page and readers open documents to pages in the middle. Writing styles and typographic conventions have developed in ways that facilitate non-linear access. For example, headlines in newspapers allow readers to quickly find the beginnings of stories and white space between paragraphs allows readers to quickly find the beginnings of new topics. Good readers have years of practice in exploiting the scanning abilities that these visual cues give.

Speech is also linear but with digital storage and random access, speech no longer needs to be listened to in a linear manner. With playback controls, listeners can rapidly shift attention between speech segments. Unfortunately, speaking and audio document styles have not yet developed in ways that take full advantage of this non-linear access. Moreover, no one has had much practice listening to audio documents with random access.

Skip and Scan is a slogan around which audio document styles and listening skills can develop. The ideal behavior is that a listener should frequently change what he hears (skip), without spending a lot of time deciding what skips to make, and in that manner scan through the contents of an audio document. By analogy, consider the slogan *Direct Manipulation* [Hutchins, et al. 1986, Shneiderman 1983]. It presents an ideal for how people should interact with information in screen-based interfaces. They should get the feeling of directly manipulating information objects, even though those manipulations are mediated by mouse operations. Similarly, Skip and Scan audio documents should give listeners the feeling that they are scanning the documents' contents; they should not be conscious of the computer that is mediating the interaction. To accommodate multiple contributors, documents must include expandable structures that preserve a listener's ability to scan even as contributors add new information.

While Skip and Scan describes ideal listener behavior, designers need to informally evaluate designs for audio documents even before implementing them and conducting tests with users. To aid them, a set of document properties serve as style guidelines; documents that satisfy those properties will afford listeners the opportunity to skip and scan.

Lists of structured objects that contain links to other lists are building blocks for audio documents that satisfy the guidelines, but they are not the only good building blocks. The guidelines in this chapter will prove useful in evaluating other potential building blocks as well. Moreover, the guidelines apply far beyond telephone interfaces, to audio documents controlled through speech recognition and to 'keyhole' interfaces such as small LCDs. Even a large display can offer only a small keyhole on even larger information spaces; the guidelines in the chapter will help design and evaluate large hyperdocuments containing text, graphics, and video as well as audio.

1 DESIRED BEHAVIORS

The ideal audio document should support several types of behavior. An expert listener should be able to scan it, either to find a particular piece of information, or to get an overview of the document contents. At least in the near term, before listener skills and familiarity with audio document genres develop, listeners will not be familiar with the conventions of audio documents. A listener who does not know how to scan should still be able to navigate through the document, although not as quickly or fluently as someone who scans it. Moreover, while listening to a document, novices should naturally acquire the skills necessary for expert performance.

1.1 SCANNING

Curiously, the verb *scan* has nearly reversed meanings in the last hundred years. It used to mean analyzing and marking a text closely [Murray and Burchfield 1933]. More recently, scanning a visual document has come to mean looking at it quickly in order to find a particular item of interest or to get an overview of its contents and develop a mental map of where to find the parts of it that are most interesting. Scanning is valuable because different readers are interested in different parts of documents, or prefer to read parts in different orders. If the goals and background knowledge of all readers were the same and could be predicted in advance, then the reader control that scanning allows would not be necessary: documents could present information linearly in exactly the order that readers needed it.

Visual scanning is accomplished by peripheral vision, reading quickly, and rapid eye gaze shifts, as summarized in the second column of Figure 3.1. In the audio realm, non-speech audio cues [Gaver 1986] and presentation of multiple audio signals in a spatially localized way [Wenzel 1991] may accomplish some of the effects that peripheral vision gives. Digital signal processing algorithms can speed up the playback of recorded speech to roughly twice the speed at which it was recorded, without harming intelligibility significantly [Arons 1992]. The Skip and Scan slogan for audio documents focuses on the last of the three capabilities that facilitate visual scanning, namely the rapid change of focus of attention between parts of the document.

Effective scanning involves ignoring most of the text in a document without taking a long time to execute the shifts of attention that make that possible. Rhetorical and typographical conventions help readers to ignore irrelevant text. For example, the typographical convention of white space between paragraphs together with the rhetorical convention of topic sentences allows readers to read at most the topic sentence of any

	Visual Documents	Audio Documents
Peripheral Awareness	peripheral vision of spatial layout	background non-speech cues; spatially localized audio
Rapid Perception	read quickly	accelerated playback
Rapid Change in Focus of Attention	shift gaze	Skip and Scan

Figure 3. 1 The capabilities that contribute to scanning.

irrelevant paragraph. Shifts of attention to different text on a page or screen display are executed quickly through shifts in eye gaze. Shifts of attention between pages, however, take much longer, and the time increases with the number of intervening pages. For this reason, style guidelines usually suggest that figures be interspersed in the text of an article rather than grouped at the end.

Effective scanning in audio documents requires an analogous interaction style. Listeners should hear only a small percentage of the total sounds in a document without spending a lot of time on the navigation operations that make it possible to hear just the right sounds.

1.1.1 Hear small percentage of document

Several interaction behaviors will contribute to reducing the amount of speech played back. Callers should:

- avoid whole sections of the document that are irrelevant.
- not wait through the playback of uninteresting parts in order to get to interesting parts.

• know when the rest of a recording will be uninteresting and skip it.

In short, listeners should avoid uninteresting recordings wherever possible. Barring that, they should quickly identify them as uninteresting and interrupt them.

1.1.2 Fast Navigation

"Unconscious" navigation

Listeners' cognitive attention should be devoted primarily to the contents of what is played back, and not to thinking about how to navigate through the document. Unlike visual documents, where the output channel remains constant but eye gaze shifts, shifts of attention in audio documents require a change in what sounds are played on the output device. The tool for controlling those changes should be ready-to-hand (Heidegger as discussed in [Dreyfus 1991]), in the sense that a skilled user should be conscious of the tool only in special circumstances where normal use of the tool fails. By analogy, consider skilled users of emacs, or any mouseless text editor. In normal operations of editing a document, they are not conscious of what buttons they press. Only during non-routine operations such as defining macros are they conscious of using the keyboard to execute commands.

In audio documents, callers can navigate "unconsciously" when they:

• don't need to hear any navigation prompts.

• don't need to think long about the best next button to press.

Short keystroke sequences

With paper documents, shifts of attention between pages take much longer than shifts of attention within a page. In audio documents, all shifts of attention come from button presses. Shifts that require fewer button presses, then, will be executed more quickly.

1.2 ACCOMMODATING NOVICE USERS

At least in the early stages of the evolution of audio documents, they will have to accommodate listeners who do not yet know how to scan them. Novice listeners will not know what buttons to press and tend not to interrupt playback of sounds [Engelbeck and Roberts 1990]. The interaction style for novices, then, needs to be more conversational:

- Document always suggests what buttons to press.
- Document occasionally pauses to wait for button presses.

Novice callers may need some time to think about which buttons to press. Therefore:

- Document suggests buttons slowly.
- A listener should take as long as necessary to make a choice without losing the ability to accept or reject a suggested action. In particular, there should not be time-dependent actions that are available only for a short period of time. Recall that the joke collector application did not allow this kind of interaction, because callers could follow a link from one "track" to another "disc" only while that track was playing.

Novice listeners will also not have practiced audio navigation skills, and so will be conscious of all of the navigation choices that they make. If the choices are too frequent, the listeners may be unable to focus their attention on the contents of documents. This suggests that:

- Listeners should not have to make too many navigation decisions.

1.2.1 Quick Start

It is important that a document reveal its contents early in the interaction, even to novices. Some listeners will have made enough of a commitment to the technology or to the document that they will endure a tutorial before getting to the document contents, but it is better to teach callers to use buttons as part of navigating through the document. This is especially true with telephone applications, where the caller may be unfamiliar with the benefits of the application as well as unfamiliar with the technology. Any attempt to teach the caller about how to navigate in the first few seconds of the call has to be subordinate to selling the application concept.

1.2.2 Learning to Scan

There should also be a natural transition from novice to expert behavior. In the process of listening to a document without scanning, then, a novice caller should:

- Learn the document structure.
- Practice pressing the buttons that are used in scanning.

1.3 TRADEOFFS

Note that there is a tradeoff between supporting non-scanning behavior and teaching listeners how to scan. While a novice who just wants information from one document may be best served by hearing the document without pressing buttons, practice at pressing buttons will help novices to make the transition from novice to expert behavior.

There need not be a tradeoff, however, between supporting novices and experts. In any speech segment, all of the recorded information relevant to novices can be placed after the information that is of interest to all callers, novice and expert. The expert callers can execute skip operations to other nodes before hearing the information that is only useful to novices.

1.4 SUMMARY

Readers scan visual documents, in part, by quickly shifting their eye gaze. This allows readers to ignore large portions of a document that are likely to be irrelevant to their needs, without spending a lot of time deciding which parts to ignore. By analogy, audio documents need to allow listeners to hear small percentages of documents without spending a lot of time getting to the relevant parts. Since many listeners will not initially know how to scan, audio documents should allow novices to hear some useful information without scanning. Moreover, in the process of listening to documents, novices should acquire the skills necessary to scan them.

2 THE ELEMENTS OF STYLE FOR AUDIO DOCUMENTS

This section translates the desirable interactions described above into desirable properties of the documents themselves. It describes how to recognize an audio document that will afford these interactions to typical listeners. The translation is valuable because it enables evaluation of audio documents without conducting behavioral tests with users.

All of the desirable properties are stated in terms of a network, or state-machine representation of audio documents. That is, a document is represented as a collection of nodes, which contain speech segments (*sound-bytes*), and directional links, or transitions, that can be followed from node to node. Each of the links is labeled by the button press that causes the link to be followed. In addition, there can be timeout transitions between nodes: a timeout transition is executed after playback of the entire contents of a node, if no other button is pressed first.

The desired properties constrain the contents of the nodes and the topology of the network, the links between nodes. The network representation is convenient both because it allows a succinct statement of the desired properties, and because it is already widely used to describe audio documents. For example, the HyperVoice application interpreter automatically generates network representations and several commercial software toolkits expect programmers to specify network representations directly.

2.1 HEARING A SMALL PERCENTAGE OF THE DOCUMENT

The organization of the nodes' contents will determine whether a listener can hear the interesting information while avoiding the uninteresting information. Overall, listeners will use the contents of some nodes to determine which other nodes to listen to, and the beginnings of individual nodes to decide whether to listen to the later parts of those nodes.

2.1.1 Small Nodes

The first desirable property is small nodes. A node is the unit of speech playback inside which callers do not have any semantically meaningful skip actions available (timed fast-forward and rewind actions may be available, though). If nodes contain long speech segments, then scanning will be defeated. In particular, listeners who are interested in the later portions of recordings will either have to listen to the beginning part of the recording, or they will have to guess how far to move with timed fast-forward and rewind buttons, which may take a long time. Consider, for example, how hard it is to find the beginning of a particular scene on a videotape using a VCR's fast-forward and rewind buttons.

2.1.2 One Topic per Node

Strunk and White advise writers to include just one topic per paragraph[Strunk and White 1979].

> Ordinarily, however, a subject requires division into topics, each of which should be dealt with in a paragraph. The object of treating each topic in a paragraph by itself is, of course, to aid the reader. The beginning of each paragraph is a signal to him that a new step in the development of the subject has been reached. (p. 16)

Analogously, each node in an audio document should deal with only one identifiable topic. A listener may be interested in but one of two topics but will be unable to skip over the other one if it appears after the first one in a node. In addition, if nodes have more than one topic, listeners cannot know whether it is safe to skip the remainder of a node after deciding that the first topic is irrelevant.

2.1.3 Progressive Disclosure

The contents of each node should follow the principle of progressive disclosure: the most important information should come first. If this rule is violated and the most important information comes later in the node, then listeners may have to wait through information that does not interest them in order to hear the important part. This principle is analogous to the admonition to writers to begin paragraphs with topic sentences. Perhaps it is even more important for audio documents, because the penalty is greater when the first words fail to summarize the rest of the contents: the listener may have to listen to the entire node, while a reader can scan ahead visually.

Consider that answering machine and voice mail messages often fail to follow the rules of one topic per node and progressive disclosure. Recipients of messages often keep listening just to be sure that there is nothing important at the end of a message, or because callers do not say their phone numbers until the ends of messages. Sometimes, the beginning of the message will even announce that there are two unrelated points. But when someone listens to the message, there is no way to skip ahead to the second point, if the first is explained in too much detail. This suggests a hypothesis that frequent voice mail users will learn the art of progressive disclosure in their messages and will develop the habit of recording two separate messages for the same person when there are two separate topics to discuss. This would be an interesting hypothesis to test empirically.

2.1.4 Summary Nodes

Some nodes should summarize the contents of collections of other nodes. Listeners can hear summary nodes then skip entire collections that are not relevant. By analogy, in an outline document, each header summarizes the contents that are indented below it.

Readers can skip over whole sections of the outline based on the contents of the top-level headers.

2.1.5 Short paths

The desire for short keystroke sequences to accomplish navigation actions translates directly into the graph property of short paths between nodes. Of course, this does not require that there be a short path between every pair of nodes, but rather that there be a short path from each node to every other node that is often accessed right after listening to it. For example, there should be a short path between a summary node and a node that gives details about that topic.

2.2 FAST NAVIGATION

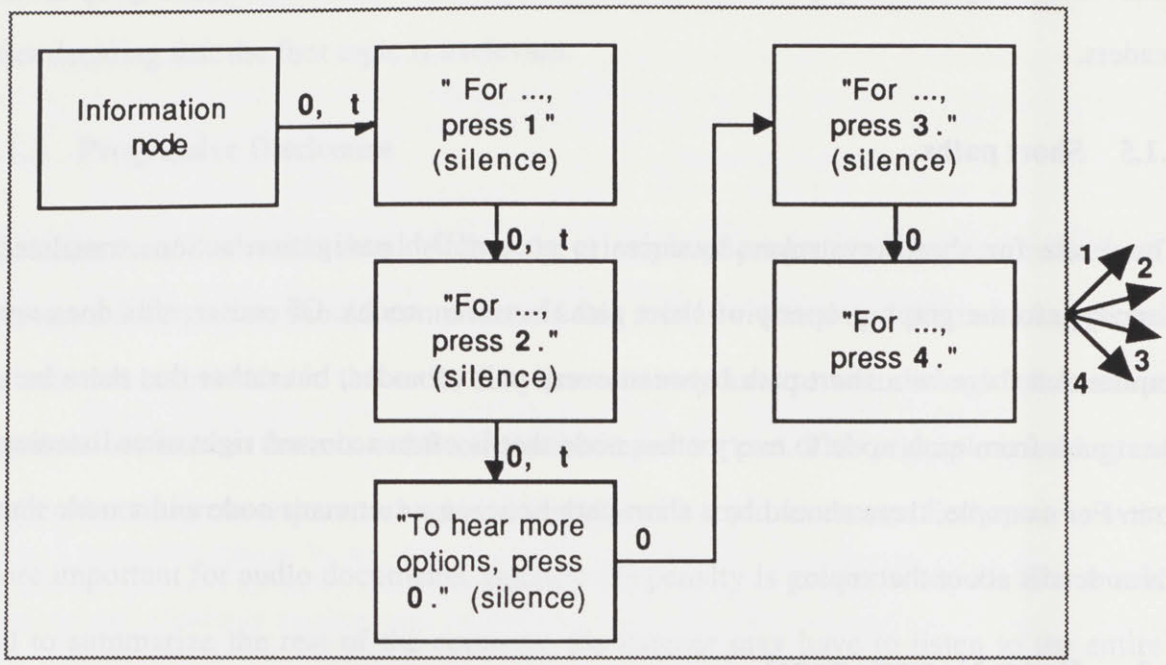
Listeners can navigate when they know what transitions are available and where they go. They can navigate quickly when they can predict the available transitions without hearing prompts for them. Regularities, orderings, and metaphors make that possible.

2.2.1 Navigation Prompts

Navigation prompts help novices to navigate through a document. These prompts, however, should not interfere with normal scanning. The solution is that each information node in the document can have one or more attached navigation prompt nodes, as illustrated in Figure 3.2 and described below

2.2.1.1 Pauses after navigation prompts

A brief silence should follow each navigation prompt. This serves two purposes. First, the pauses indicates when to press buttons for those who are uncomfortable interrupting the spoken voice. Second, they give listeners the chance to decide on an option before forcing



Information node
0, t
" For ..., press 1 " (silence)
0, t
"For ..., press 2 ." (silence)
0, t
"To hear more options, press 0 ." (silence)
0
"For ..., press 3 ." (silence)
0
"For ..., press 4 ."
1
2
3
4

Figure 3.2 The help prompts for an information node. The node has four links to other information nodes, labeled **1**, **2**, **3**, and **4**, which also work from the navigation prompt nodes. The links labeled **t** indicate timeout links.

them to think about the next one. In effect, the pauses slow down the recitation of the prompts.

2.2.1.2 Separate node for each navigation prompt

Expert listeners, however, should not have to wait through the slow recitation of the prompts in order to find particular commands. The prompts for each of the available actions should be separate nodes. The silences should go at the ends of the nodes. Then, listeners who know the button for advancing between navigation prompts (**0** in the diagram) can skip through them at their own pace, missing all of the pauses at the ends of the nodes.

2.2.1.3 Automatic transitions between prompts

There should also be timeout links between navigation prompt nodes for actions that are likely to be relevant to novices. That is, after playing one navigation prompt and then the short silence, if the listener has not pressed any buttons, the document should automatically transition to the next navigation prompt node. This will allow novices to hear all of the navigation prompts without pressing any buttons. Without the timeout links, they would have to learn how to skip between options before they could even hear what the commands are for navigating between information nodes, which would interfere with the goal of a quick start at listening to the contents of the document. In the tradeoff between supporting novice listeners who do not skip and forcing them to learn how to skip, this choice is in favor of supporting novices who have not yet learned to skip. The information about how to navigate through the information nodes is too important to be delayed while callers learn to skip through the navigation prompts.

2.2.1.4 Order prompts by utility for novices

The prompts should be recited in the order of their likely importance to novices. Experts will be able to skip through the prompts, so the order is less important for them.

2.2.1.5 Hide prompts for options that novices rarely need

If there are many options to prompt, of which only a few are likely to be relevant to novice callers, the timeout links should connect only the most relevant ones. In the diagram, the first two commands were deemed relevant to novice callers. At the end of the prompts for the commands more relevant for novices, there should be a prompt for the button to press to hear the other options. In addition to relieving the cognitive burden on novices by not reciting prompts for actions that are unlikely to be useful to them, this also provides a way for novices to learn the button that skips through the help prompts.

2.2.2 Timeout links only between subnodes

To help people keep track of available commands, their availability should not be time-dependent. This can be insured by restricting the use of timeout links, the links that automatically transition if no buttons are pressed during the playback of a node. Define a *virtual node* as an ordered set of nodes $(A_1,...,A_n)$ where:

- Each node A_i may contain links to the node before it and after it in the sequence.
- All other links are shared by all the nodes. That is, if there is a link labeled **3**, say, from A_i to node X, then there are also links labeled **3** from every other node A_j to X.

Call the nodes $(A_1,...,A_n)$ *subnodes*. It is OK to have timeout links between adjacent subnodes. Otherwise, they should be avoided. With this restriction, the only button presses whose meaning can change after a timeout link are those for moving forward and back within the sequence of subnodes.

Consider, for example, the timeout links between navigation prompt nodes as suggested in Figure 3.2. The entire figure represents a single virtual node. That is, pressing **1**, **2**, **3**, or **4** has the same effect from any of the nodes in the figure. The **0** transitions connect adjacent subnodes, so they do not have to be shared by all the subnodes. Since they are subnodes of a common virtual node, it is OK to have timeout links between the information node and the first navigation prompt, and between the navigation prompts.

2.2.3 Orientation Cues

Strunk and White advise writers to provide explicit orientation cues:

> As a rule, begin each paragraph either with a sentence that suggests the topic or with a sentence that helps the transition. If a paragraph forms part of a larger composition, its relation to what precedes, or its function as a part of the whole, may need to be expressed. This can sometimes be done by a mere word or phrase (*again; therefore; for the same reason*) in the first sentence. (p. 16)

Analogously, short phrases can orient listeners after transitions between nodes in an audio document. Such cues take time to play, however, so they should only be used when needed. For example, a node that recites the date of an event may be self-orienting if it is the only date recited as part of an event announcement. If, on the other hand, announcements include the event date and the last date to make reservations, the phrase "Event date" should precede the recitation of the contents of that field.

2.2.4 Regularities, Orderings, and Metaphors

Navigation prompts can tell callers what transitions are available. Orientation cues can prepare callers to understand the information after a transition. Both, however, take listeners' time and cognitive attention away from the contents of the document. To achieve fast navigation, listeners need to be able to *predict*, not just recognize:

- what transitions are available;
- what buttons initiate those transitions;
- what kind of information will be in the node at the destination of each transition;
- how the information at the destination is related to the information in the current node;
- which transition to take first in order to get to a particular piece of information.

Easily understood regularities, orderings, and metaphors can help listeners to make these predictions. Regularities allow listeners to predict available transitions based on analogy to other nodes already encountered in the document. Metaphors allow listeners to predict available transitions based on analogy to some external structure. Metaphors may also help listeners to understand the regularities in a document. Orderings allow listeners to predict which transitions will move them toward a desired piece of information.

For example, if every event announcement node has available transitions to nodes for the previous and next events in chronological order, listeners will learn to follow those transitions without hearing the prompts and without needing orientation cues as to what is being played after the transition. If prompts describe those transitions using the phrases "next announcement" and "previous announcement", and the buttons for executing those transitions are spatially arranged (e.g., **7** to the left of **9**), listeners may start thinking about navigation in terms of a list metaphor, which will make it even easier to predict those transitions. If the announcements are sorted in order of event date and the caller is looking for events on a particular date, then the date of the current announcement determines whether to go forward or backward in the list.

Style guidelines that call for regularity and helpful metaphors carry about as much weight as politicians' promises to "fight for the needs of the common man." Both are good ideas, but without more specifics, they are vacuous. Clearly, for example, every node in an audio document should not contain exactly the same information (the ultimate regularity), and not every metaphor will help callers to navigate. This section adds another layer of detail, cataloging the different kinds of regularities, the synergies between regularities and metaphors, and how prompts can emphasize the metaphor.

2.2.4.1 Regularities

When regularities in the network topology reflect perceived regularities in the information, listeners can predict available transitions and their destinations based on what transitions were available from 'similar' nodes. The value of regularities, then, depends on easily understood definitions of similarity. A set of nodes can be similar because they all have links to the same destination node or they all contain similar kinds of information. A set of transitions executed by the same button press can reflect an underlying similarity in the relations between the contents of the source and destination nodes. Sets of nodes and links can form structures with identifiable roles; the roles identify similarities between nodes in different structures.

Same destinations for outside transitions: subcomponents

One form of regularity is when a set of nodes all have transitions, activated by the same button press, with the same destination node. For example, separate nodes may present the fields of an event announcement, but pressing **9** moves to the next announcement from any of those nodes.

Consider the special case when all the transitions whose destination node is outside a particular set are shared by all the nodes in the set. That is, if one node in the set has a transition to a destination outside the set, then all of them have a transition, activated by the same button press, to the same destination node. Call any such set a *component*, and the nodes in the set *subcomponents.* The virtual nodes discussed above are special cases of components that have limited connections between subcomponents. In the degenerate case, any set consisting of just a single node fits the definition of a component. The other degenerate case is that the set of all nodes in an audio document is a component, since it has no transitions outside the set. The remainder of this section discusses components and transitions between components rather than individual nodes, so that, for example, an event announcement can be treated as a single component even though it may be represented by several nodes.

Same kind of information in components

A set of components can all contain information that listeners recognize as the same type. For example, a number of components may all contain event announcements.

Same relationships between components

A number of pairs of components may comprise a recognizable relation. For example, there may be a set of components that all contain lesson plans and a set of lists of questions, with a transition from each lesson plan to one of the lists of questions. All of the transitions, then, can be thought of as going from a lesson to its list of related questions.

Same roles across sets of components

There may also be similarities between sets of components. Components in these sets can have *roles*, with predictable transitions between roles. Sets with identifiable roles are called *structures.*

For example, two categories of event announcements both begin with components playing the 'header' role. Both components playing the header role have transitions, labeled by the same button press, to components playing the 'item' role. There can be more than one component playing the 'item' role, and each has a transition to another playing an 'item' role, except that the last 'item' component has a transition to a 'footer' component. Notice that the available transitions between roles are not quite regular, because not all components playing the 'item' role have transitions to components playing the 'footer' role. This slight irregularity makes it hard, for example, to listen to all of the announcements in a list without visiting the 'footer' component as well.

One of the requirements for a structure to be useful is that roles of components be easily identifiable. This was the failing of the two-sided lists used in one of the issue discussion applications. It was hard for listeners to remember whether they were in the positive or negative comments and hence whether to go right or left for another comment. The use of different voices (voice fonts) or auditory icons [Gaver 1986] might alert listeners to roles unobtrusively.

Combinations

Combinations of these kinds of regularities are even more powerful. For example, a list of lesson plans in the curriculum line draws on all four kinds of regularities. Each lesson plan contains the same type of information, each plays the 'item' role, and each has a transition to a list of questions about that lesson plan. Each lesson plan component has subcomponents for the fields, all with the same transition to the next announcement. Moreover, the subcomponents of each lesson plan are themselves a structure, with a single 'field' role, so that a caller who has learned to skip through the fields of one lesson plan can predict how to skip through the fields of another.

2.2.4.2 Ordering

If transitions between components are assigned based on an ordering of the components, then listeners may be able to predict the correct first transition to follow in order to reach a desired destination component. For example, a list of event announcements may be sorted based on the date of the event. Then, given a desired date or range of dates to look for an announcement, and given the date of the current announcement, the caller can predict whether to go forward or back in the list. If, on the other hand, the announcements are not sorted based on the factor currently of interest to the listener, (e.g., sponsoring organization), then the listener may have to resort to exhaustive search.

The value of ordering is not limited to lists. For example, a hypercube may be ordered independently along n dimensions. It is also not limited to total orders. For example a partial order[2] can provide significant guidance to the listener. Consider a tree representation of an organization chart, with the president as the root and people with no subordinates as leaves. From each node, transitions are available to go to the next sibling, to the parent, or to the first child. From anywhere in the tree, the listener can quickly find a particular employee, knowing only that employee's 'upline', the people he reports to

[2]In a partial order, not all pairs are related, but if a< b, and b < c, then a < c. A tree defines a partial order, with a < b if, following parent links from a, one eventually reaches b.

directly or indirectly. The listener simply follows parent transitions until finding someone on the desired employee's upline and then follows child and sibling links to descend the tree while staying on the employee's upline. The listener need not know in advance all of the people on the path from the current node to the nearest person on the upline, or even the order of the people on the upline. If, on the other hand, the listener wanted to find the person with the most seniority in the organization, this partial ordering would not be of much use.

2.2.4.3 Metaphors

While regularities help listeners to make predictions based on previous experience with an audio document, metaphors help callers to make predictions based on experiences with some external concept. The two cardinal rules of any metaphor, then, are:

1) The external concept must be familiar to users.

2) The external concept must have predictive power in navigating the document.

The first rule may seem trivial, but designers often fail to heed it in practice. For example, recall that the joke collector used the compact disc jukebox as an external concept. Many people were not familiar with the skip forward and skip backward keys on a compact disc player. Moreover, nobody had encountered a CD jukebox where songs on one disc were linked to other discs, because no such jukeboxes exist.

There will always be limits to the predictive power of any metaphor, for the external concept is by definition not the same as the target domain, but merely analogous in some ways. In the case of audio documents, metaphors can help predict the types of transitions available and some properties of the destination nodes. It is unlikely, however, that any metaphor will predict the button presses that will execute those transitions, since there are few uses of keypads like the telephone's.

Metaphors, then, will be most helpful in conjunction with regularities. Metaphors can help callers to understand the roles in a structure, the similarities in sets of components and sets of links, and the orderings of components. If the same metaphor is applicable in many places in an audio document, listeners may learn the mappings between the transitions that the metaphor predicts and the button presses that initiate those transitions.

Keypad layout matches external structure

If an external concept has a spatial layout, the keypad layout should match it. For example, suppose that three nodes are laid out from left to right in the external concept (a list, say). The button to move from the middle node to the right node should be on the right side of the telephone keypad, and the node to move from the middle node to the left node should be on the left side of the telephone keypad. This will make it easier for listeners to remember the button presses associated with the transitions.

Document contents explicitly state the metaphor

The contents of the document should explicitly describe the metaphor that listeners can use. For example, in the initial node for playing back a list of event announcements, a prompt can state that it is a list of announcements. The navigation prompts, too, should emphasize the external concept. For example, the node that plays back one event announcement from a list can have an attached prompt of the form, "For the next announcement, press 9."

3 EXTENSION TO BULLETIN BOARDS

Telephone bulletin boards add two new twists to the concept of audio documents. If a bulletin board as a whole is considered as a document, then the document does not have a single author and it changes over time. Even as the contents change, the document should be scannable. In addition, callers should have the same degree of control over the process of adding new information to telephone bulletin boards as scanning gives them when listening to information.

3.1 MULTIPLE AUTHORS

Any addition of new nodes and links to a document, then, should retain all the desirable document properties described above. The new nodes should contain short recordings and follow the progressive disclosure principle. Some of the added nodes should contain summary information that helps in selecting other nodes to hear. The nodes should include orientation cues and prompts for navigation. After the addition of new nodes, the document should still have regularities, orderings, and metaphors that help listeners predict the available transitions from nodes.

3.1.1 Aids for Adding

Contributors to a document, however, are not typically human factors experts. Audio documents can help contributors preserve the desirable properties in four ways, as described below.

Expandable Structures with Independent Components

To accommodate addition of information by contributors who do not collaborate with each other, the structures in the document should consist of self-contained components, such as event announcements. The metaphors for navigating those structures must be 'expandable.' That is, they must have roles with indefinite numbers of instantiations. For example, lists have an indefinite number of items and trees have indefinite depth.

While navigation metaphors help listeners predict how to navigate, metaphors for addition should help contributors to add information in such a way that the navigation metaphors still work. In particular the navigation metaphors should provide predictive power about where to go to make particular kinds of additions. For example, contributors might go to the list header to add a new item to that list, or to an item to add a question about it.

Input Format Divides Components into Meaningful Segments

To encourage short nodes, the input format should subdivide each independent component into meaningful segments. Three techniques can accomplish this, each placing successively more responsibility on the contributor:

Automatic analysis of acoustic structure. The contributor makes a single long recording. Through analysis of that recording, a computer divides it at places that appear to be beginnings of new phrases [Hindus and Schmandt 1992].

Predefined fields. The contributor makes a separate recording for each field.

Author-defined segmentation. Authors explicitly press a button whenever they think they have completed an idea and are about to begin a new one [Degen, et al. 1992].

Input Format Encourages Inclusion of Important Information

Unsophisticated contributors may not be aware of information that will help future listeners to understand their contributions or help the computer to arrange them. Four features can encourage and aid contributors in this regard:

Reminder prompts. Before recording, the computer can remind contributors to include certain information. Most people record such a reminder as the outgoing message on their answering machines. Field names can serve the same purpose in segmented input.

Initial values. The computer can automatically fill in default values for some fields, such as the date and time of recording.

Picklists. When only a few alternatives make sense, contributors can select one from a menu.

Validity checks. The computer can reject invalid entries of typed-in values such as dates and quantities.

Computer Fills in Transitions and Navigation Prompts

The computer can automatically integrate the new component into the existing structures. Based on the location from which the contributor initiated adding the component, and on symbolic fields such as typed-in dates, the computer chooses one or more existing structures and locations to add it, and creates the necessary transitions between the new component and existing ones. For example, a new event announcement gets added to the current list, in appropriate position to maintain an ordering by event date, and is also added to the beginning of a master list of announcements, thus preserving its reverse chronological ordering by recording time.

In addition, the computer should automatically provide navigation prompts for the transitions from the component. It can do this by reusing existing prompts for similar components. For example, the computer might reuse the prompt, "For the next announcement, press **9**."

3.1.2 User Control During Input

The same level of control that scanning provides to listeners should also be available to contributors during the process of adding new information. First, they should press buttons to initiate recording. By contrast, most answering machines and voice mail systems begin recording when the machine is ready, not when the caller is ready. This often leads to garbled beginnings of messages. Second, contributors should have the chance to review, re-record, and save or throw away recordings, and control the timing of all these actions as well.

3.2 FINDING NEW INFORMATION: CHRONOLOGICAL ORDERING

In a telephone bulletin board, listening and recording are interspersed. The same listener may call repeatedly, to check for new information. To accommodate such callers, the document should separate new nodes from old nodes. That is, there should be paths that connect all the new nodes without going through nodes that have already been heard. This can be accomplished through chronological ordering. Note that the same information may appear in several places in a document, so that chronological ordering can be provided in addition to other arrangements of the information that may intersperse new and old recordings.

4 BEYOND THE TELEPHONE

4.1 OTHER AUDIO DOCUMENTS

Two projects [Arons 1991b, Muller and Daniel 1990] have explored a network model for audio documents in which actions are initiated by speech recognition of spoken utterances rather than by button presses. Muller and Daniel first proposed the concept of a voice-only hyperdocument with speech navigation. Arons implemented a medium-sized document (80 nodes, approximately 700 links) and refined some of the user interface features of it.

When considering the possible value of speech input, command-line interfaces, such as UNIX and DOS, are a useful analogy. Assume speaker-independent, continuous word recognition with accuracy and speed at the levels now achieved for recognition of touch-tones. This would allow hands-free input of text faster than the fastest typists, but otherwise would have the same characteristics as a command-line interface.

Novice listeners will find it easier to guess phrases than to guess the button presses that initiate transitions in the document, especially if many synonyms initiate the same links

[Furnas 1985, Wixon, et al. 1983]. Still, the document will have to include navigation prompts that list the available phrases for situations where callers are unable to guess. Metaphors for thinking about the structure of audio documents will still be helpful. Instead of having the keypad layout match the spatial aspects of the metaphor, however, the text strings should draw on the metaphor, for example using the string "more details" to label a link between a summary node and a node that gives more details. Overall, the text labels should make it easier for novice listeners to get started and for them to learn the commands that will let them scan through an audio document. The use of speech input does not, however, fundamentally alter the interactions that a document should afford or the document properties that will facilitate such interactions.

For experts, the actual scanning can be accomplished more quickly with button presses than with spoken input. It takes longer to say, "more details" than it does to press a button, even assuming a perfect text-to-speech recognizer. Thus, for speed of scanning, buttons will be better than speech input. Other input devices, such as eye trackers or direct neural connections, might be even faster.

4.2 KEYHOLE INTERFACES

The central limitation of audio documents is the inability to present several things in parallel and let the user shift attention without computer aid, as is possible with the printed page or computer screens. Audio interfaces are not the only ones that suffer from this limitation. For example, many electronic organizers have LCD screens as small as two lines by fifteen characters. The use of very large fonts on a full size screen, to aid the visually impaired [Ladner, et al. 1987] creates the same limitation. Any of these can be thought of as keyhole interfaces. A small window (a keyhole) moves around over an otherwise opaque document. A good design strategy for keyhole interfaces is to give users a lot of control over when and where the keyhole moves and to structure the document in such a way that readers can predict the available movements and their effect.

All of the properties described in this chapter as desirable for audio documents are equally desirable for any keyhole interface.

4.3 BROWSING IN VISUAL HYPERDOCUMENTS

Since audio provides only an ephemeral display, the research reported in this thesis had to confront the fundamental problem of how to organize information so that users naturally maintain orientation as they navigate through it. While the availability of large displays partially obscures this fundamental problem, especially for the fairly small information spaces used with most current hypermedia systems, the problem is still present. In effect, any size display can be a keyhole relative to a much larger information space. Many of the guidelines for information organization derived for audio hyperdocuments apply more generally to any information space that is much larger than the size of the display used to browse it.

Larger displays, however, do change the analysis in two ways. First, readers can shift their eye gaze between information chunks that are displayed simultaneously, reducing the need for fine-grained computer assisted skips. Second, part of the screen can show available transitions that will change what is displayed. Much research has gone into exploiting these two sources of power. Still, however, if the information space is much larger than will fit in a few displays, readers may get lost in hyperspace [Conklin 1987], losing a sense of orientation after following a few links [Utting and Yankelovich 1989].

4.3.1 Displaying Lots of Information at Once

With a large display, readers can use eye gaze shifts as well as machine-mediated changes of attention. To exploit the eye gaze shifts, a hypermedia document can display a number of related pieces of information simultaneously. This is the idea behind tabletop cards [Trigg 1988] in the Notecards system. The basic unit of information is the card, which can be any size, but is usually much smaller than the screen. A tabletop card is a set of cards placed on the screen in particular locations, effectively making the size of a tabletop card the whole screen. When a tabletop link is followed, the new set of cards is displayed, and the reader can use eye gaze shifts to scan the contents of the new set of cards.

Even with a very large display, mechanically mediated (as opposed to eye gaze mediated) shifts of attention may be needed, for two reasons. First, the information space may be too large to fit on a single display. Second, the mechanically mediated shifts are not spatially constrained. While modern typographic conventions convey a lot of information about relationships among parts of documents through spatial proximity, the number of relationships may swamp the limitations of space. For example, there is no way to draw five regions on a page in a way that all of them are contiguous with all four of the others [Bollobas 1979].

4.3.2 Displaying Memory Aids and Orientation Cues

In audio documents, listeners rely on memory and cognitive processing to know what links are available, what is in the current node, and what nodes have been visited recently. A less ephemeral display, such as a screen, can provide visual cues that allow readers to use perception instead of memory for some of the tasks. Much research has gone into such visual cues, both to show the context of nodes that have already been visited, and to show some information about the nodes that are nearby.

Context mechanisms

The first type of memory aid displays the temporal context of which nodes have been visited and which links traversed. Researchers have explored variants on list displays [Foss 1988, Utting and Yankelovich 1989, Walker 1987] and tree displays [Foss 1988].

Visual Maps

Many variants of map displays have been used to alert readers to what links are available and where they lead. A global graph browser displays all of the nodes in a document and how they are related [Halasz July 1988, Utting and Yankelovich 1989]. If the nodes are highly connected, these graphs are not very helpful as the lines connecting nodes cross too frequently. Several others use 'fish-eye' displays that make the current node the center of attention [Collier 1987, Furnas 1986, Travers 1989].

For graphs that are purely hierarchical (i.e., trees) a global graph browser can be more effective, since none of the lines representing links need cross. One alternative display method for hierarchies is an outline, where the children of a node are listed below that node, and indented a couple of spaces [Remde, et al. 1987, Walker 1987, Weyer and Borning 1985]. Another method is spatial inclusion, where the box displaying a node contains smaller boxes for each of its child nodes [diSessa and Abelson 1986, Feiner 1988, Travers 1989].

Domain-specific maps can also help readers. For example, a display of a cell, a physical device and its substructure, or a timeline can have embedded buttons [Landow 1989, Shneiderman 1987]. Readers can keep returning to the domain-specific map to select links, rather than following links between nodes.

Orientation Cues

Some of the available display space can present explicit orientation cues. Landow proposed a set of rhetorical rules for authors of visual hypertexts [Landow 1987, Landow 1989] that included the provision of orientation cues at the source and destination nodes of every link. Since one node may be the destination of several links, however, and the different links may require different orientation cues at the destination, this can be difficult to achieve in practice. In addition, as the number of available links grows, an increasing portion of the available display is taken up with orientation cues rather than information.

4.3.3 The Elements of Style for Visual Hyperdocuments

While these mechanical aids are of some use, they do not address the primary reason for orientation problems, which is that readers are unable to predict what links will be available and where they go. The greatest strength of hypertext links is their ability to connect nodes that are not spatially near each other, but this means that they have the potential to connect conceptually unrelated nodes. Just as with audio documents, the real solution to the problems of orientation in hypermedia systems lies in organizing the information: making the document topology reflect regularities and orderings in the information space and providing external metaphors to help readers understand the topology.

Preferred paths

One way to impose organization on the document is for the document designer to suggest a subset of links that are worth following. This can be accomplished through procedural path descriptions [Zellweger 1988, Zellweger 1989], through graphical displays of the subset [Trigg 1988], or through a Petri Net formalism that restricts the availability of links in different contexts [Furuta and Stotts 1989, Stotts and Furuta 1989]. All of these methods attempt to aid reader orientation by reducing the number of choices available to just those that the author thinks will be most valuable to readers. Thus, these approaches are analogous to the audio document style guideline of hiding prompts for transitions that novices rarely need.

Regularities, orderings, and metaphors

This chapter has catalogued the kinds of regularity, ordering, and metaphor that will help listeners to predict the available links. In visual documents, prediction is less important, since visual cues can allow readers to recognize available links, perhaps as fast or faster than they would be able to predict them. Still, the same regularities, orderings, and metaphors will allow readers to recognize the meaning of the visual cues.

Several researchers have proposed structures that impose regularities on hyperdocuments, especially lists and hierarchies [Akscyn, et al. 1988, Garzotto, et al. 1991], as well as metaphors such as stacks [Harvey 1988] and books [Shneiderman, et al. 1991]. This chapter suggested that even links between structures can and should mirror regularities in the information space.

Consider how these guidelines might apply to a virtual reality system for browsing the scientific literature. It could draw on a bookshelf metaphor to present lists of articles. Several identifiable relations might define a set of links available from each article to a shelf of other articles that:

• the current one cites (ordered by frequency that other articles cite them);

• cite the current one;

• appear in the same book, journal, or conference proceedings;

• match the current one on keywords or some full-text similarity measure [Creecy, et al. 1992, Dumais, et al. 1988].

The familiar bookshelf metaphor and the regularities in available links to other bookshelves will allow readers to focus on the contents of the articles rather than the navigation mechanisms.

5 SUMMARY

Skip and Scan is a slogan for audio documents. Listeners should hear short speech segments and then press buttons to skip to the beginnings of other speech segments. If the document is well organized, this behavior can have the same effect as visual scanning of paper and screen-based documents. That is, listeners can select a small fraction of the document to listen to, without spending a lot of time selecting the right parts. Audio documents should also support novice listeners, letting them get a quick start and telling them what buttons to press while gradually building their skills at navigation so that they, too, can skip and scan. Finally, in collectively authored audio documents, the document should help contributors to add information in ways that others will be able to scan, and contributors should get the same degree of control during the recording process that they get while listening.

To aid designers of audio documents, this chapter not only identified these desirable interaction styles, but also identified properties of audio documents that will make such user behavior possible. Table 3.3 summarizes that translation.

The next chapter presents programming constructs that make it easy to create audio documents with these properties. Those tools use the form metaphor for addition to help contributors add information that can be navigated using list metaphors. Other metaphors are possible, however, such as grids or calendars. The desirable document properties, then, serve not only to validate the programming tools in the next chapter, but also to guide the design of any future programming tools for audio documents.

6 CONCLUSION

Digital storage and random access have ushered in a new era in audio. But as a society, we do not yet have the authoring or listening skills relevant to the new era. These

Design Constraint	Desired Behavior	Document Properties
Small relative keyhole size	Scanning	
	Hear a small percentage of document	Summary nodes Short paths
	Fast navigation	Orientation Cues Regularities Same destination nodes for outside transitions Same kind of information in components Same relationships between components Same roles across sets of components Ordering Metaphor Choose a known external concept External concept highlights the regularities and ordering Keypad layout matches spatial layout in external concept
Small absolute keyhole size	Fine-grained scanning	
	Hear a small percentage of document	Small nodes One topic per node Progressive disclosure
Novice users	Quick start; Learn to scan	Navigation prompts Timeout links only between subnodes Document contents explicitly state the metaphor
multiple independent authors	Add information without disturbing desirable document properties	Expandable structures with independent components Input format divides components into meaningful segments Input format encourages inclusion of important information Computer fills in transitions and navigation prompts
authorship over time; repeat listeners	Find new information fast	Chronological ordering

Table 3.3 A summary of how design constraints translate into desired listener and author behaviors and then into desirable properties of the documents.

guidelines are a first attempt to define the elements of style for audio documents. As people gain more experience authoring and listening to audio documents, the guidelines will evolve. Right now, even the best audio document is much harder to scan than a well-designed visual document. But visual document design has a several thousand year head start.

Educated people have had years of formal and informal training on how to scan visual documents but little in scanning audio documents: as a result, it is hard for us even to have good intuitions about how good audio documents could be, relative to visual documents. It might even be that well-designed random access audio documents will be more effective communication tools than visual documents. After all, people learn spoken language at an early age, without formal instruction, while reading and writing require formal instruction. The guidelines in this chapter represent the first step on a path toward improving the scanability of audio documents. The end of that path is impossible to predict with confidence. Perhaps the spoken word will even replace the written word as a medium for communication of complex ideas.

4 HyperVoice

Chapter 2 showed that login processes, menus, and lists are helpful abstractions for *designing* and describing multi-author audio documents. Chapter 3 argued that menus, lists, and forms are helpful metaphors for *using* multi-author audio documents. This chapter shows that those abstractions are also helpful for *programming* multi-author audio documents.

Previous software toolkits for building phone-based applications utilize a state-machine abstraction. Programmers specify graph nodes, whose contents are application data and navigation prompts, as well as the allowable transitions between nodes. During a telephone call, a state-machine interpreter keeps track of the current node and plays the sounds associated with it, halting playback and transitioning to a new node whenever it detects a touch-tone. This abstraction layer should already be familiar from Chapter 3: the Skip and Scan audio document properties were all described as properties of state-machine representations of documents.

HyperVoice introduces a new, higher-level abstraction layer, called the *application* layer. This layer separates data from presentation formats, allowing both to be reused in modular fashion. A single information object or list of objects can be presented in multiple ways, using different presentation formats. A single presentation format can apply to several information objects or lists, and to new objects as contributors add them over the phone.

The application layer primitives are login procedures, lists (including menus), and forms. An interpreter generates state-machine layer specifications from application layer specifications, based on parameters of the primitives that specify:

• the selection and ordering of application data;

• aids to entry of application data;

• access restrictions;

• user interface style, including what commands will be available for navigation and how navigation prompts will be generated.

Automatic generation of state-machine layer programs offers three advantages over direct specification of state-machine programs:

Faster development. The specification of a particular list, for example, does not include any of the features that are common to state machine representations of all lists. This makes programs shorter and reduces opportunities to introduce errors that will need to be debugged.

Participatory design. Programs are easier for non-programmers to understand because they use the same abstractions that are useful in informal descriptions of designs.

Consistent, good user interfaces. Automatic determination of dialogue sequencing and prompts insures a consistent user interface. Part of the interface style is built in and style parameters help application programmers to specify other aspects of the interface style in a way that is consistent with the Skip and Scan document properties. By contrast, a program specified directly at the state-machine layer may lack regularities or mappings to external metaphors. Even worse, a programmer can accidentally write a prompt that tells the caller to press the wrong button. With HyperVoice, automatic generation and placement of prompts insures that they match the available actions.

Any higher-level language makes some programs easier to express while making other harder or even impossible to express. HyperVoice cannot implement programs based on other abstractions, such as matrices or, more generally, hypercubes. The parameters of the list presentations do not even allow all possible presentations based on a list metaphor:

for example, the later items in the list cannot be reordered dynamically based on how long the caller spends listening to earlier items.

On the user interface side, HyperVoice can not generate all the documents that can be expressed at the state-machine layer. As argued above, this may be beneficial, since it guides the generation of documents that conform to the Skip and Scan guidelines. But HyperVoice cannot even generate all the documents that conform to those guidelines. For example, it can generate the prompt: "For the next announcement, press 9," but not the prompt, "To hear another announcement, press 9."

The appropriate question, then, is whether HyperVoice is expressive enough to accommodate the functionality and user interfaces needed for its target domain of applications. HyperVoice can generate the functionality required for all the applications in Chapter 2, as summarized in Table 2.9, through parameters that exploit the structure of information objects, especially non-voice fields. HyperVoice also provides enough flexibility to specify user interfaces, both dialogue sequencing and prompts, that are tailored to those applications.

This chapter begins with an overview of the HyperVoice application layer language. It also includes one section each on the language features that determine the selection and arrangement of information, aids to entry of information, access restrictions, and the user interface style. Appendix A is a programmer's manual for the HyperVoice language. It systematically describes the parameters of each of the language primitives and their intended semantics.

1 LANGUAGE OVERVIEW

HyperVoice assumes the following dialogue structure for all applications:

- A login procedure determines the initial list presentation for each caller.

- The caller navigates through the items in a list and selects one in order to jump to a related list.
- From some locations in some lists, the caller can initiate the addition of a new object, which the caller edits using a telephone form.
- Callers with enough privileges can initiate special commands, such as cutting and pasting items between lists.

Table 4.1 and Figure 4.2 provide overviews of the primitives that are used to specify HyperVoice programs. The table summarizes the function of each of the primitives while the diagram emphasizes the relations between them. The arrows in the diagram indicate when an instance of one primitive can contain a pointer to an instance of another primitive.

A HyperVoice program begins with a Login primitive. It determines the initial greeting that callers will hear and whether or not they have to login in order to access the system.

Primitive	Description
Login Process	Specifies the login procedure necessary to access an application.
User	Has fields for a recorded name, an id number, a password, and the 'privileges' that should be accorded to a particular user.
List Jump	Specifies special digit sequence a caller with appropriate privileges can press to jump to a particular list.
List Presentation	Pairs a List with a List Format to specify the presentation of one list of objects.
List	A list of objects.
List Format	Specifies how to play back the contents of a list, including how to filter and sort it, and how new objects can be added.
Filter	Selects a subset of a list to be presented.
Validity Check	A predicate on the contents of a field. Used in Filters and Field Edit Formats.
Sort Ordering	Specifies the order in which to present objects from a List.
Item Format	Specifies which fields of one object to play and in what order.
Field Format	Specifies how to play back one field.
Select Action	Used as part of an Item Format to specify what action to take when an object is selected from a list.
List Action	Specifies how new items can be added.
Extension Format	Specifies what kind of object will be added and where it will be added.
Edit Format	As part of an Extension Format, specifies a telephone form for editing a new object.
Field Edit Format	Specifies how to present one field in a telephone form.

Table 4.1 The primitives of the HyperVoice language.

It also determines the root List Presentation and initial privileges, plus any List Jumps that will be available through special commands.

A List Presentation pairs a List with a List Format, which separates the application data (a List) from the presentation format (the List Format). Much as skilled procedural

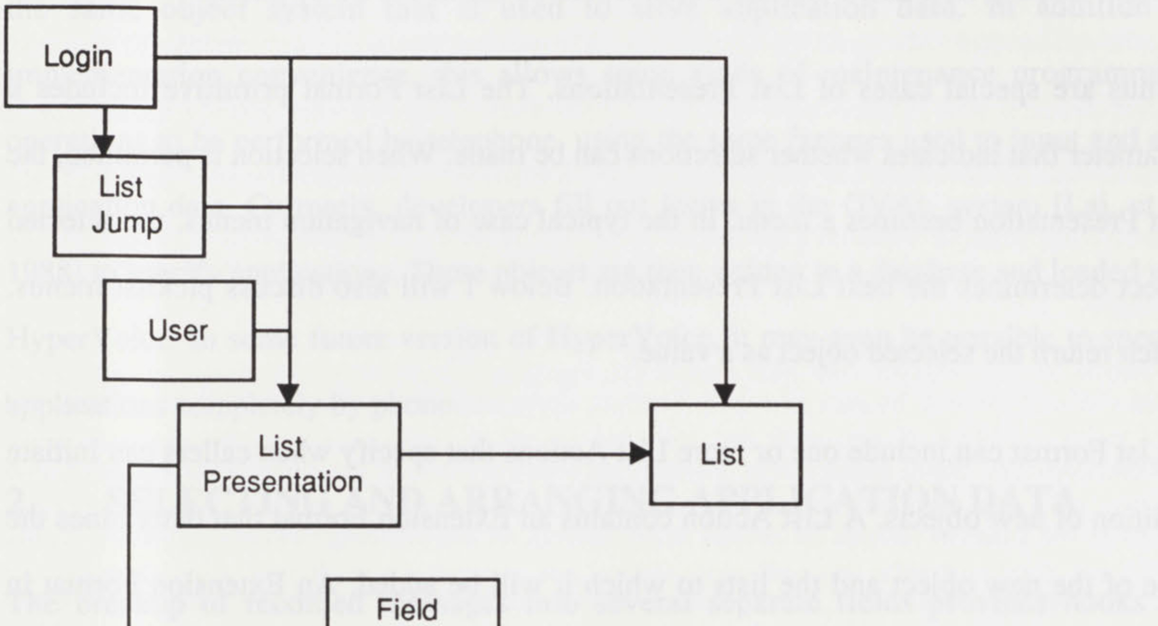


Figure 4.2 The relations between language primitives.

programmers create reusable procedures, a skilled HyperVoice programmer finds ways to reuse Lists and their component objects, as well as List Formats and their component presentation formats. Reuse of a List, for example, by selecting different subsets or ordering them differently, makes it easy to add an object to the list and have the change affect several list presentations. Reuse of List Formats allows callers to create new lists without creating new List Formats, which would be tedious to do by phone.

Menus are special cases of List Presentations. The List Format primitive includes a parameter that indicates whether selections can be made. When selection is permitted, the List Presentation becomes a menu. In the typical case of navigation menus, the selected object determines the next List Presentation. Below I will also discuss picklist menus, which return the selected object as a value.

A List Format can include one or more List Actions that specify when callers can initiate addition of new objects. A List Action contains an Extension Format that determines the type of the new object and the lists to which it will be added. An Extension Format in turn contains an Edit Format that specifies features of the telephone form for editing the new object.

When presenting any list of objects, HyperVoice provides cut and paste operations. The cut operation removes the current object from the current list. The paste operation adds the last cut object to the current location in the current list. This can be used, for example, to manually sort a list or to throw out objects that are no longer relevant. Access to the cut and paste operations may be restricted through access privileges, as will be discussed in the section below on access restriction.

The HyperVoice language uses an object-oriented data model for application data. Object classes determine the fields of which messages are composed. The database handles five data types: recording, text, date, integer, and pointer to another object. Programmers can create new object classes that reflect the natural structure of the messages to be posted in particular applications. For example, the Event Announcement object class has eight fields: headline, event date, time, location, contact number, details, date added, and category.

All of the HyperVoice primitives are implemented as instances of pre-defined classes in the same object system that is used to store application data. In addition to implementation convenience, this allows some kinds of maintenance programming operations to be performed by telephone, using the same features used to input and edit application data. Currently, developers fill out forms in the OVAL system [Lai, et al. 1988] to specify applications. These objects are then written to a database and loaded into HyperVoice. In some future version of HyperVoice, it may even be possible to specify applications completely by phone.

2 SELECTING AND ARRANGING APPLICATION DATA

The breakup of recorded messages into several separate fields provides hooks for specifying different selections and arrangements of information. List Formats use the contents of non-voice fields of the objects to specify the selection and order of objects in a list presentation. Item Formats and Edit Formats select and order the playback of fields from individual objects.

2.1 SELECTING AND ORDERING OBJECTS

A Filter, as part of a List Format, determines a predicate on objects. When presenting a List, only objects that satisfy the predicate are presented. For example, a filter on a typed-in 'event date' field can filter out announcements of events that have already passed. Filters can express predicates that are boolean combinations of restrictions on the values in single fields. Thus, a filter can restrict the values in two different fields independently, but cannot restrict the value in one field to be less than the value in another field.

Filters on fields that contain pointers to other objects can be especially valuable in expressing complex selection operations. For example, in the project management application described in chapter 2, someone could hear all of the status reports related to tasks for which he was responsible. To accomplish this, status reports, tasks, and people are three separate object classes. Each status report object contains a field with a pointer to a task object. Each task object has a field that contains pointers to one or more person objects. The apparently complex filter, then, can simply follow those pointers to determine if a particular status report is about a task for which a particular person is responsible. Without explicit modeling of the relationships between the three types of information objects, through pointers between the objects, this kind of filter would not be possible.

Field Order Formats, as part of a List Format, determine the order in which to present the selected objects. Each Field Order Format specifies a field of the objects to compare, and whether the objects should be sorted in ascending or descending order. For example, the specification *['event date' ascending]* would cause an announcement of tomorrow's event to precede an announcement of next week's event. When more than one Field Order Format is specified, the second breaks ties in the ordering specified by the first, the third breaks ties in the ordering specified by the first two, and so on.

2.2 SELECTING AND ORDERING FIELDS

Item Formats include parameters that specify which fields of objects to play back, and in which order. This allows different fields and different field orderings in different contexts. For example, in the teachers' curriculum line application, question objects included a field for the lesson plan with which they were associated. This field was only of interest to the head teacher, who accessed all the questions through a single master list. When other teachers accessed questions by selecting a particular lesson plan, the 'Lesson Plan' field was not played back.

Similarly, it may be appropriate to hide some fields in forms. This is especially helpful when an initial value generator (discussed below) fills in a value automatically that will be needed for filtering or sorting purposes, but which callers need not edit or even know about. Field Edit Formats have a parameter for whether to include fields in the form.

3 AIDS TO INFORMATION ENTRY

The operations for selecting and sorting objects work only when the objects have been added to the appropriate lists and when they have non-voice fields with correct values filled in. The programmer can structure the information entry process to facilitate that.

3.1 SELECTION OF LIST(S) TO ADD TO

Two parameters of the Extension Format specify which lists new objects will be added to. Typically, new objects are added to the current list, but they can be added to other lists instead. New objects can even be added automatically to more than one list. This is how, for example, new questions in the teachers' curriculum line are added to both the list of questions for the current lesson and to the master list of questions that the head teacher accesses.

3.2 INITIAL VALUES, PICKLISTS, AND VALIDITY CHECKS

Three language features help contributors to fill in meaningful values in fields that will later be used for sorting and filtering operations. First, Field Edit Format primitives can specify initial values for the contents of particular fields. These can either be actual values, or specifications of how to generate values at run time. There are specifications for filling in today's date, for filling in a pointer to the User object for the person who logged in at the beginning of the call, and for filling in a pointer to the current list of objects. In addition, the initial value can be specified by an Extension Format primitive that creates a new object to link into the field and fills in some of the linked object's fields.

One advantage of treating the language primitives as objects is that an existing List Format can be specified as the initial value for a new List Presentation. For example, the creation of a new event category is accomplished by adding a new List Presentation to the menu of categories. The initial value specifier for its 'list' field causes a new List to be created. The initial value for the 'list format' field specifies an existing List Format to use.

Validity checks restrict the values that will be accepted into a field. For example, in an event calendar application, a validity check on the 'event date' field would reject values that cannot be interpreted as dates and reject dates that have already passed.

A picklist allows callers to set a field to contain a pointer to another object, selected from a predetermined lists of objects. A picklist is just a List Presentation that has two important characteristics. First, the List Format indicates that selections are allowed from the list. Second, the Item Format primitives that specify how to play back particular objects from the list contain a parameter that indicates what action to take when the object is selected. For picklists, the appropriate action is to return the object as a value. The value returned from a picklist selection is then added to the field that is being edited.

4 ACCESS RESTRICTIONS

Several language constructs allow the programmer to restrict access to certain actions and List Presentations. During a registration procedure, a caller enters a user id and a password that pick out a User object from a predetermined list. That User object determines an initial privilege level (*none*, *regular*, or *super*), and an initial List Presentation. If no login is required, all callers get the same initial privilege level.

Some actions, such as the List Actions that add new objects and the List Jumps described below, require a minimum privilege level. Callers who know a special code can enter it during a call to increase the initial privilege level (analogous to becoming a super user on a UNIX system).

The initial List Presentation determines what other List Presentations will be accessible. Some callers can start places that have access paths to fewer List Presentations. A second way to restrict access to some List Presentations is to make them accessible only through List Jumps. While most navigation during a phone call involves a move from hearing one information object to hearing a related list of objects, the List Jumps determine jumps to unrelated lists that will be available from anywhere in the information space. Thus, while most of HyperVoice adopts a navigation metaphor, List Jumps introduce a form of direct addressing, with users entering an unprompted code that determines the address. If the programmer sets a high minimum privilege level for a List Jump, this will restrict access to the destination List Presentation.

Note that the initial privilege level and List Presentation assigned to each user can be changed by phone, since they are just fields of the User object. It is even possible to add or remove new User objects over the phone. The teachers' curriculum line used this facility.

5 USER INTERFACE STYLE

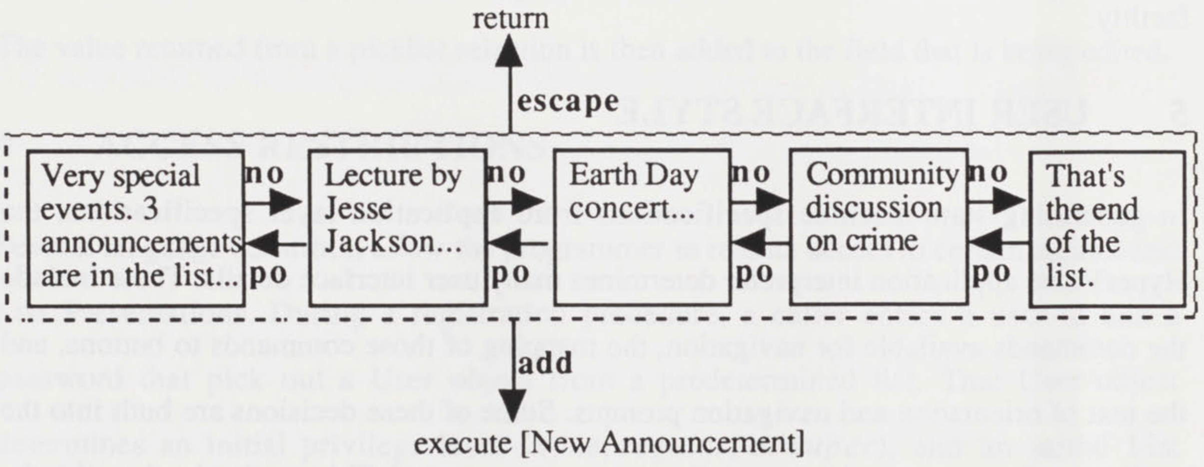
In generating state-machine specifications from application layer specifications, the HyperVoice application interpreter determines many user interface details. These include the commands available for navigation, the mapping of those commands to buttons, and the text of orientation and navigation prompts. Some of these decisions are built into the application interpreter; others are under programmer control. This section begins with the built-in features of the mappings from List Presentations and Edit Formats to state-machine programs, then proceeds to describe the effects of various interface style parameters on those mappings.

5.1 BUILT-IN STYLE

5.1.1 List Presentations

A single List Presentation object generates a whole structure at the state-machine layer. To a first approximation, there is one component for each object that passes the filter specified in the List Format, plus a header component and a footer component. The caller begins by hearing the list header, and then uses forward and backward buttons to move between objects in the list. Figure 4.3 illustrates this first approximation of how a list of three event announcements would be presented.

In Figure 4.3 and others that follow, words rather than numbers label the transitions between components. As will be described below, the interpreter looks up the mapping of word labels to telephone buttons in a table. In all of the field trials of prototypes, the **next object** and **previous object** commands were mapped to **9** and **7**. Boxes in the diagrams inherit links from the outside in; the inner boxes are subcomponents. That is, any link



return
escape
Very special events. 3 announcements are in the list...
Lecture by Jesse Jackson...
Earth Day concert...
Community discussion on crime
That's the end of the list.
no
po
add
execute [New Announcement]

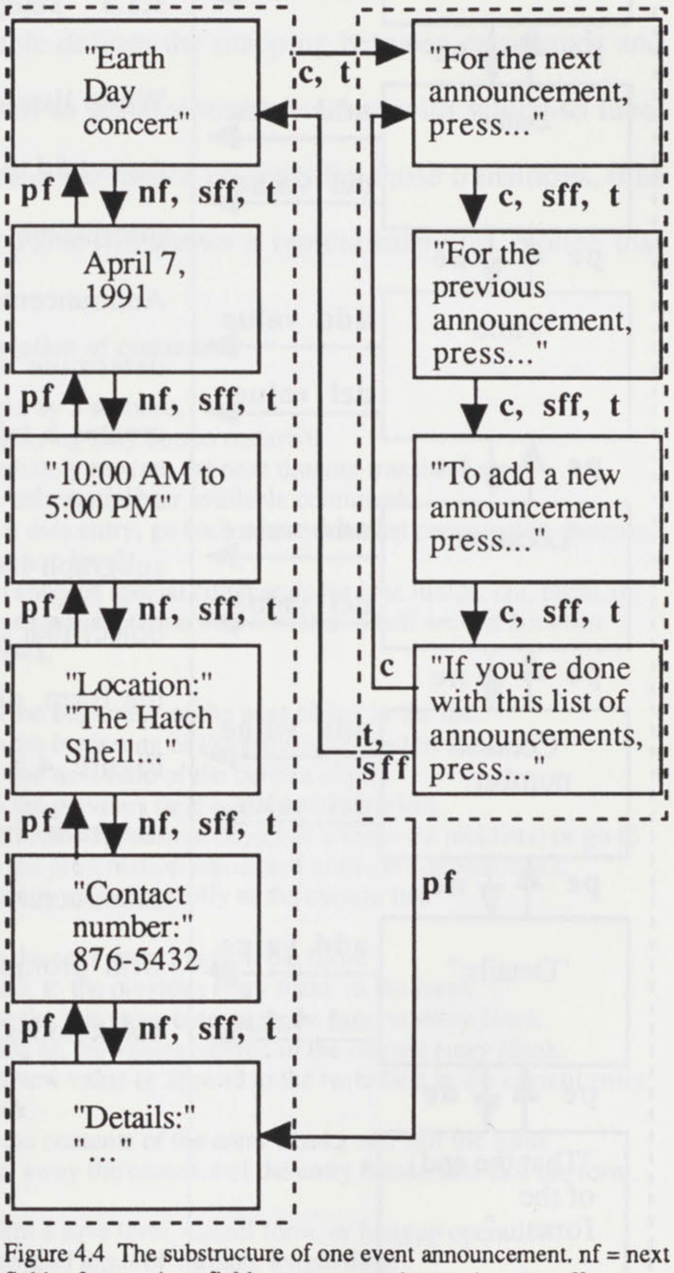
Figure 4.3 The initial expansion of a List Presentation object into a state-machine program. **no = next object; po = previous object.**

emanating from an enclosing box is available from any of the nodes inside it. For example, the caller can press **escape** from anywhere in the list to return to the previous list that was presented, or press **add** from anywhere in the list to start the addition of a new announcement.

5.1.2 Object Substructure and Navigation Help

Figure 4.3 is just a first-order approximation of the actual state-machine graph that is generated. Actually, the component for each object is subdivided into one node for each field. As described above, the Item Format determines which fields to include, and in what order. In addition, each component, including the header node, has attached one or more prompt nodes that indicate the available commands from that component.

Figure 4.4 illustrates the subdivision of one of the event announcement components from the previous figure. There are several transitions available between subnodes. The **next field** and **previous field** commands skip back and forth between fields. The **commands** command goes to the



"Earth Day concert"
April 7, 1991
"10:00 AM to 5:00 PM"
"Location:" "The Hatch Shell..."
"Contact number:" 876-5432
"Details:" ...
pf
nf, sff, t
c, t
"For the next announcement, press..."
c, sff, t
"For the previous announcement, press..."
"To add a new announcement, press..."
"If you're done with this list of announcements, press..."
c
t, sff
pf

Figure 4.4 The substructure of one event announcement. nf = next field; pf = previous field; c = commands; t = timeout; sff = smart fast forward.

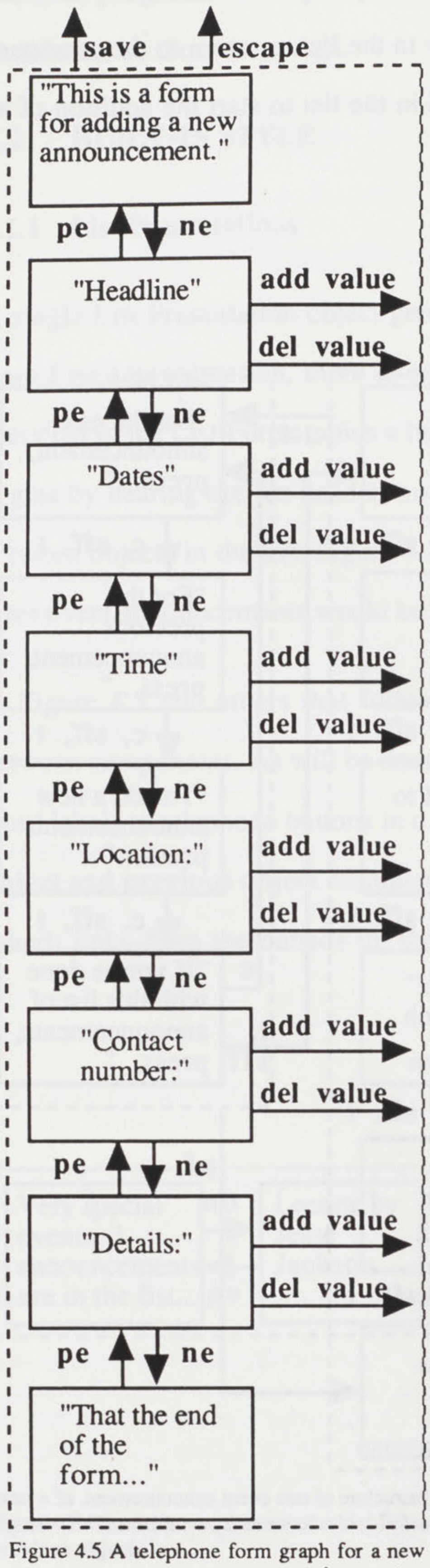


Figure 4.5 A telephone form graph for a new event announcement. **pe = previous entry blank; ne = next entry blank.**

prompts for available commands and then skips through them. In addition, **smart fast forward** and **timeout** transitions skip through all of the nodes in linear sequence. Thus, if a caller simply waits, all of the fields will be played back in order, followed by all of the command prompts.

5.1.3 Telephone Forms

While listening to the announcements, a caller can press **add** to initiate execution of an List Action. The interpreter creates a new object of type Event Announcement and uses the Field Edit Formats to determine any initial values for fields. Then, it creates a telephone form, which is a subgraph for editing the object, and calls that subgraph. The subgraph has one component for each field of the object that the Field Edit Formats say to include in the form, plus a header and a footer, as shown in Figure 4.5. Of course, the components in the diagram have substructure analogous to the substructure given for list nodes above in Figure 4.4, with prompt nodes attached to each of the entry blank nodes.

5.2 STYLE PARAMETERS

A number of characteristics of the graphs generated are under programmer control. These include the mapping of commands to buttons, the advancement and selection mechanisms for menus, the generation of orientation cues and the generation of navigation prompts.

5.2.1 Command Mapping

As mentioned above, a global lookup table defines the mapping between commands and prompts. HyperVoice uses this table both to determine the buttons with which to label transitions between nodes and to determine the text of prompts for those transitions, thus ensuring consistency between the two. Table 4.6 shows a typical mapping. Notice that

Command name	**Button**	**Description of command**
Global		
rewind	5	Rewind by 5 seconds.
pause	8	Stop playing (any button restarts).
smart fast forward	#	Immediately execute the next timeout transition.
commands	0	Go to the prompts for available commands.
escape	*	Cancel data entry, go back to previous list presentation, hangup from top level.
enter special commands	00	Begin entry of special commands for List Jumps, cut, paste. 00 means press 0 twice with less than a half second between presses.
In Lists		
next object	9	Go to the beginning of the next object in the list.
previous object	7	Go to the beginning of the previous object in the list.
next field	6	Go to the next field of the current object.
previous field	4	Go to the previous field of the current object.
select	2	Either return the current object as a value (in picklists) or go to the list presentation associated with the current object.
add object	1	Add a new object, usually to the current list.
In Forms		
next entry blank	6	Go to the next entry blank in the form.
previous entry blank	4	Go back to the previous entry blank in the form.
delete value	3	Delete the last value entered in the current entry blank.
add/replace value	1	Replace all the values entered in the current entry blank.
append value	2	Add a new value or append to the recording in the current entry blank.
save form	9	Save the contents of the entry blanks and exit the form.
cancel form	*	Throw away the contents of the entry blanks and exit the form.
Miscellaneous		
confirm	1	Confirm a save form, cancel form, or hangup operation.
delimiter	#	End keypad input of variable length data.

Table 4.6 A typical mapping of commands to buttons.

previous object maps to **7**, which is to the left of **9**, the button that **next object** maps to. This keypad layout emphasizes the spatial metaphor of a list. Also notice that the table makes it easy to identify commands that share the same button. For example, the **next field** and **next entry blank** commands both map to the same button. The programmer can disable commands by assigning them to negative number buttons.

5.2.2 Menu Styles

List Formats include two parameters that determine the menu style. All menu styles allow callers to hear prompts for a number of options and select one of them. The styles differ in how callers advance through hearing the prompts, and in how they select options. There are three mechanisms for advancing through the prompts: waiting, pressing buttons (skipping), or both. There are also two mechanisms for making selections: numeric and positional. These two independent dimensions define a two-by-three matrix of potential menu styles, shown in Figure 4.7.

For example, the standard numbered menu style requires callers to wait through the recitation of prompts early in the menu in order to hear the prompts later in the menu.

		How to Select	
		Numeric	Positional
How to advance	Wait	Standard	Timed 1-button
	Skip	Stepped numeric	3-button
	Both	Combined	Timed 3-button

Figure 4.7 A matrix of menu styles. The bottom two rows are Skip and Scan styles since they permit callers to skip through the options at their own pace.

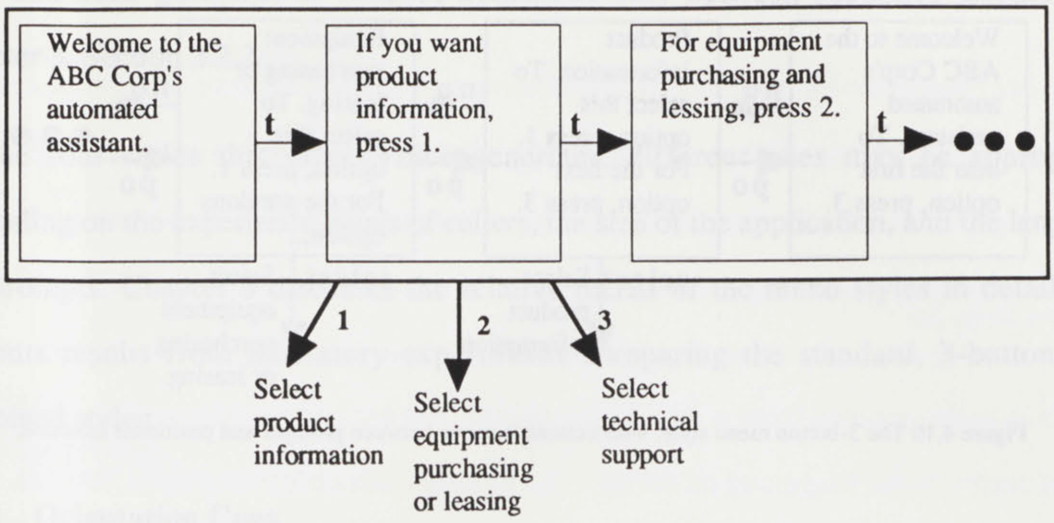

Welcome to the ABC Corp's automated assistant.
t
If you want product information, press 1.
t
For equipment purchasing and lessing, press 2.
t
●●●
1
2
3
Select product information
Select equipment purchasing or leasing
Select technical support

Figure 4.8 The standard menu style, with timeouts to advance and numeric selection.

Standard menus employ numeric selection: a caller can select an option by pressing the number associated with it, even if that option is not currently playing. Figure 4.8 shows a state-machine representation of this menu style.

The combined menu style allows but does not require callers to wait through the recitation of the prompts. They also have the option of pressing the smart fast forward button to skip through the options more quickly. Figure 4.9 shows a state-machine representation of this menu style.

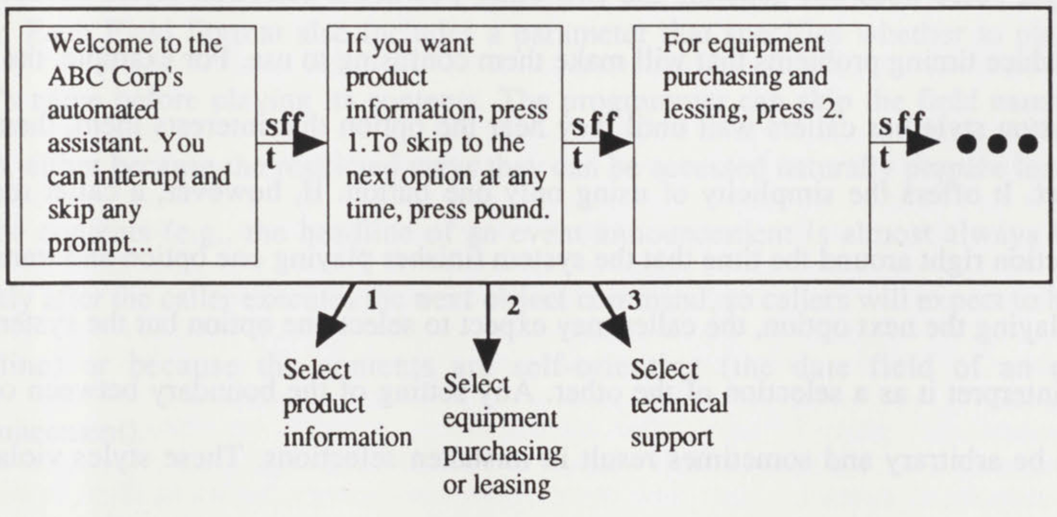

Welcome to the ABC Corp's automated assistant. You can intterupt and skip any prompt...
sff
t
If you want product information, press 1.To skip to the next option at any time, press pound.
sff
t
For equipment purchasing and lessing, press 2.
sff
t
●●●
1
2
3
Select product information
Select equipment purchasing or leasing
Select technical support

Figure 4.9 The combined menu style, with both timeouts and smart fast forward to advance between objects, and numeric selection.

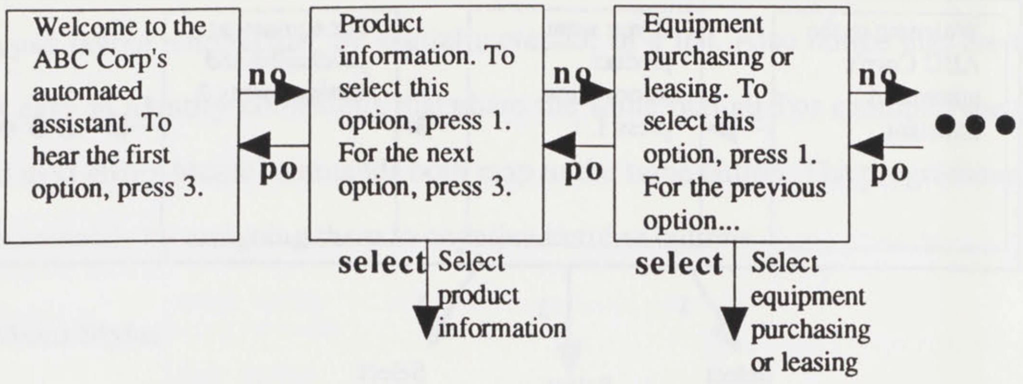

Welcome to the ABC Corp's automated assistant. To hear the first option, press 3.
no
po
Product information. To select this option, press 1. For the next option, press 3.
no
po
Equipment purchasing or leasing. To select this option, press 1. For the previous option...
no
po
●●●
select
Select product information
select
Select equipment purchasing or leasing

Figure 4.10 The 3-button menu style, with buttons to move between prompts and positional selection.

The 3-button menu style follows the structure of Figures 4.3 and 4.4, but switches to positional selection. Positional selection lets callers select only the current option, in contrast to numeric selection which allows callers to select any option while listening to any other option. This style also requires callers to press buttons to advance through the prompts, by pressing **next object**; no timeout transitions are available.

In addition to the three styles shown above, HyperVoice will generate the stepped numeric style, which requires button presses to advance but still uses numeric selection. It removes the timeout transitions from Figure 4.9.

HyperVoice does not generate the two other positional selection styles because they introduce timing problems that will make them confusing to use. For example, the timed 1-button style has callers wait until they hear the option that interests them, then press **select**. It offers the simplicity of using only one button. If, however, a caller makes a selection right around the time that the system finishes playing one option and transitions to playing the next option, the caller may expect to select one option but the system may misinterpret it as a selection of the other. Any setting of the boundary between options will be arbitrary and sometimes result in mistaken selections. These styles violate the

Skip and Scan guideline of timeout transitions only between subnodes discussed in Chapter 3, Section 2.2.2.

Of the four styles that HyperVoice generates, different ones may be appropriate depending on the experience levels of callers, the size of the application, and the length of the prompts. Chapter 5 discusses the relative merits of the menu styles in detail, and presents results from laboratory experiments comparing the standard, 3-button, and combined styles.

5.2.3 Orientation Cues

As discussed in the previous chapter, orientation cues help callers keep track of their locations in documents, but take time away from listening to the contents of the documents. Because there is no general solution to this tradeoff, HyperVoice includes some parameters that allow programmers to choose whether to include orientation cues in particular situations.

Field Names

Field names are one optional orientation cue in presenting the contents of an object. An Item Format includes Field Formats that specify which fields to include and in which order. Each Field Format also includes a parameter that specifies whether to play the field's name before playing its contents. The programmer can skip the field names for fields, either because the restricted ways they can be accessed naturally prepare listeners for the contents (e.g., the headline of an event announcement is almost always heard directly after the caller executes the **next object** command, so callers will expect to hear a headline) or because the contents are self-orienting (the date field of an event announcement).

Item count

Each list header includes an optional statement of how many items are in the list. The programmer specifies whether to include this information through a parameter of the List Format.

Return from list

When a caller returns from a list (by executing the **escape** command), the system can either return to the beginning of the last list, or to the last object visited in that list. In the latter case, the caller first hears an orientation node that says the current position in the list (e.g., "returning to item <n> in") and then the header prompt for the list. The programmer specifies as a parameter of the Select Action for the object whether to return to the beginning of the list or to the current item.

5.2.4 Prompt generation

The programmer can tailor many of the prompts for commands to reflect the contents of the information being presented. Some of the tailoring requires entry of whole sentences, while in other cases the programmer specifies a single word that a pre-processor uses to generate complete instructions.

List navigation prompts

The List Format includes a parameter 'Name for Items' that generates several prompts for navigating through a list. Suppose that the programmer sets this parameter to the text, "announcement". One prompt becomes, "For the next announcement, press 9." Another becomes, "For the previous announcement, press 7." "To go back to the last announcement, press 7," plays after the list footer. A fourth becomes, "To add a new announcement, press 1." Note how these phrases encourage callers to think in terms of actions on a list of event announcements, the external metaphor, rather than in terms of state-machine layer actions.

In the List Format for a 3-button style navigation menu, the 'Name for Items' would be, "option". This would generate navigation prompts like, "For the next option, press 9" and "For the previous option, press 7." Again, notice how these prompts encourage callers to think in terms of the metaphor of a menu of options.

Selection prompts

There are three kinds of selectable lists, requiring different prompts for selection. One kind is the picklist, used in telephone forms, which treats any list of objects as a set to select from. The second is the navigation menu, a list of List Presentations, which allows a caller to choose which List Presentation to visit next. The selection prompt for both these types of menus is, "To select this <Name for items>, press <n>" In the case of navigation menus, for example, where the 'Name for items' field is set to, "option", the prompt would be, "To select this option, press <n>" The number <n> is also determined automatically. With positional selection, it is the button for the **select** command. With numeric selection, it is the position of the option in the list.

The final kind of selectable list occurs with objects whose fields contain pointers to other, related List Presentations. Selection of an object causes navigation to its linked List Presentation. For example, in the teachers' curriculum line application, each lesson plan contained a pointer to a List Presentation for questions about the lesson. This kind of selection benefits from a prompt that explains the nature of the relationship between the selected object and the destination List Presentation. In the curriculum line example, it would be, "For questions about this lesson plan, press <**select**>." To generate this selection prompt, the programmer enters the phrase, "questions about" as a parameter in the Select Action object.

List header prompt

Each List Presentation includes 'title' and 'description' prompts. These are used to generate the contents of the header node. Response lists, however, are created automatically without asking a caller to record the 'title' and 'description' prompts. In that case, HyperVoice generates the contents of the header node from the 'Response List?' parameter of the List Format. In the list header node, the interpreter includes the text from the 'Response List?' field, followed by the contents of a headline field from the object that the list contains responses to. For example, in the opinion forum applications, when a caller selects the responses to a comment, the header says, "Responses to" and then plays the headline of the original comment.

Form header prompt

The header node for a form begins with the prompt, "This is a form for adding a new <type name>. Think of it like a paper form, but instead of writing the information in entry blanks, you'll record it or enter it using the buttons on your telephone." In this prompt, the name of the object type being edited is substituted for <type name>. After this initial prompt, the form header plays an optional prompt that the programmer specifies in full as part of the Edit Format.

Field names and descriptions in forms

Each entry blank in a form begins by playing the field name, which is programmer specified. If the field is currently empty, then the entry blank node also contains an optional 'field description' prompt that the programmer specifies as part of a Field Edit Format. Pilot testing of different field description prompts suggests that a descriptive prompt is better than a prescriptive one. For example, a good descriptive prompt would be, "This field should contain a typed-in date indicating when the event will occur." A prescriptive prompt might be, "Please type in the date on which the event will occur."

The apparent trouble with the prescriptive style is that it suggests that entry of a date is required and that the caller can begin to type in the date right away. In fact, the caller has to press the **append value** button first to indicate that he or she wishes to enter the date rather than skipping over that entry blank.

5.3 CONFORMITY TO SKIP AND SCAN DOCUMENT PROPERTIES

Generation of a Skip and Scan audio document is a partnership between HyperVoice, the application programmer, and the contributors who enter information over the phone. The HyperVoice interpreter generates network topologies that automatically satisfy many of the Skip and Scan requirements. Other topology and node properties require the programmer to make wise choices. Both the system and the application programmer make it easier for callers to add information in ways that conform to the Skip and Scan document properties, but some responsibility still rests on the information contributor. Below I review the document properties and the distribution of responsibility for satisfying each.

5.3.1 Hearing a Small Portion

One topic per node. HyperVoice automatically creates one node for each field during playback. It is up to the programmer to design object types that contain just one topic per field, and up to contributors to follow fill in fields with the appropriate information.

Summary nodes. The list header nodes summarize what the caller can expect to hear in the list. Typically, the prompts for list headers are either written by the application designer, or generated automatically based on the 'Response List?' parameter of the List Format. It is also the responsibility of the application programmer to include a field in each object type that will contain enough summary information to allow listeners to decide whether to hear the rest of the object.

Orientation cues. Because of the network topologies that HyperVoice generates, some of the nodes will be self-orienting, such as the beginning of a new event announcement. The application programmer can also specify several parameters, as described above, that determine whether to include certain orientation cues, and how to generate the orientation prompt in the form header.

Progressive Disclosure. Contributors are entirely responsible for ensuring progressive disclosure within each recording that they make.

5.3.2 Fast Navigation

Regularities, Metaphors and Orderings. HyperVoice always generates list and form structures that have identifiable roles (e.g., header, item, footer). HyperVoice provides high level primitives that make it easy for the programmer to express orderings. The programmer is responsible for some aspects of the prompts and for choosing a mapping of commands to buttons that emphasizes the spatial aspects of the list and form metaphors.

Short paths to frequently accessed nodes. The application programmer is responsible for arranging objects into list presentations, with appropriate links from objects to other list presentations, so that there will be short paths to frequently accessed nodes.

Timeout transitions only between subnodes. HyperVoice generates timeout transitions in only two places, and both satisfy the rule of using them only between subnodes of a virtual node. Timeout links advance between nodes for the fields of an object, but these are subnodes in the sense defined in the previous chapter: the available commands do not change between fields. Similarly, timeout links advance between the navigation prompt nodes, but these, too, are subnodes of the overall information node.

Adding Information. HyperVoice regenerates state-machine representations when contributors add new information. It automatically fills in links and reuses navigation prompts from the list presentation to which a new object is added. Programmers specify where to add new objects based on the contributors' locations when they initiated entry.

Explicit actions to record, review, and save. The form graphs that the HyperVoice interpreter generates all satisfy this property. Callers can press **next object** to skip over any entry blank that they do not wish to fill in. Adding new values, and in particular recording, are never initiated with timeout links.

New nodes separated from old nodes. The application developer specifies where new nodes will be added. Unfortunately, HyperVoice does not give the programmer much to work with, providing metaphors that either allow chronological access to data, or access data by semantic relationships such as response links, but not a mixture of both.

6 LIMITATIONS AND FUTURE RESEARCH

While the HyperVoice language is expressive enough to implement a wide range of cooperative work applications, several limitations have become apparent, both in the functionality it provides and the interfaces it generates.

6.1 AD HOC QUERIES

While the sorting, filtering, and item presentation formats can generate multiple presentations of a set of information objects, the presentation formats have to be pre-defined. It would be nice to allow a caller to specify at run-time ad hoc queries and ad hoc arrangements of the information in lists. Telephone forms may provide a good starting point for exploration of how callers can specify those queries.

6.2 TEXT FIELDS

HyperVoice does not include text as a data type. Although they are cumbersome, there are several techniques for entering text over the phone [Detweiler, et al. 1990, Fast and Ballantine 1988, Marics 1990]. A text-to-speech synthesizer could play back the contents to other callers.

6.3 CONTRIBUTOR-DEFINED AND MACHINE-GENERATED SEGMENTATION

HyperVoice developers always pre-define the structure of information objects. Contributors cannot add additional structure that listeners can then use as a basis for navigation. Often, however, this facility would be useful, since contributors may be able to structure messages even when designers were unable to predict that structure in advance. One simple approach that researchers are already experimenting with is to provide segment markers that contributors can insert by pressing a button while recording [Degen, et al. 1992]. Computer analysis of a long recording may also yield useful segmentation automatically, based on pauses or on changes of speaker [Hindus and Schmandt 1992]. Then, listeners can press buttons to jump between contributor-defined segments instead of between pre-defined fields of objects.

6.4 NEW PRESENTATION METAPHORS

The List Format is good at presenting information as lists of objects with links between some objects and other lists. While this metaphor is flexible enough to accommodate what would visually be presented as lists, as calendars, and as networks, there is at least one other presentation metaphor that would be useful, and probably others. The obvious one is the matrix, or grid. To use a grid metaphor, the underlying data may be a grid, or it may be a list with a crosstab function defining the rows and columns of the matrix. That is, the row values are all the different values that appear in one of the fields of the objects, and the column values are all the different values that appear in another of the fields. For example, in a meeting scheduling application, there might be a list of responses that various people made to several suggested meeting times. In a matrix presentation of these objects, there would be one row for each person who had responded to any suggested time, and one column for each suggested time that anyone responded to. Each cell would present one person's response to one time suggestion.

6.5 ABSTRACTION MECHANISM

The HyperVoice language needs an abstraction mechanism. After writing several programs I found it convenient to define two macros, for menus and for creating response lists, which are presented in Appendix A. There is currently no way to create new macros in the language without revising the interpreter.

6.6 MIXING CHRONOLOGICAL AND CONTENT ACCESS

As described above, HyperVoice does not provide the application programmer much assistance in separating the new nodes from the old ones. Master lists that allow system administrators to have chronological access to the information objects may be the seed of a more general solution. This general solution would allow all callers to easily switch back and forth between chronological access and more content-related access.

7 RELATED RESEARCH

7.1 TELEPHONE TOOLKITS

Previous researchers have used variants of the state-machine abstraction layer to specify telephone-based interfaces. None of these systems, however, attempted to generate state-machine programs from higher-level specifications.

The first published toolkit for telephone applications that uses a graph representation [Richards, et al. 1986] was developed at IBM as part of the Olympic Messaging System project [Gould, et al. 1987]. Several commercial toolkits (TFLEX [Magnum 1990] and PhonePRO for the MAC, TRT and others for the PC) take the graph language as their basis, though they all use slightly different terminology. Some of the toolkits take advantage of visual representations of graphs and provide direct manipulation tools for modifying them (TFLEX and PhonePRO).

One research project has explored the use of grid-based spatial conventions in addition to explicit lines between graph nodes in visual representations of programs [Repenning and Sumner 1992, Sumner, et al. 1991]. For example, if the boxes for two graph nodes are arranged horizontally and share an edge, there is an implicit timeout link from the left node to the right node. Similarly if the boxes for several nodes are arranged vertically, they are the possible destinations of the links from another graph node. These conventions make the two-dimensional representations of state-machine programs easier to understand visually.

7.2 APPLICATION LAYER LANGUAGE

Other researchers have identified semi-structured objects and presentation formats as useful abstractions for specifying visual information sharing applications. The OVAL system [Lai, et al. 1988, Malone, et al. 1988, Malone, et al. 1992] includes screen-based tools that allow end-users to specify applications in terms of these abstractions. For example, end-users can request that a list of objects be presented in a tabular view and can make menu selections that specify how to sort them and which fields of each object to display.

HyperVoice makes several improvements on the specification language used for OVAL applications. First, OVAL has only two data types, pointers and strings, with a limited ability to interpret strings as dates. HyperVoice includes dates and numbers as data types, which allows sorting and filtering operations to use comparison operators that are appropriate for those data types (for numbers, 2 < 10, but for strings, “2” > “10”). Second, OVAL does not reify presentation formats as separate objects. Thus, two lists of objects cannot share the same presentation format nor can one list use more than one presentation format at a time. Finally, OVAL specifications for editing new objects do not include validity checks on new values and are much more limited in their specification of initial values. For example, OVAL tools do not allow the automatic creation of embedded objects, objects that point to other newly created objects.

In addition, OVAL specifies filtering operations more flexibly but less elegantly than HyperVoice. OVAL does not include filters in specifications of presentation formats: instead, agents perform filtering operations to generate new lists of data that are subsets of existing lists. This adds additional flexibility. For example, two agents can add subsets of several lists to a single destination list. When the additional flexibility is not needed, however, it adds unnecessary complications. It may be difficult to tell that an agent is performing a simple filter operation rather than some more complex operation, which makes programs more difficult to understand and to debug.

7.3 AUTOMATIC GENERATION OF SCREEN INTERFACES

The OVAL system automatically generates several kinds of screen presentations of lists of information objects, including forms, tables, matrices, networks, and calendars. In all these cases, users specify only a few parameters, such as which fields to include. The system automatically chooses the widgets to use in presenting the information and how to lay out those widgets on the screen. For example, consider the 'form' view of a single object. OVAL places the field names on the left-hand side of a window and places the fields' contents on the right-hand side. When a field has a list of alternative values, OVAL automatically attaches to the field name text a pull-down menu containing those values. All of the widget selection and layout decisions are made at run time to accommodate changing data objects and to allow users to dynamically specify the way they want to view information.

Several other recent systems also automatically translate semi-structured application data into interactive screen displays [de Baar, et al. 1992, Hayes, et al. 1985, Johnson 1992, Kim and Foley 1990, Olsen 1989, Olsen, et al. 1992, Szekely 1990, Szekely, et al. 1992, Vander Zanden and Myers 1990, Wiecha and Boies 1990]. All these systems assume that a separate application program will specify which data to present at which time. In particular, they do not include constructs that filter and sort lists, set privileges for who can add to lists, set initial values and provide validity checks for fields of objects. Their application layer abstractions, then, are not particularly interesting.

Instead, these systems are interesting because their automatic user interface generation tools encode style guidelines explicitly as rules of translation from information objects to widget instances. The action of the translator is determined by a set of explicit style rules, analogous to the style parameters in HyperVoice presentation formats. For example, ITS [Wiecha, et al. 1989] and DON [Kim and Foley 1990] style rules can decide how to organize a large set of options into a hierarchical menu, based on such properties as which objects the options act on. ITS and Humanoid [Szekely 1990] separate the generation of a widget tree from determining the graphical layout of the widgets, and do not parameterize the graphical layout procedures in terms of style rules. DON extends the rule-based system to make decisions about the graphical layout as well.

By contrast, HyperVoice provides fewer parameters that encode different interface styles. The two most important two style parameters are the advancement mechanism and the selection mechanism for menus. A few other style parameters determine orientation cues and the generation of prompts.

Different sets of rules can encode different interface styles. Humanoid can even determine the appropriate set of style rules to apply, based on attributes of the object being presented. For example, it might apply a different set of style rules to menus depending on whether the menu allows single or multiple selections. This would be analogous to HyperVoice automatically choosing the audio menu style from among the four it can generate; in fact, HyperVoice requires the application programmer to set the style explicitly for each menu.

HyperVoice, on the other hand, addresses two issues that the other systems do not. First, HyperVoice presents information via sound rather than sight. As emphasized in chapter 3, good audio interfaces provide actions for the fine-grained changes of attention that can be accomplished in visual interfaces with eye gaze shifts. The biggest difficulty in automatic generation of visual interfaces is graphical layout that is aesthetic and maximizes the value of eye gaze shifts. The biggest difficulty in audio interfaces, on the other hand, is the provision of a predictable set of fine-grained navigation actions that accomplish the same changes of attention.

The second unique feature of HyperVoice is that it generates interfaces that are completely self-explanatory. All of the other systems assume a minimal level of user familiarity with using a mouse to make menu selections and navigate in dialog boxes. HyperVoice, on the other hand, automatically generates the text of prompts that tell callers how to make selections from menus and how to fill out forms.

8 CONCLUSION

HyperVoice lets designers, programmers, and users speak the same language. Lists, menus, forms, and login procedures are useful abstractions for design, as demonstrated by the participation of non-programmers in the design of some of the applications described in Chapter 2. HyperVoice reifies these abstractions as primitives of a programming language. The HyperVoice interpreter then generates state-machine graphs and navigation prompts that encourage callers to use those abstractions as metaphors for navigation and data entry. This helps developers to specify applications that collect, route, distribute, and arrange information.

It helps developers to route and to arrange information in ways that benefit listeners. Login procedures can start callers on different initial List Presentations. The same objects can be presented in several different ways within an application, using different List Formats and Item Formats.

HyperVoice also helps applications developers to elicit information from contributors in a format that enables these selections and arrangements. The developer uses the location from which a contributor initiates addition of information to determine the type of object to add and which lists to add it to. To help contributors enter the symbolic fields needed for sorting and filtering, developers specify initial values, picklists, and validity checks. Developers can encourage shorter recordings by breaking messages into separate fields.

Overall, then, HyperVoice contributes to a partnership among developers, contributors, and listeners. Developers specify audio document structures that accommodate additions from contributors, who annotate recordings with just enough symbolic information to enable HyperVoice to arrange the information for easy scanning by listeners.

5 AUDIO MENU STYLES

Audio menus are used frequently in HyperVoice applications and in commercial interactive voice response (IVR) applications. As described in Chapter 4, Section 5.2, HyperVoice can generate four menu styles. They vary along two dimensions, the method of selecting an option and the method of advancing through the prompts for options, as shown in Figure 5.1. The Skip and Scan guidelines suggest that callers will make selections more quickly if they can skip through the options at their own pace. Callers who are not yet familiar with how to skip, however, will find automatic advance helpful. The guidelines also suggest that there is a tradeoff between the short paths to frequently selected options that numeric selection allows and the predictable selection button that positional selection provides.

This chapter compares user performance with three styles of audio menus. The standard style pairs automatic advance with numeric selection. The other two styles both allow skipping. The Combined style combines skipping with automatic advance and numeric selection. The 3-button style does not include automatic advance; it pairs skipping with

		How to Select	
		Numeric	Positional
How to advance	Wait	Standard*	Timed 1-button
	Skip	Stepped numeric	3-button*
	Both	Combined*	Timed 3-button

Figure 5.1 The matrix of menu styles. Asterisks mark hose tested in the experiments.

positional selection. The fourth style that HyperVoice generates, Stepped Numeric, was omitted because of its similarity to the combined style, but may be worth including in future studies.

Two laboratory experiments confirm that the optimal menu style depends on callers' level of experience with the menu style and with the contents of particular menus. For the first few trials, subjects made selections from standard menus more quickly than with the two skipping styles. Soon, however, performance with the skipping styles equaled or surpassed the standard style. If skipping menu styles were widely adopted, even first-time callers to applications will be able to skip effectively. It is not yet clear, however, which skipping style should be adopted.

1 PREDICTIONS: EFFECTS OF MENU STYLE FAMILIARITY AND MENU CONTENTS FAMILIARITY

This section hypothesizes the effects of three variables on selection time, as summarized in Figure 5.2. The columns divide users into those who know how to skip between options and those who do not yet know how. Each row indicates a level of familiarity with the contents of the menus. Callers may be so unfamiliar with the menu contents as to not even recognize the correct option immediately upon hearing it. For example, the first time someone calls an audiotex application, he may need to listen all the way through a category name and then think about it for a few seconds before deciding whether it is the right option to choose. The second familiarity level is when callers can quickly accept or reject any particular option, but cannot recall the position in the menu of the desired option. This will be typical of occasional users of audiotex systems. It may also apply to first-time callers if the menu options are unambiguous (e.g., north, south, east, and west) or the caller is looking for an exact match with a known target (e.g., a name in a directory listing). At the third familiarity level, callers know the positions of desired options in

		Style Familiarity	
		Don't know how to skip	Know how to skip
Menu Contents Familiarity	No immediate recognition	Standard > combined > 3-button	3-button > combined = standard
	Recognition but not recall	Standard > combined > 3-button	3-button > combined > standard Advantage of 3-button and combined over standard increases with target position
	Recall	standard = combined > 3-button	standard = combined > 3-button

Figure 5.2 A matrix of familiarity levels. Hypotheses about the relative performance of the different menu styles appear in the cells for the different familiarity levels. Standard > combined indicates better performance (faster selection) with standard menu.

menus. This will be true of very frequent users of a system, or of users who consult written documentation to make selections.

First consider the left column, where callers do not know how to skip. In the top cell, they have to think about each option before accepting or rejecting it. Callers in this cell will take longer to listen to the options in a menu with the combined style than with standard menus, because of the extra prompts that tell callers how to skip. The 3-button style will fare even worse, because callers will have to listen to prompts for how to hear the next option and then execute those actions, which will be unfamiliar to them. The same reasoning applies even if callers can quickly recognize desirable options, as in the second cell in the column. The hypothesis varies slightly in the bottom cell, where callers know the positions in the menus of the desired options. Again, 3-button menus will fare worse than standard menus: because of positional selection, callers will still need to advance through the options, which will require learning the navigation buttons. Here, however, callers should make selections from combined menus about as quickly as with standard menus, because they can enter numeric selections without listening to any of the prompts.

The top right cell is for callers who are familiar with skipping menu styles, but are calling a new application whose menu options cannot be accepted or rejected without some thought. The very existence of this cell is a hypothesis: it assumes that once people have learned how to skip in one telephone application, they will be able to transfer that skill to another application. This assumption was tested in the second experiment.

In the top right cell, callers will need to listen all the way through most menu options in order to make judgments about whether they are appropriate, so there will be little or no advantage to skipping. On the other hand, there should be little or no penalty from the prompts that tell people how to skip, because callers know how to press buttons that skip over the prompts. Positional selection should have a slight advantage over numeric selection, because of its simplicity: positional selection avoids the cognitive load of keeping track of the numbers associated with menu options. Hence, the 3-button style should allow slightly faster selection than the other two.

The middle cell of the right column represents situations where callers can quickly accept or reject menu options and they already know how to skip. Here the advantages of skipping are large, so both skipping styles should lead to better performance than the standard style. Again, the simplicity of positional selection should favor 3-button over combined menus. Moreover, the advantages of the skipping styles should increase the more items there are in the menu and the longer the prompt for each item is.

Finally, in the bottom right cell, callers know the positions of target options in the menus. Numeric selection can bypass listening to the prompts altogether, thus negating the advantage of being able to skip through the options. Hence, standard menus should do as well as combined menus. Both should do better than 3-button menus, which will require many more keypresses because of positional selection. Anecdotal evidence from the teachers' curriculum line supports this hypothesis. That application used 3-button menus. The top-level menu had seven options, the last of which contained a general list of comments. The head teacher checked that list for new comments every day. After a few weeks she asked why she couldn't just press a number to select the general list of comments. In discussing potential redesigns of the system after the initial field trial, she insisted on numeric selection even when I argued for the advantages of positional selection.

2 LABORATORY EXPERIMENTS

Two laboratory experiments validate some of the hypotheses described above. Both studies were conducted in collaboration with Robert Virzi and Don Ottens at GTE Laboratories. The first study compared standard and 3-button menus on a task that fits in the 'recognize but not recall' row. Results were reported in [Resnick and Virzi 1992]. The second study included combined menus as well, and varied the menu contents familiarity. Some of the results were reported in [Virzi, et al. 1992].

2.1 EXPERIMENT 1: SELECT A NAME FROM A LIST

The experiment compared 3-button menus to standard menus in a within-subjects design. Each subject selected target names from lists of between three and twelve names. Both younger (college-age) and older (near retirement age) subjects were run. Selection times and error rates were measured. After subjects had used both styles, they were asked which menu style they preferred.

The overall result was that, after an initial learning period of a few menus, subjects made selections faster with 3-button menus than with regular menus, and preferred 3-button menus overall. Error rates were low and not significantly different between the two styles.

2.1.1 Methods

Subjects

Two groups of subjects were run in this experiment. The first group was composed of 12 subjects recruited from a local university (mean age 23). A second group of 6 subjects was drawn from an older population (mean age 62). We chose this older population because past experience has indicated that older users tend to be resistant to new technology and to have greater difficulty using telephone-based interfaces.

Stimuli

A list of 100 names was randomly drawn from the telephone directory of a large corporation. Each name was presented as a first name followed by a last name exactly as it had appeared in the directory.

A total of 72 trials were prepared. Each trial consisted of a target name and from 2 to 11 distractor names, leading to list lengths of 3 through 12 names. The position of the target name in the list was a more important variable than the length of the list, as it turned out, because subjects made selections as soon as they heard the target names. The target name appeared in each of the 12 serial positions 6 times.

A random order was drawn for presenting the stimuli, and this same random order was used for both conditions and for all subjects. In other words, each subject performed the same set of 72 tasks using both menu styles. Half the subjects used 3-button menus first while the other half used standard menus first, which allowed us to check that subjects were not remembering the tasks when they switched to the new menu style.

A telephone interface was constructed that implemented each of the menu techniques. One female voice was used for all system prompts and a second female voice was used to present each of the names composing the lists. Users interacted with the systems from a telephone by pressing the touch-tone generating keys.

Procedures

Subjects were seated before a standard desk set telephone. The general experimental procedures were explained but they were given absolutely no instruction on how they were to interact with the system. Instead they were told that they were to imagine that they had called a company with an automated directory service. They were told to follow the directions given by the system and to select the target name. Half the subjects in each group were presented the standard method first, the other half of the subjects interacted with the skip and scan method first. Between conditions they were warned that the method of selection had changed, and that they should attend to the instructions presented by the system.

Prior to the start of each trial, users listened to a name repeated over the telephone handset. This was the target name for the trial and it also appeared on a printed card next to the telephone as a memory aid. Users were told to press any key on the keypad when they were ready to begin the trial. Timing started when this key was pressed. In the skip and scan method, subjects started on the header node, which contained a set of instructions[3]. No instructions were required for the standard method as the prompt completely contained the information required to complete the task. After each trial, users were told whether or not they had selected the correct name on the previous trial and then the next target name was announced.

---

[3] The instructions, if not interrupted, took 15 seconds to recite. The exact text was as follows: "<n> names are in the list. Scan through the names using 9 to skip ahead and 7 to skip backward. It's OK to interrupt the spoken voice at any time. Select a name by pressing 1. For the first name, press 9."

After exposure to both systems, an overall preference question was asked, followed by an open-ended interview regarding the good and bad points of the two methods.

2.1.2 Results

Results are first presented for the group of 12 younger subjects, followed by the results for the 6 older subjects.

Younger Group

In Figure 5.3 the mean correct reaction times for the two conditions are shown as a function of target position. The best-fitting regression lines are superimposed. An Analysis of Variance (ANOVA) was calculated with the factors of Condition (standard menu method vs. skip and scan) and Target Position. The ANOVA confirms what the figure reveals. Overall, subjects were faster with the skip and scan method (F(1,11) = 83.417, p<.001) and were faster when the target name was earlier in the list (F(11,121) = 140.572, p<.001). Moreover, the interaction term was significant (F(11,121) = 14.685,

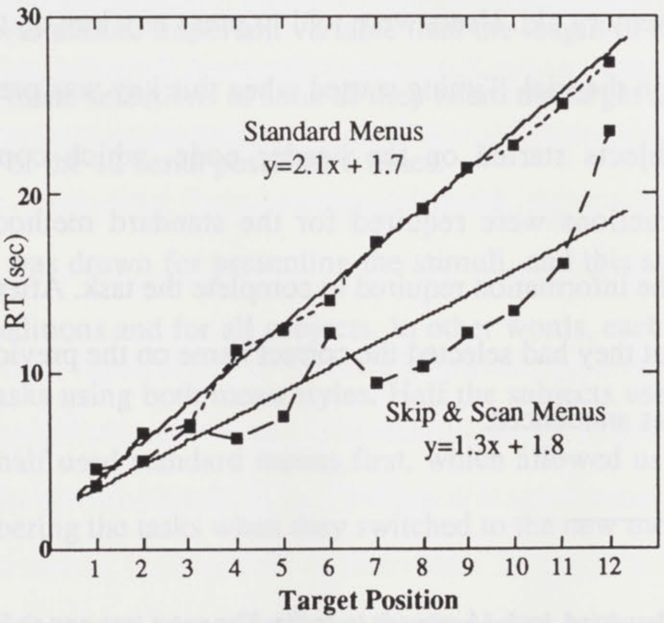


Figure 5.3. Mean correct selection time is shown as a function of target position for the younger subject population. The regression equations appear next to each menu style.

p<.001) showing that the advantage for the skip and scan method is greater as the target name appears later in the list. This is not surprising as in the standard method subjects had to wait for the target item while in the skip and scan method users could jump forward in the list based on a match with the first name.

We were interested in learning effects as well as overall performance. The trials were matched (i.e., each subject performed the same set of 72 trials in both conditions and the n^{th} trial in both conditions contained identical target names and lists of distractor names). As a result, we were able to calculate on a per subject basis two statistics that measure the learning effect. One statistic, the *crossover point*, was defined as the first trial on which the user was faster with the skip and scan interface. The other statistic, the *divergence point*, was defined as the beginning of the first run of five trials on which the user was faster on each trial with the skip and scan interface. The first statistic, the crossover point, had a mean value of 4.7 trials, a median of 3.0 trials, and a range of between 2 and 12 trials. The second statistic, the divergence point, had a mean value of 10.1 trials, a median of 6.0 trials, and a range of between 2 and 38 trials. Taken together, these results suggest that performance with the skip and scan menus surpassed performance with the skip and menus fairly rapidly.

Error rates were low. In the skip and scan condition, errors were made on fewer than 1% of all trials. For the standard method, errors occurred on just over 2% of the trials. Most of these errors occurred on trials in which the user had to press two keys to make a selection (e.g., item number 10) when the second key was not pressed before the timeout so that the system interpreted the selection as item number 1.

When asked which system they preferred overall, all 12 subjects expressed a strong preference for the skip and scan method over the standard method (p<.001 by sign test). When probed as to why they preferred it, users stated that they thought it was faster, more efficient, and put them more in control.

Older Group

In Figure 5.4 the mean correct reaction times for the two conditions are shown as a function of target position with the regression lines superimposed. An ANOVA was performed with the factors of Condition (standard menu method vs. skip and scan) and Target Position. Unlike the younger subjects, the difference between the two methods was not reliable (F(1,5) = 1.526, p>.10). However, they were faster when the target name was earlier in the list (F(11,55) = 59.492, p<.001) and the interaction term was significant (F(11,55) = 4.374, p<.001). For this older population, the skip and scan method was slower when the target name was early in the list, but there was a small advantage when it was later in the list.

In general, the learning effect for this population was much more dramatic. As with the younger subjects, two statistics were calculated for each subject. The mean crossover point for the older subjects was 7.5 trials, the median was 12.5 trials, with a range of between 1 and 15 trials. The mean divergence point was at 21.0 trials, with a median of

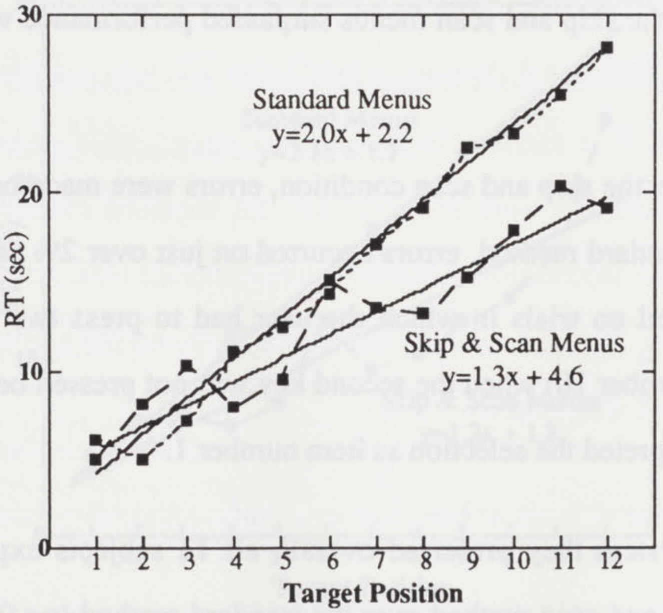


Figure 5.4 Mean correct reaction time is shown as a function of target position for the older subject population. The regression equations appear next to each menu style.

13.5 trials, and a range of between 5 and 56 trials. When compared to the younger population, the older group clearly took longer to learn the new technique, primarily because of their resistance to interrupting the prompts.

Older subjects made considerably more errors than the younger subjects, with 9.6% and 10.4% errors for the skip and scan and standard methods, respectively. One common error in the standard menu condition was not pressing two digit numbers fast enough, so that the system selected item 1 instead of item 12. Another common error, especially in early trials, was to press the number associated with the name before the target. This occurred because some subjects associated the numbers with the names following them rather than the names preceding them. This could be rectified with longer pauses between names, although this would increase the menu selection time.

When asked which system they preferred, 5 of the 6 older subjects stated a preference for the skip and scan method ($p<.10$ by sign test). When asked for the reasons behind their preferences, all six indicated that they preferred the one that seemed to be fastest.

2.1.3 Discussion

The entire experiment corresponds to the menu familiarity level where callers can recognize but not recall target positions. That is, they can quickly accept or reject an option (a name) because they are searching for an exact match with a target, but they cannot predict the correct target position, because each trial consists of a different set of names. This corresponds to the second line of the hypothesis table. The early trials correspond to callers who do not know how to skip, and confirms that standard menus lead to faster selections than 3-button menus. The later trials fit in the right column of the hypothesis table, where callers have learned how to skip. The results confirm the hypothesis that 3-button menus yield faster selection times than standard menus, and that the advantage increases with higher target positions.

		Style Familiarity	
		Don't know how to skip	Know how to skip
Menu Contents Familiarity	No immediate recognition		
	Recognition but not recall	Standard > 3-button	3-button > standard Advantage of 3-button over standard increases with target position
	Recall		

Figure 5.5 A summary of the results from the first experiment.

This experiment also provided a rough measure of how long it takes subjects to learn to skip. For the younger subjects, surprisingly few trials were required. In retrospect the list header prompt used with the 3-button menus in this style was probably not optimal. Additional fine-tuning of the prompts in the 3-button style might shorten the learning period even further.

This experiment also included a measure of user preference. Reduced selection times might not lead to user satisfaction, after all. The subjective preference measure, however, confirmed that preferences were related to selection times, as users preferred the 3-button menus in all but one case, and even in that case, the subject indicated that she preferred the style that she thought was faster.

2.2 Experiment 2: Weather and News

The second experiment considered menu selection as part of a more complex task, varied menu contents familiarity as well as style familiarity, and tested the combined menu style in addition to 3-button and standard menu styles. It also tested the hypothesis that the skill of skipping through menu options will transfer to a new application.

Overall, 3-button menus yielded selection times as fast or faster than the others in all but the first few trials. Surprisingly, this result held even after callers had enough practice to learn the target positions of desired options. The transfer of skill to new applications was confirmed.

2.2.1 Methods

Stimuli

Two IVR systems were built for this experiment. One allowed users to determine the weather in over 60 cities around the globe. The second application provided news reports and general information. The tree structures of the two menu systems were identical, although, of course, the labels describing the options were different. The mean number of options per menu was 4.8 with a low of 3 options and a single menu with 12 options. Appendix B lists the complete menu tree for both applications.

Subjects were asked to find the weather in particular cities and to find news on particular topics. Each task required making two or three menu selections.

Pilot testing of the two applications was performed to ensure that the prompts were understandable and that users would make few errors on the tasks. Some refinement of the specific prompts for each technique was required. In particular, the original prompts for the combined menu style included prompts for the skip button only in the menu headers. Several pilot subjects learned to skip the menu headers but never tried to skip menu items. The revised design includes a prompt for skipping after the first item in each menu as well as after the menu header.

Subjects

A total of thirty-six subjects, in three groups of twelve, completed sets of tasks. Subjects were recruited at colleges in Waltham, MA. One group was assigned to standard menus, one to 3-button menus, and one to the combined menu style.

Procedures

Twelve tasks were created that required subjects to traverse the menu hierarchy to reach a terminal node (a city in the weather application or a topic in the news application). The sets of twelve tasks for the two applications were structurally identical. Only the names of the menu options were changed. The mean number of menu choices subjects had to make to reach the terminal nodes was 2.75, with a minimum of 2 and a maximum of 3 choices required. The mean path length (the sum, over all menus required for a given task, of the target positions of the correct options) for the 12 tasks used in each application was 8.75 with a minimum length of 4 and a maximum length of 13.

All subjects in the experiment completed 36 trials. For each subject, the first 12 trials were conducted using the Weather application, with the remaining 24 trials conducted using the News application. In the Weather application, the twelve tasks were assigned to each of the twelve subjects using a 12x12 Latin square[4], thus ensuring that each task was

[4]A Latin square is an nxn matrix with the following properties:

a) the numbers 1 through n appear in each row.

b) the numbers 1 through n appear in each column.

c) For every pair of numbers (k, l), l is in the cell directly to the right of k exactly once in the matrix.

completed once in each sequential trial position. The same is true for the first 12 trials of the News application. The last 12 trials of the news application consisted of a repeating block of 3 trials. Again, the set of 3 trials repeated in this block varied across subjects so that, over subjects, each of the 12 unique tasks appears equally often in each sequential trial position

Design and analysis

Twelve subjects used each of the menu styles, making this a between-subjects factor. We chose not to use a within-subjects design for the Menu Style factor because we wanted to examine learning over trials, and once a subject had learned to skip using either the 3-Button or Combined menu style, we felt that this behavior might be carried forward to the other, contaminating the learning effect. We selected three-trial subsets of the data, as described below, to represent particular classes of user experience. This Trial Block factor is within-subjects.

Performance over all 36 trials of the experiment was examined for each menu style to show the pattern of learning that occurred. These results are presented graphically.

The primary statistical analysis in the experiment was a 3x5 ANOVA, with Menu Style a between-subjects factor and Trial Block a within-subjects factor. Five discrete blocks of trials were examined in the ANOVA: (1) the first three trials of the weather application in which subjects are learning the menu technique and have low familiarity with the

---

The use of a Latin Square to assign tasks to subjects insures a) that each subject (row) completes all n trials and that b) in each trial position (column) every task is performed by some subject. Moreover, in case there are learning effects from a subject completing one trial directly after the other, c) each trial directly follows each other trial exactly once, for some subject. A 12x12 Latin square is included in Appendix B.

application content; (2) trials 10 through 12, in which subjects have gained some familiarity with the technique and the Weather application; (3) the first three trials of the second application in which subjects were already familiar with the menu technique, but were not familiar with the application content; (4) trials 10 through 12 of the second application, in which subjects were proficient with the technique and were also familiar with the News application; and (5) the last three trials of the experiment, for which subjects were experienced with the menu style, the application content, and the particular target, simulating expert use.

2.2.2 Predictions

The five trial blocks roughly correspond to cells in the hypothesis matrix, and hence the hypotheses should predict the relative performance of the three menu styles on these trial blocks. The relevant predictions from that matrix are repeated in figure 5.6.

The first block, trials 1-3, corresponds to the top left cell of the matrix. Subjects have not yet learned how to skip. The menu options require thought about whether options fit into categories rather than merely recognition of exact matches. For example, if asked to find

		Style Familiarity	
		Don't know how to skip	Know how to skip
Menu Contents Familiarity	No immediate recognition	Standard > combined > 3-button (block 1)	3-button > combined = standard (block 3)
	Recognition but not recall		3-button > combined > standard (blocks 2 and 4)
	Recall		standard = combined > 3-button (block 5)

Figure 5.6 Predictions about the second experiment.

the weather in Barcelona, it may take a little thought to decide whether to accept or reject "Asian cities" as a menu option. Standard menus should outperform 3-button menus, with combined menus in the middle. This ordering is determined by how little time is spent prompting listening to prompts for how to skip through the options.

By the second block, trials 10-12, callers should have learned how to skip and also should be familiar enough with the menus to quickly accept or reject options. Some of the callers may learn the positions of the options in the top menu, since this menu is repeated. Hence, this is a combination of the bottom two cells in the right column, but predominantly the middle cell, since only some of the subjects will have memorized the top menu options, and no one will have memorized the others. 3-button should outperform combined menus because of the simplicity of positional selection and both should outperform standard menus because of skipping.

In the third block, the first three trials with the news application, callers will again be unable to accept or reject options quickly. This block tests the hypothesis that the skill of skipping through options will transfer from the previous application. The predictions from the top right cell apply: 3-button menus should be somewhat better than the other two, because of the simplicity of positional selection.

The fourth block tests the same condition as the second block. 3-button menus should outperform standard menus, with combined in the middle.

The fifth block, trials 34-36, tests performance after subjects have practiced repeated trials. These trials are the fifth repetitions of each of three tasks. Subjects should have learned the target positions for most of the menus by this point, which places this block in the bottom right cell. Standard and combined menus should fare better than 3-button menus, because of numeric selection.

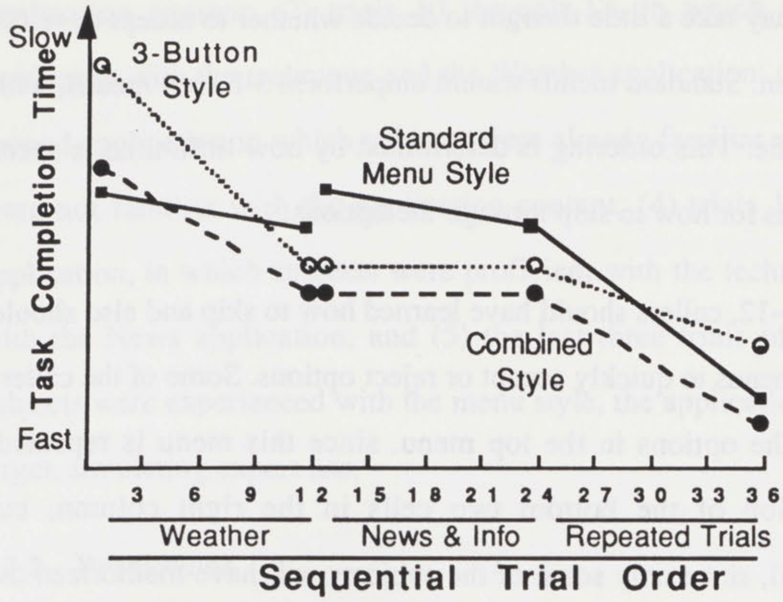


Figure 5.7 Predicted Results.

2.2.3 Results

Figure 5.8 shows the mean task completion times for the three menu styles as a function of sequential trials. Tasks were assigned to sequential positions using a Latin square. Across subjects, then, the same set of 12 tasks contributes to each trial position, except that trials not completed correctly were excluded from the analysis (compare to the predicted results shown in Figure 5.7).

A subset of the data was subjected to an ANOVA. Factors entering into the ANOVA were: Menu Style (between-subjects) and Block, as described above (within-subjects). The main effect of Menu Style was not significant (F(2,33) = 0.30; p > .05). The Block effect (F(4,132) = 121.47; p < .0005) reached significance. More importantly, a significant interaction between Menu Style and Block (F(8,132) = 7.10; p < .0005) obtained. This is shown in Figure 5.9.

The differences between the means in the first block were all reliable, with Standard menus producing fastest times and the 3-Button style producing the slowest times. By the

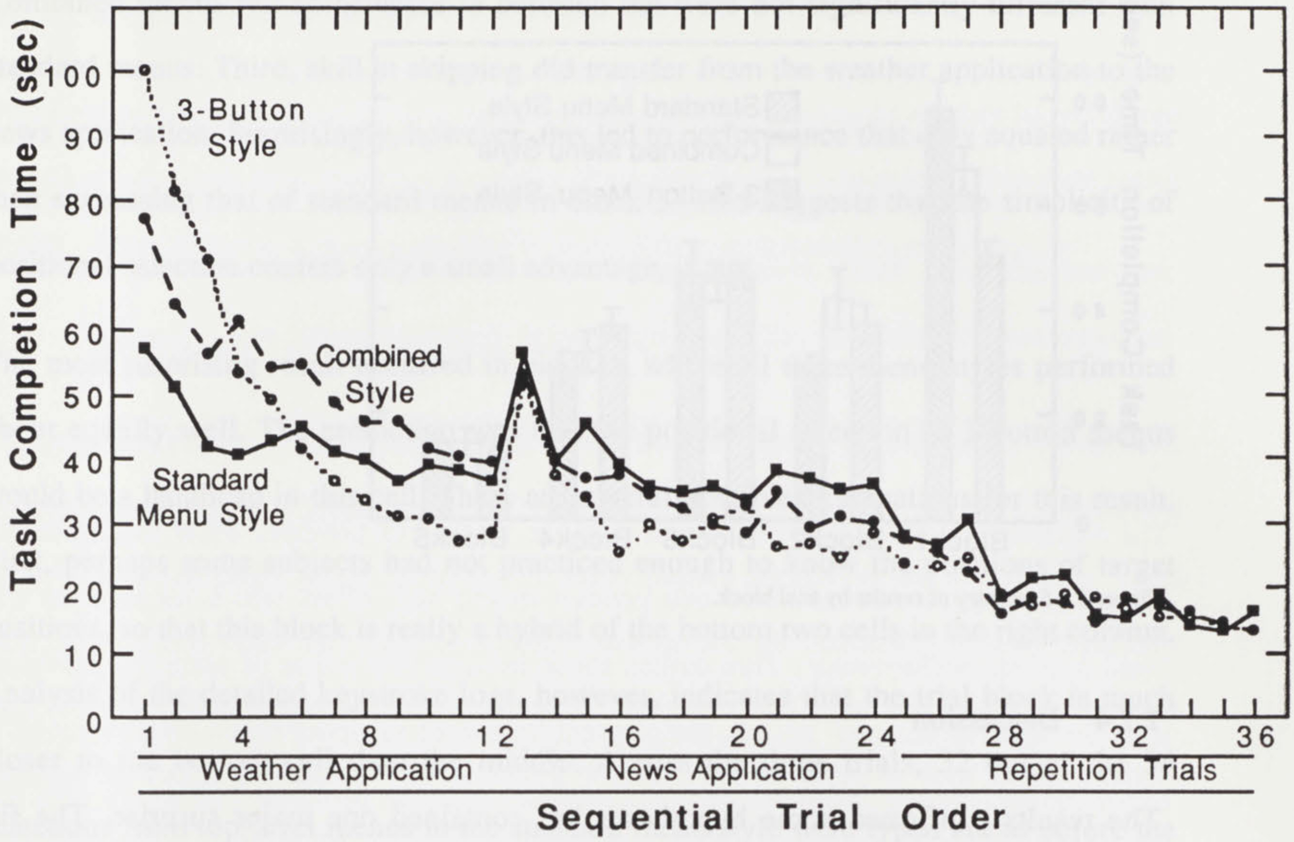


Figure 5.8 Results by trial for the three menu styles

second block of trials, however, the 3-Button style was leading to significantly faster times than the other two. In block 3, where a new application was introduced, the three styles produced equivalent times. In block 4, the speed advantage for the 3-button style relative to the Standard style reemerged. Finally, there were no differences among the techniques evident in block 5, in which subjects had extended practice with the specific tasks.

Error rates in the three conditions were low, and did not suggest a speed/accuracy tradeoff. They were 6.94%, 9.49%, and 5.79% for Standard, Combined, and 3-Button menus, respectively. An ANOVA with the single factor of Menu Style indicated that there were no significant differences among the means (F(2,33) = 1.575; p > .05).

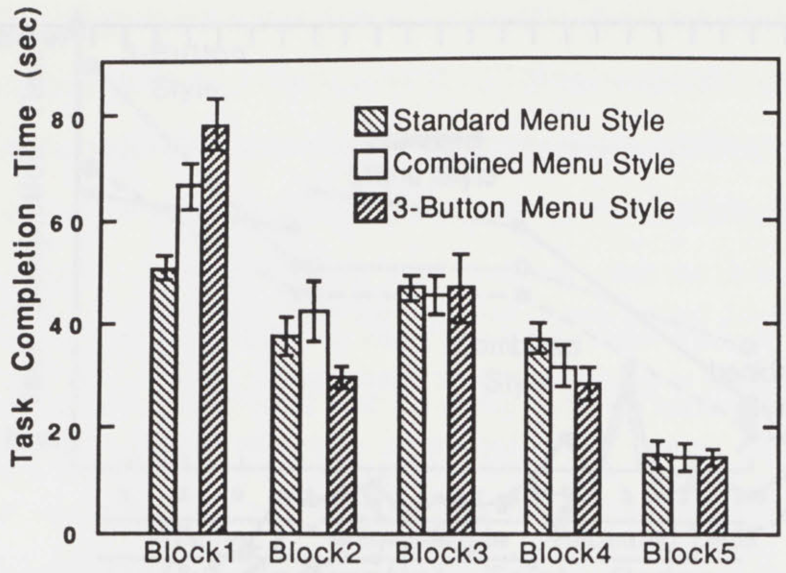


Figure 5.9 Summary of results by trial block.

2.2.4 Discussion

The results confirmed some hypotheses but contained one major surprise. The first confirmation is that standard menus fared better than 3-button before callers had learned to skip. The second confirmation is that 3-button menus fared better than standard menus when callers knew how to skip and were somewhat familiar with the menu contents.

Menu Contents Familiarity	Style Familiarity: Don't know how to skip	Style Familiarity: Know how to skip
No immediate recognition	Standard > combined > 3-button (block 1)	3-button = combined = standard (block 3)
Recognition but not recall		3-button > standard (blocks 2 and 4)
Recall		standard = combined = 3-button (block 5)

Figure 5.10 Summary of results by familiarity level for experiment two.

Combined menus fell somewhere in between but were not significantly different than standard menus. Third, skill at skipping did transfer from the weather application to the news application. Surprisingly, however, this led to performance that only equaled rather than surpassing that of standard menus in block 3. This suggests that the simplicity of positional selection confers only a small advantage, if any.

The most surprising result occurred in block 5, where all three menu styles performed about equally well. The prediction was that the positional selection of 3-button menus would be a handicap in this cell. There are several possible explanations for this result. First, perhaps some subjects had not practiced enough to know the positions of target positions, so that this block is really a hybrid of the bottom two cells in the right column. Analysis of the detailed keystroke logs, however, indicates that the trial block is much closer to the bottom cell than the middle. Across the three trials, 32 out of the 36 selections from top-level menus in the standard menu style were typed ahead before the subjects heard the associated options. Even from the bottom menus, whose contents were less familiar to subjects, 27 of the 36 selections were made without hearing the prompt for that option.

A second explanation is that it may have taken a long time to recall the appropriate menu option. Analysis of the keystroke logs indicates that type-ahead selection times from single menus in the last three trials averaged more than 2 seconds in the combined style and more than 3 seconds in the standard menu style. One would expect these numbers to decrease with additional practice. This, however, is an insufficient explanation of the result, since the typed-ahead numeric selections took only about half as long as the average selection times in the 3-button style.

The real explanation may be a combination of the two factors just discussed. Numeric selection may yield only a couple of second advantage over the 3-button style, even when callers can type-ahead, but the time penalty in standard menus for callers who cannot remember the correct number to press may be more than a couple of seconds. This analysis would suggest that even more practice would eventually yield faster selection times with the standard style than with the 3-button style. The interesting result here is that the amount of practice in this experiment (four previous selections of the same target) was not sufficient. Many practical applications are unlikely to get even that amount of concentrated practice from callers.

3 LIMITATIONS: THE EFFECTS OF PROMPT LENGTH

The experiments reported here did not vary the lengths of prompts. Pilot testing before the second experiment indicated that prompt length may affect which menu style will yield optimal performance. This makes sense because the value of skipping increases with the amount skipped. In fact, the greatest impact of skipping menu styles may be to allow longer, clearer prompts for menu options, without penalizing listeners who can quickly accept or reject the option. Any definitive guidelines on audio menu styles will have to take into account prompt length as a variable.

4 FUTURE RESEARCH: CONTENTS SELECTION STYLES

Contents selection generalizes the idea of numeric selection to that of entering a value that picks out an option from the list. The difference is that the value may describe the contents of the option, as opposed to numeric selection which merely describes the position of the option in the list. For example, the caller might enter in a date or the first few characters of a name. This selection style will have much in common with numeric selection, but callers will sometimes be able to make contents selections without hearing the options even the first time they encounter the menu, effectively increasing the number of callers in the bottom row of the hypothesis matrix. Contents selection has not been explored further in this thesis.

5 RELATED WORK

Previous research of three kinds is relevant here. First, a number of studies have refined the standard menu style. Second, a couple of studies have investigated alternative menu styles. Third, there have been a number of studies of screen-based menu styles.

5.1 STANDARD MENU STYLE REFINEMENTS

Two variables have been considered in research that attempts to refine the use of standard menus. One is whether prompts should be presented in key-action order ("Press 1 for X") or in action-key order ("For X, press 1"). Early papers argued for the first style [Gould and Boies 1983] while the more recent consensus favors the action-key order [Engelbeck and Roberts 1990, Halstead-Nussloch 1989, VMUIF 1990]. The second variable is the number of items per menu. The common wisdom is that three or four is the optimal number, and a laboratory study backs that up [Engelbeck and Roberts 1990]. Such advice is frequently ignored, however, since the categories that seem most natural often contain more than four items. I will return to this theme in the section on visual menus.

5.2 OTHER TELEPHONE MENU STYLES

Rosson and Mellen [Rosson and Mellen 1985] created a hierarchical graph in which each interior node contained a recording of a category name (e.g. entertainment, restaurants, or hotels.) Subjects were provided four buttons, two to move back and forth between categories, one to select the current category, and one to move back up the hierarchy. Effectively, this was a 3-button menu style, with selection mapped onto a spatial movement downward. Unfortunately, the mapping of information to the graph structure was not considered a variable in the study. Its novel features were not discussed, nor was it compared to the more conventional style of providing prompts for all of the categories in one node ("For entertainment, press 1; for restaurants, press 2; for hotels, press 3;...")

Roberts and Engelbeck [Roberts and Engelbeck 1989] explored a grid metaphor for making selections. Operations to configure advanced telephone functions such as call routing were laid out in the cells of a grid. The commands to navigate between nodes were spatially mapped to the telephone keypad (i.e., **2** up; **8** down; **4** left; **6** right.) They compared the grid interface to a hierarchical menu interface to the same set of options, but found no significant differences in time required to perform tasks, or in subjective preferences.

Another menuing metaphor is conversational. Callers answer a series of yes-no questions. Callers can interrupt questions by pressing the buttons for yes and no, which are consistent throughout the interface. Questions always begin with an interrogative phrase, such as "do you want..." These phrases can get repetitive, so the yes-no systems often ask a series of related questions, "Personal banking options. Do you want account balances? check update? transfers?" If the caller answers no to the question about account balances, it doesn't play the interrogative phrase again, but instead plays just the phrase, "check update?"

While the metaphor is quite different from traversing a list of options, asking a series of related questions makes this style quite similar to the 3-button style. 'Yes' is the select button and 'no' moves to the next option. The third button, to go back to a previous option, is omitted. The "Personal banking options. Do you want..." phrase serves the same role as the menu header in the 3-button interface style. It orients callers to the sequence of choices that are about to be offered. One obvious advantage of the 3-button style over yes-no menus is that the menu header is not attached to the first option. Thus, it is possible to skip the menu header without skipping the first option, which is not possible with the yes-no style. It would be interesting, however, to compare these two styles in empirical tests.

5.3 VISUAL MENUS

A number of studies have explored characteristics of visual menus [Kiger 1984, Landauer and Nachbar 1985, Laverson, et al. 1987, Lee and MacGregor 1985, MacGregor, et al. 1986, Mehlenbacher, et al. 1989, Miller 1981, Paap and Roske-Hofstrand 1986, Shneiderman 1986, Sisson, et al. 1986]. Some of these characteristics, such as visual grouping of items, and the use of pie versus linear menus [Callahan, et al. 1988] are clearly not relevant to audio menus. Other results, such as the effects of describing categories with examples [Dumais and Landauer 1983, Dumais and Landauer 1984], probably carry over without modification.

One interesting controversy began with the identification of a depth versus breadth tradeoff in hierarchical menus. That is, given a set of options, how should a hierarchical menu structure be set up? Should there be more options on each menu (breadth) or more levels of menu (depth). Several studies compared alternative menu structures for particular sets of data. Researchers developed mathematical models of selection time, some based on selection time from individual panels taking time linear in the number of options, and others assuming it was log-linear. Whichever happens to hold for visual menus, selection time is clearly linear in the number of options in an audio menu, unless callers type ahead numeric selections.

The entire depth versus breadth controversy, however, hinges on treating the menu structure as an independent variable, and assumes that it will have no effect on the ambiguity of labels for categories. Many of the studies cited above used artificial sets of targets, such as the integers from 1 to *n*, in which case the options can be divided into sets of any size without making unnatural categories. In practical applications, however, there will be only a few menu structures that yield natural category labels. The optimal menu structure, then, may be influenced far more by the natural categories for the options than by any considerations of the optimal breadth or depth of trees.

6 CONCLUSION

Audio menu styles differ in their advancement mechanisms and their selection mechanisms. The system can automatically advance between options, it can require users to skip between options, or it can provide both capabilities. Callers can make selections by pressing a number associated with the option, or using a single select button to select the option currently being played back.

Three of the most promising styles were compared in two laboratory experiments, whose results are summarized in Figure 5.11. Standard menus yielded the fastest selection times initially, before callers had learned how to skip. Once callers had learned to skip, however, 3-button menus yielded performance at least as good as standard menus, and better performance when callers were somewhat familiar with the menus but had not yet memorized their contents. Even after twenty-four trials on the same application, including four repetitions of a block of three trials, subjects were not any faster with standard

		Style Familiarity	
		Don't know how to skip	Know how to skip
Menu Contents Familiarity	No immediate recognition	Standard > combined > 3-button (2)	3-button = combined = standard (2)
	Recognition but not recall	Standard > 3-button (1)	3-button > standard (1, 2) Advantage of 3-button and combined over standard increases with target position
	Recall		standard = combined = 3-button (2)

Figure 5.11 Summary of the results from both experiments.

menus than with 3-button menus.

Once callers learned to skip, both of the Skip and Scan styles, 3-button and Combined, performed at least as well as standard menus. The experiments were inconclusive about the importance of numeric selection for callers who are very practiced in using the same set of menus.

The industry should standardize on some Skip and Scan menu style: even first-time callers to an application would then be able to transfer their skipping skills from other applications. It is not yet clear, however, which Skip and Scan style on which to standardize: 3-buttons, yes-no as described in the related research section, combined, or stepped numeric. One hopes that future research can decide this before the standard menu style becomes too entrenched.

6 CONCLUSION

A practical problem can drive applied research by providing a concrete goal against which to measure progress. The practical problem, however, is only a means to the end of accumulating knowledge that is general, communicable, and useful. The solution to the practical problem, or part of the solution, may apply to other interesting problems. Or, the process of finding a solution may guide the process of solving other interesting problems.

In this thesis project, the development of phone-based applications to support issue discussions, announcements, and question answering drove the research. Each required iterative refinement of both the functionality and the user interface, culminating in implementation and field trial of the application.

The development of particular phone-based applications led to more general results in three areas. First, the functionality required in those applications can also support other communication and coordination tasks. The field trials suggested variables that influence which tasks are good candidates. Second, the Skip and Scan interaction style and document properties generalize the user interface results from the field trials. The Skip and Scan guidelines apply beyond phone-based interfaces to all audio documents and to 'keyhole' interfaces more generally. Third, the HyperVoice application generator demonstrates techniques for generating audio interfaces from high-level abstractions that can be applied widely. Those techniques can be extended to include other high-level abstractions.

1 COMMUNICATION AND COORDINATION APPLICATIONS

Three characteristics distinguish phone-based applications:

• Recorded voice as the medium of communication;

• Remote access to information;

• Use of telephone buttons to navigate through the information.

Telephone bulletin boards are already used commercially for classified ads, mostly personal ads to match single people. The field trials described in chapter 2 demonstrated that more complex phone-based applications can also support publicity of announcements and issue discussions.

These applications are instances of a much larger class. For example, chapter 2 also presented designs for applications that support status reporting and gathering answers to commonly asked questions. More generally, most of the group communication and coordination applications that are currently being developed for networked computers, such as workflow [Carasik and Grantham 1988, Winograd 1988], sales lead tracking, and scheduling [Beard, et al. 1990, Greif and Sarin 1987], are reasonable candidates for telephone applications.

The source of power in supporting these more complex applications is the addition of structure to message objects. The structure enables the provision of fine-grain navigation commands that decrease the time necessary to find relevant information. Automatic sorting and filtering operations act on fields that contain symbolic information and pointers to other objects. Overall, the structured messages make it possible for people to browse larger collections of information than would otherwise be possible.

A number of factors will influence the success or failure of applications in particular settings, as reported in Chapter 2 and summarized below.

Green Flags
Time-critical information
Need for access from home or while traveling
Need for expressiveness of voice
Users have weak composition, keyboarding, or reading skills
Opportunity to create a 'honeymoon period'

Red Flags
Well-entrenched communication patterns
Poor distribution of costs and benefits
Need for anonymity
Naturally textual information
Need to scan large information chunks
Need to remember large information chunks

2 USER INTERFACES

Skip and Scan is a slogan that describes the ideal style of interaction callers should have with telephone bulletin boards. They should listen to short segments and then interrupt, skipping ahead to other short segments. Repeated skips should give the effect of scanning.

Two laboratory experiments confirmed the value of the Skip and Scan interaction style for audio menus. After the first few menu selections, when subjects were learning how to interrupt and skip, they made selections more quickly with Skip and Scan menu styles than with the standard menu style. If Skip and Scan menu styles were used widely, then there would no longer be an initial learning period, and Skip and Scan menus would be clearly superior.

The source of power that enables Skip and Scan interactions is making the implicit structure of recorded information explicit and available for fine-grained navigation. For example, in audio menus, Skip and Scan styles make explicit the implicit structure of audio menus as lists of prompts for different options, and give callers a button to press to skip through the prompts at their own pace. Similarly, the explicit breakup of message objects into separate fields allows for a 'smart' fast-forward button that skips ahead to the next field of an object.

To extend the Skip and Scan style to other information structures, such as grids, one need only exploit this source of power further. The document designer can find the implicit structure of the collection of information and map it to a regular set of fine-grained navigation links.

2.1 THE END OF READING?

The Skip and Scan interface style applies more generally to audio documents. The playback of the documents may be controlled by buttons on a remote device such as a telephone, a local device such as a desktop computer or a digital tape recorder, or by spoken words or phrases that a speech-to-text program recognizes. Audio documents are likely to gain in importance relative to text documents in the next decades as authoring and listening skills improve.

The printed word, from its invention, has always had more permanence than the spoken word and given readers more control over the process of absorbing information. Five centuries ago, technological innovations in printing radically improved its portability and replicability. In the last 120 years, telecommunication technologies have improved the timeliness of the written word. In the last sixty years, computing has provided searching techniques based on string matching. These technological innovations propelled the cultural ascendancy of the printed word. Countries now measure their social 'development' in part by their literacy rates.

Technological advances in other media, however, are threatening the cultural ascendancy of words, both written and spoken, as the medium of communication. Hundreds of millions, perhaps billions, of people participate in a worldwide culture through music and video images, without understanding the English words that accompany them. Neil Postman, in *Amusing Ourselves to Death*, laments this shift because sounds and images, unlike words, do not convey propositional content [Postman 1985]:

> One must begin, I think, by pointing to the obvious fact that the written word, and an oratory based upon it, *has a content*: a semantic, paraphraseable, propositional content. This may sound odd, but since I shall be arguing soon enough that much of our discourse today has only a marginal propositional content, I must stress the point here. Whenever

language is the principal medium of communication—especially language controlled by the rigors of print—an idea, a fact, a claim is the inevitable result. The idea may be banal, the fact irrelevant, the claim false, but there is no escape from meaning when language is the instrument guiding one's thought. Though one may accomplish it from time to time, it is very hard to say nothing when employing a written English sentence. (p. 49-50)

Even for communication with words, technological changes are increasing the value of speech relative to writing. In the electronic age, the spoken word can now be stored, transported, and duplicated. Postman will likely decry this shift as well if it occurs, for the written word will always be more abstract than the spoken word. If all other things were equal, however, speech would be preferable to text precisely because of its concreteness, its expressiveness. In fact, some researchers are even trying to add expressiveness to machine-generated speech [Cahn 1990].

There are differences between comprehending information visually and aurally that have to do with biological rather than cultural traits. But in predicting the limits of audio documents, it is virtually impossible for an observer within our culture to separate the biological from the cultural traits. Skip and Scan is a first attempt to develop authoring and listening styles that take advantage of random access audio. Other researchers are exploring how sound localization and fast forward and reverse playback can aid listeners [Arons 1991a, Arons 1992]. Right now it appears that reading is far superior to listening, but it is not clear whether this superiority would remain after decades of improvements to audio document styles and after a generation of people grew up practicing their hyperdocument listening skills every day.

3 SOFTWARE TOOLS

The HyperVoice software tools make it easy to implement telephone bulletin boards that are customized to particular communication applications. Development takes less time and the resulting programs are less likely to contain bugs. The language is simple enough that non-programmers can participate in a rapid prototyping process. Moreover, HyperVoice automatically generates user interfaces that conform to the Skip and Scan guidelines.

The source of power in HyperVoice is that it narrows the abstraction gaps between design, programming, and use. Lists, menus, forms, and login procedures are helpful abstractions for all three activities. Chapter 2 uses these abstractions to describe a number of applications and non-programmers used them to discuss alternative designs. Lists, menus, forms, and login procedures are also the basic building blocks of the HyperVoice application layer language. Finally, they are the metaphors which help callers to understand the operation of an application.

The development of the HyperVoice application generator is a case study in how to develop application and interface generators for specialized domains. The key is to find a higher-level set of abstractions that capture the commonalities of programs in the domain and parameterize the differences. The search begins with the construction of designs and possibly tests of a few applications in the domain, using lower-level abstractions. Three techniques may then help to identify a higher-level set of abstractions:

- Notice the abstractions that are useful for describing the sample application to designers or users. Those abstractions may also be useful as programming constructs.

- Identify the pieces of low-level code that were copied and edited in building the sample applications. The sections that were copied may perform easily described functions that can be named as higher-level abstractions. Within each such section of code, the parts that were edited are candidate parameters of the new abstraction.

- Separate one sample program into conceptually distinct pieces. Find abstractions that specify one conceptual piece and try to automatically generate the other part. In the case of HyperVoice, the application layer abstractions came from separating application functionality from the user interface and trying to automatically generate the latter.

4 ANSWERS AND QUESTIONS

Can phone-based applications support complex group communication and coordination tasks? And if so, how? The answers in this thesis are: Yes, use HyperVoice to generate Skip and Scan interfaces.

Perhaps more important than questions answered are questions raised. If the previous conceptions of the limits of telephone-based applications were too narrow, what are the true limits? If the conception of audio as a serial medium is misleading in the age of random access, then could speaking and listening someday supplant writing and reading? What are the techniques for developing application generators for new domains? This thesis begins the process of answering these larger questions.

A THE HYPERVOICE PROGRAMMING LANGUAGE

Chapter 4 described the novel and important features of the HyperVoice language. This appendix presents the details of the language more systematically. It serves as a programmer's manual for the HyperVoice language. It presents the language syntax and an informal description of its semantics. Then, it presents macros for two common programming clichés. Throughout, examples from Chapter 2 ground the discussion of the language semantics and justify the need for some of the programming constructs. Appendix B presents the pre-processor and interpreter, which give a more detailed semantics for the language.

1 THE LANGUAGE

The HyperVoice programming language is unusual in two ways. First, programs are networks of data objects rather than linear texts. For example, the presentation format for a list of objects contains a pointer to an Item Format object that specifies how each item should be presented and another pointer to a List Action object that specifies how new items will be added. These linked specifications may be reused in specifying the presentation of other lists.

Second, HyperVoice programs are declarative rather than procedural. They specify how data are to be organized for presentation to callers, but they do not directly specify the flow of control in executing programs. The language interpreter automatically determines the flow of control and other user interface details.

1.1 THE OBJECT SYSTEM

The messages on a HyperVoice bulletin board are stored in an object-oriented database. Object classes determine the fields of which messages are composed. For example, the Event Announcement object class has eight fields: headline, date, time, location, contact number, details, date added, and category.

The database handles five data types: recording, text, date, integer, and pointer to another object. The fields, however, are not typed. Any field can contain zero or more values of any type, although fields may be more or less meaningful depending on whether they contain the right types of data. A programmer can create new object classes that reflect the natural structure of the messages to be posted in a particular application.

Object instances (messages) are grouped into ordered sequences, called *lists*. An object may appear in more than one list. For example, in an events calendar application, there is one list for each event category, plus an overall list of all announcements. Each announcement appears twice, once in the overall list and once in a category-specific list.

It might at first appear that lists introduce an unnecessary indirection, since the fields of objects can contain multiple values. Anywhere that a field contains a pointer to a list of objects, one might instead put pointers to all of the objects in the list. The indirection is helpful, however, when the contents of the list change. With lists as objects in their own right, several objects can contain pointers to a list. Then, when the contents of the list change, all of the objects will automatically have access to the updated list.

Lists are implemented as a class in the object system, as illustrated in the table below. A word about notation is in order here, since tables of this sort pepper the remainder of the appendix. The top line in the table indicates that a class is defined and gives its name: List. The leftmost column gives the names of the fields of the class. The middle column indicates the types of values that are expected in the field, in order for the class to be meaningfully interpreted. The rightmost column gives a short English description of the expected contents of the field or what the contents will be used for. In the case of the class List, there is a single field, called 'Contents'. It expects zero or more values that are pointers to other objects. The star (*) notation indicates that zero or more values of the indicated type are expected, and the square brackets indicate a pointer to another object.

Class: *List*		
Field Name	Alternative Values	Description
Contents	*[]	Zero or more pointers to objects, of any class

All of the language primitives are implemented as objects in this object system. The OVAL system [Lai, et al. 1988] provides a screen-based interface for entry of objects. Chapter 4 drew analogies between presentation formats in OVAL and HyperVoice. Thus,

Primitive	Description
Login Process	Specifies the login procedure necessary to access an application.
User	Has fields for a recorded name, an id number, a password, and the 'privileges' that should be accorded to a particular user.
List Jump	Specifies special digit sequence a caller with appropriate privileges can press to jump to a particular list.
List Presentation	Pairs a List with a List Format to specify the presentation of one list of objects.
List	A list of objects.
List Format	Specifies how to play back the contents of a list, including how to filter and sort it, and how new objects can be added.
Filter	Selects a subset of a list to be presented.
Validity Check	A predicate on the contents of a field. Used in Filters and Field Edit Formats.
Sort Ordering	Specifies the order in which to present objects from a List.
Item Format	Specifies which fields of one object to play and in what order.
Field Format	Specifies how to play back one field.
Select Action	Used as part of an Item Format to specify what action to take when an object is selected from a list.
List Action	Specifies how new items can be added.
Extension Format	Specifies what kind of object will be added and where it will be added.
Edit Format	As part of an Extension Format, specifies a telephone form for editing a new object.
Field Edit Format	Specifies how to present one field in a telephone form.

Table A.1 The object classes that the application interpreter recognizes.

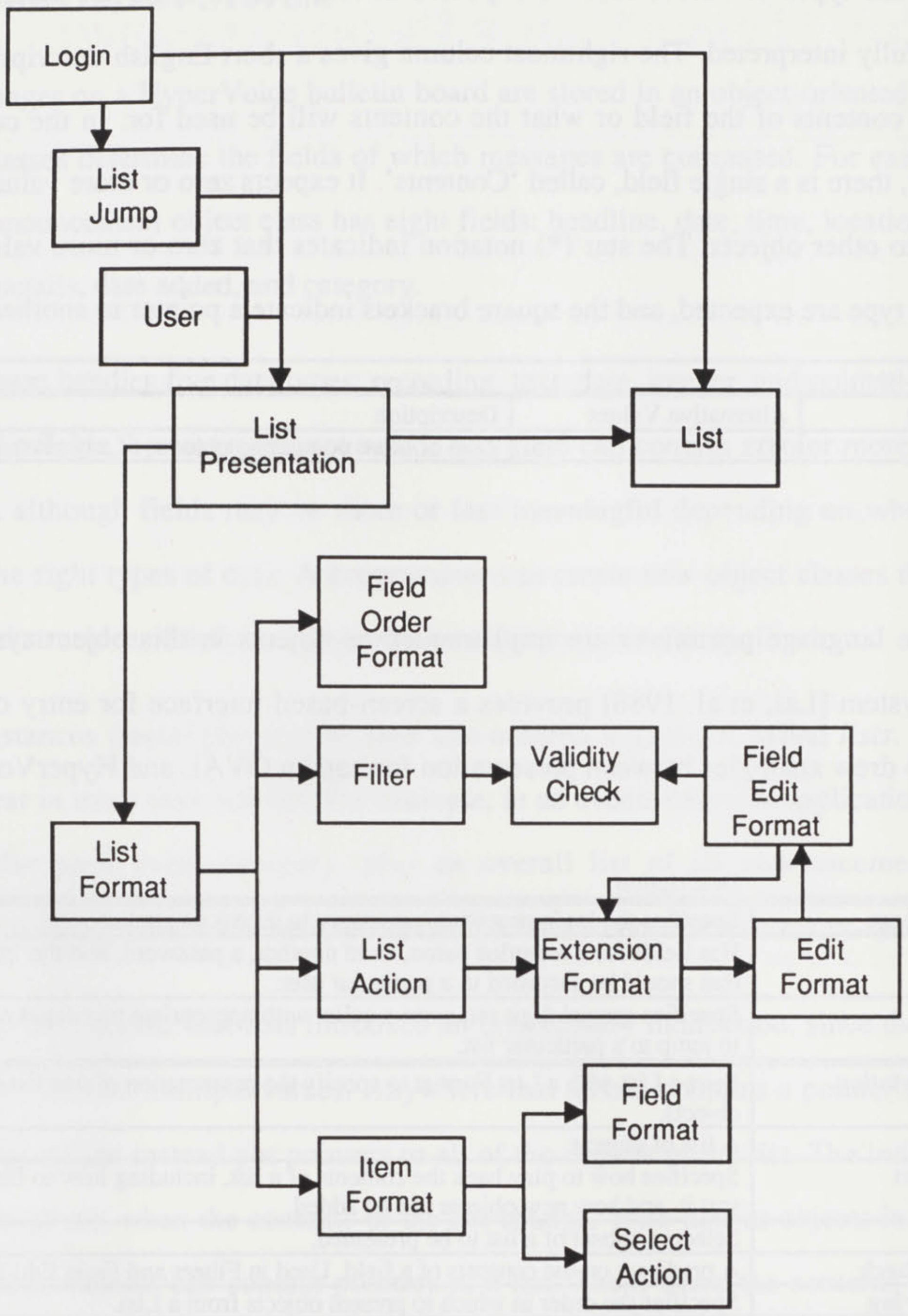


Figure A.2 The primitives for specifying presentation and editing formats. An arrow indicates that an instance of one primitive may contain a link to an instance of another.

OVAL was a source of ideas for HyperVoice. Here, OVAL is also used as a software tool, a front end for writing HyperVoice programs. It provides screen-based forms for editing objects and an easy way to create pointers between objects. While a production version of HyperVoice would require a front end that does more to guide programmers through the process of specifying an application, the current OVAL front end has made it possible to specify new applications in a matter of hours.

Table A.1 and Figure A.2 provide overviews of the primitives that are used to specify HyperVoice programs. They are repeated from chapter 4.

1.2 LOGIN PROCEDURES

Before any lists or menus are presented, a phone call can begins with a welcome message and sometimes a registration procedure in which the caller enters an id number and a password. The login procedure performs several important tasks. First, it can restrict access to only users who know an id number (and password, if desired.) Second, it sets the privileges for the rest of the call. Callers without certain privileges may not be allowed to add new objects to certain lists. Third, it sets the current_user variable to be an object that represents the user. When the caller adds a new message the system can automatically set a field of the message to be a pointer to that object. Finally, the application can be customized to each user's needs. In particular, the login process can determine for each user a different root list of objects that will be presented.

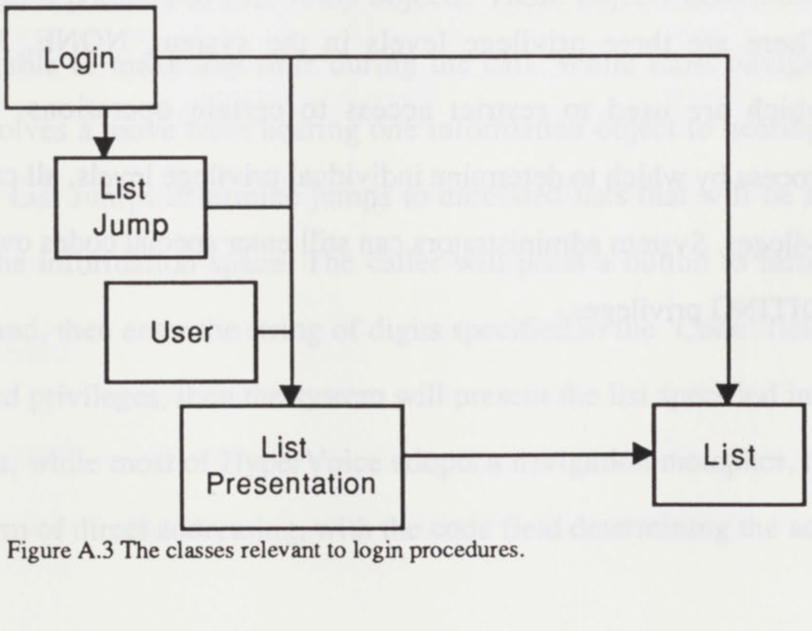


Figure A.3 The classes relevant to login procedures.

Class: *Login*		
Field Name	Alternative Values	Description
Greeting	Prompt	The first thing a caller hears
Register?	TRUE, FALSE	If FALSE, there is no login process at all. Callers go straight to the initial List Presentation.
User Id Length	Integer	When Register? is TRUE, this determines how many digits are in a user id.
Password?	TRUE, FALSE	If FALSE, users do not need to enter a password.
Password Length	Integer	If Password? is TRUE, this determines how many digits are in the password.
Users List	[List]	A list of objects of type User, which determines the eligible users.
Start List	[List Presentation]	What to present after the login process is complete.
Special List Jumps	[List Jump]*	A list of objects of type List Jump. These determine codes that can be entered throughout the application to jump to special lists.

The first thing a caller will hear is a recording of the 'Greeting' prompt. The prompt data type indicates that the programmer types in the text of a prompt that will be recorded later, during a prompt recording session. In a pre-processing step, the system will gather into a script all of the prompts that need to be recorded, both those that the programmer typed in explicitly and those that were generated by the system.

If 'Register?' is TRUE, the caller will be prompted to enter a user id, a string of digits whose length is determined by the 'User Id Length' field. If 'Password?' is also TRUE, the caller will be prompted to enter another string of digits as a password. If 'Register?' is FALSE, the system goes directly to presenting the List Presentation object in the 'Start List' field. There are three privilege levels in the system, NONE, ADDING, and EDITING, which are used to restrict access to certain operations. If there is no registration process by which to determine individual privilege levels, all callers are given ADDING privileges. System administrators can still enter special codes over the phone to upgrade to EDITING privileges.-

Class: *User*		
Field Name	Alternative Values	Description
Name	Recorded voice	
User Id	Digit string	
Password	Digit string	
Privileges	NONE, ADDING, EDITING	Privilege levels determine what operations will be available during the phone call
Start List	[List Presentation]	The starting point in the application, if different from the default specified in the Login Process

When 'Register?' is TRUE and a caller successfully enters an id and a password, that picks out an instance of class User from the list in the 'Users List' field. The User object determines the initial privilege level a caller will have. The User object also determines the root List Presentation to present as the starting point for the caller's navigation through the information. In the extreme case, different users might access completely different applications that happen to share the same telephone number.

Class: *List Jump*		
Field Name	Alternative Values	Description
Code	Digit String	The special code to initiate this List Jump
To Jump To	[List Presentation]	Which List to present
Privileges Required	NONE, ADDING, EDITING	The privilege level the caller needs to have in order to initiate this List Jump

With or without a registration procedure, the 'Special List Jumps' field of the Login object can contain pointers to List Jump objects. These objects determine jumps that a caller will be able to make any time during the call. While most navigation during a phone call involves a move from hearing one information object to hearing a related list of objects, the List Jumps determine jumps to unrelated lists that will be available from anywhere in the information space. The caller will press a button to initiate entry of a special command, then enter the string of digits specified in the 'Code' field. If the caller has the required privileges, then the system will present the list specified in the 'To Jump To' field. Thus, while most of HyperVoice adopts a navigation metaphor, the List Jumps introduce a form of direct addressing, with the code field determining the address.

1.3 PRESENTATION FORMATS

Presentation Formats determine how information will be selected and arranged for presentation and how new information objects can be added. Selection and arrangement operations can be applied both to lists of objects and to the fields of individual objects. Subsets of the objects in a list can be selected by setting a filter on the contents of one or more fields of the objects. The objects in a list can also be sorted by specifying a comparator on the contents of a field. A subset of the fields of each object can also be selected, and those fields can be presented in any order.

Presentation formats also contain links to editing formats, which determine how new objects will be added. Editing formats include specifications of the locations in the information space where actions will be available to add new objects, what privileges are required to execute those actions, and what lists the new objects will be added to. Editing formats also specify the contents of a telephone form for adding the new object, including initial values and validity checks for the contents of some fields.

Class: *List Presentation*		
Field Name	Alternative Values	Description
List	[List]	Which list of items will be presented
Format	[List Format]	Specifies how each item will be presented, in what order, and how new items can be added.
Menu Prompt	Prompt	Played when this object is an option in a menu.
Title	Prompt	Played in the list header, and when returning to the list after visiting somewhere else.
Description	Prompt	Played in the list header.

The root object for specifying the presentation of a list of objects is an instance of class List Presentation, which contains a pointer to a List and a pointer to a List Format. When a menu contains the option of going to this List Presentation, the system plays the 'Menu Prompt' recording to tell the caller about the option. For example, if the List Presentation is the news reports in an audiotex system, the 'Menu Prompt' might be, "For news..." The prompt text does not include, "Press 2" because the interpreter will automatically

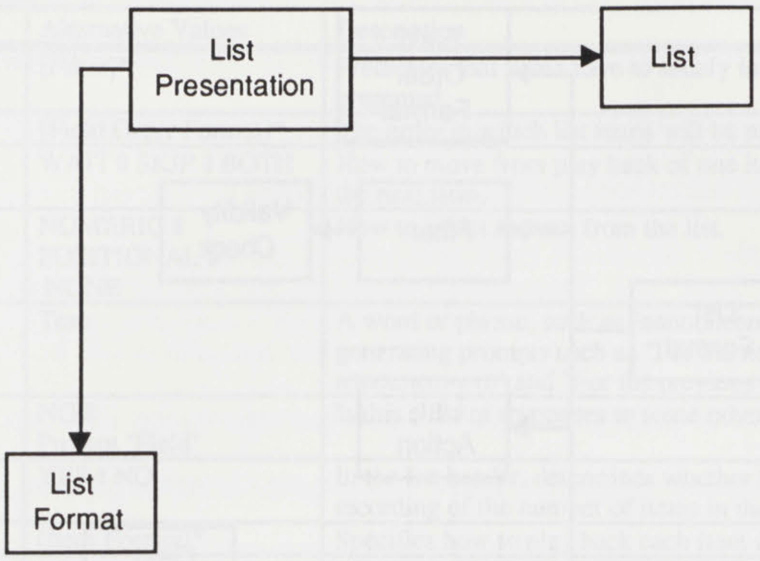


Figure A.4 A List Presentation object contains pointers to a List and a List Format.

determine the number associated with the menu option and will concatenate the appropriate prompts.

When a list is presented, items are laid out sequentially and the caller will have actions available to move forward and back in the list. The caller always begins on a list header, which precedes the first item in the list. The contents of the list header begin with the 'Title' prompt and include the 'Description' prompt . The 'Title' prompt is also used separately. When a caller returns to a list after visiting another one, the system tells her the current location (e.g., "item 3 in") and then plays the 'Title' prompt. To continue the example, the 'Title' might be, "The news desk." The 'Description' might be, "Headlines and story summaries from the Wall Street Journal, the New York Times, and the National Enquirer."

1.3.1 List Formats

List Formats specify all the details about how to arrange the information in a list for presentation, and how to prompt callers to navigate through the list. In addition, List

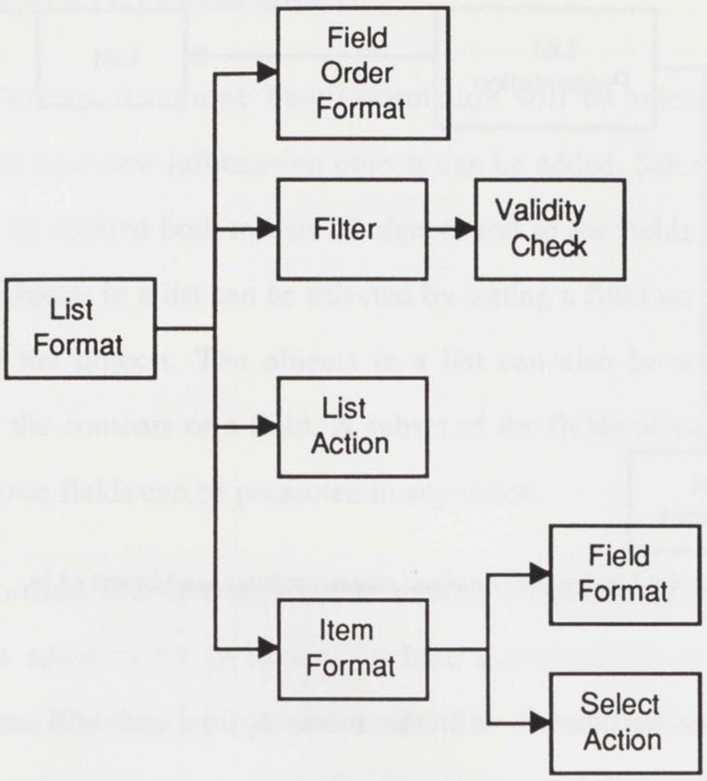


Figure A.5 The classes used to specify the presentation format for a list of objects.

Formats contain pointers to List Action objects that determine how new objects can be created, edited, and then added to lists.

Not every item in a list need be presented: the 'Filter' field determines which ones will. Each Filter object determines a predicate on items in the list, and there may be more than one Filter. An item is selected if all the predicates are true of that item, so the Filter objects are implicitly ANDed together.

If the 'Sort Order' field is empty, the selected items are presented in the order in which they are stored in the list. If included, Field Order Format objects determine the order in which to present the items. Each Field Order Format determines a comparison operator on pairs of objects, which is all that is needed to run reasonably efficient algorithms such as quicksort [Cormen, et al. 1990] to determine the total order. If there is more than one Field Order Format, the second breaks ties between pairs of objects that the first

Class: *List Format*		
Field Name	Alternative Values	Description
Filter	[Filter]*	Predicates that items have to satisfy in order to be presented
Sort Order	[Field Order Format]*	The order in which list items will be presented.
Advancement Mechanism	WAIT ‖ SKIP ‖ BOTH	How to move from play back of one item to playback of the next item.
Selection Mechanism	NUMERIC ‖ POSITIONAL ‖ NONE	How to select an item from the list.
Name For Objects	Text	A word or phrase, such as "announcement." Used in generating prompts such as "For the next announcement" and "For the previous announcement."
Response List?	NO ‖ Prompt 'Field'	Is this a list of responses to some other object?
Say Item Count?	YES ‖ NO	In the list header, determines whether to play a recording of the number of items in the list.
Item Formats	[Item Format]*	Specifies how to play back each item in the list.
List Actions	[List Action]*	Actions that can be taken to add new information objects.

comparison determines are equal, the third breaks ties when both of the first two determine that a pair of objects is equal, and so on.

The Advancement Mechanism determines how a caller can advance to the next item: either by waiting until the current one is completed (WAIT) or by pressing a button (SKIP), or a combination of the two (BOTH). The Selection Mechanism lets any list of items be treated as a menu. With NUMERIC selection, the caller presses **1** to select the first object, **2** to select the second, and so on. With POSITIONAL selection, a single select button selects the object currently being played back. If the 'Selection Mechanism' field contains the value NONE, then selection is not permitted at all. Recall that these two parameters, advancement mechanism and selection mechanism, define the space of menu styles that was explored in Chapter 4 and 5.

The Name for Objects field is used in generating the text of prompts that will tell the caller how to navigate through the list. For example, if the word "question" is filled in for that parameter, a number of prompts will be generated, such as, "For the next question, press 9," and, "For the previous question, press 7." The programmer need not write out all of these prompts, nor even be aware of all the prompts that use the word "question." As long as the programmer is happy with how the word sounds substituted into the phrase, "For the next <Name for Objects>, press 9," the programmer can be confident that the other prompts will also sound natural.

Along with the 'Title' and 'Description' fields of the List Presentation, the 'Response List?' parameter of the List Format determines what recordings will be played back in the header of the list. If the field has the value NO, then just the 'Title' and 'Description' fields are played back. Often, however, lists are automatically created to accommodate responses to some other message. When the lists are created automatically, no 'Title' and 'Description' recordings are made. This 'Response List?' field specifies how a meaningful header prompt can be generated automatically. The first value in the 'Response List?' field is a prompt fragment that is shared by all lists that use this List Format. The second value is a field name, which selects a field of the object to which this a list of responses. If the field selected contains a short recording, then the header prompt for the response list can be quite useful. For example, in the teachers' curriculum line application, the original object is a lesson plan, one of whose fields is 'Lesson Number.' Each time a new lesson plan object is added, a response list is created to accommodate questions about that lesson plan. The 'Response List?' field for the List Format is, "Questions about lesson" 'Lesson Number'. The list header for the questions about lesson number 35, then, states, "Questions about lesson 35."

The 'Say Item Count?' field also helps determine what will be played in the header for the list. After the title recording, and after any recordings generated from the 'Response List?' field, the system may inform the caller of how many items are in the list. If the 'Say Item Count?' field has the value YES, then the system will play something of the form, "Five questions are in the list." Here, five is determined by the number of items that passed the Filter, and the word question is taken from the 'Name for Objects' field.

The Item Formats determine which fields of the objects will be played back, in what order, and whether the contents of the fields will be preceded by recordings of the field names. If selection is allowed from this list, then the Item Format also determines what action will be taken if the item is selected. One Item Format is included for each of the object types that can appear in the list. If there is no Item Format for a class of object that appears in the list, the system uses a default Item Format for the class.

One feature of lists that is not parameterized, since it is available with every list, is maintenance operations. Anyone with editing privileges has access to a cut operation that removes the object being listened to from the current list and sets a clipboard variable to point to it. Subsequently, a paste operation inserts the object into a list, just before the object that is currently being played.

1.3.2 Filtering

Class: *Filter*		
Field Name	Alternative Values	Description
Or Filters	[Filter]*	Filter is satisfied if any of the specified filters is satisfied
And Filters	[Filter]*	Filter is satisfied if all of the specified filters is satisfied
Not?	TRUE ‖ FALSE	If TRUE, accepts only those objects that do not pass the filter.
Class Required	[Class]*	Object must be of one of the specified class
Field Filters	([Field] [Validity Check])*	Specified fields must pass the respective validity checks

A Filter object determines a predicate on other objects. The predicate is either a boolean combination of other predicates, or specifies the allowable classes and the allowable values of particular fields of the objects. If several Classes are linked into the 'Class Required' field, an object passes the predicate if it is a member of one of the classes, and it passes the predicate defined by the 'Field Filters.' The 'Field Filters' field specifies pairs consisting of a field name and a validity check. The field name determines the field of the object from which to select a value and the validity check determines whether the value is acceptable. If the field contains several values, at least one of the values has to pass the validity check. If more than one pair is specified, an object satisfies the predicate if it satisfies all of the validity checks on the values in its fields. The validity checks are instances of one of the four classes below. In other words, a validity check is applied to each of the values in a specified field, and the results are ORed together. This is done for each field and validity check specified, and the results are ANDed together. Finally, this result is ANDed with the result of checking that the object is an instance of one of the acceptable classes. If the final result is positive, then the object satisfies the predicate.

Class: *Date Validity Check*		
Field Name	Alternative Values	Description
Min Value Relative to Today	An integer.	The earliest acceptable date, computed from the current date when the validity check is run.
Min Value Absolute	A date.	The earliest acceptable date.
Max Value Relative to Today	An integer.	The last acceptable date, computed from the current date when the validity check is run.
Max Value Absolute	A date.	The last acceptable date.

In order to satisfy a Date Validity Check, a value first of all has to be of type date. If 'Min Value Relative to Today' is specified, it is added to the date on which the validity check is evaluated: the value must be the same as or later than that. If an absolute minimum date is specified, then the value has to be the same as or later than that. Similarly with the maxima, the value has to be the same as or smaller than the specified values. If more than one of the fields is filled in, a value has to satisfy the constraints determined by all of the field in order to pass the validity check.

Class: *Object Validity Check*		
Field Name	Alternative Values	Description
Equals	[Object]	Value in field must be a link to the specified object.
Member of	[Object]*	Value in field must be a link to one of the specified objects.
Filter	[Filter]	Value in field must pass the specified Filter.

In order to satisfy an Object Validity Check, a value has to be a pointer to an object. The first field specifies a particular object as the destination of the pointer. The second provides more flexibility, accepting pointers to any of the specified objects. The last field allows specification of a recursive filter. A pointer to any object passes this validity check, so long as the linked object satisfies the predicate specified by the object the 'Filter' field points to.

Class: *String Validity Check*		
Field Name	Alternative Values	Description
Min Value	A string	Smallest acceptable value, with string< as the comparator that defines relative sizes of strings.
Max Value	A string	Largest acceptable value, with string< as the comparator that defines relative sizes of strings.
Min Length	An integer	An integer: smallest acceptable string length.
Max Length	An integer	An integer: largest acceptable string length.

A String Validity Check accepts only values that are digit strings. The first two fields can specify minimum and maximum values for the string. In comparing two strings, string comparison is used rather than numeric, so "20" is less than "3". The last two fields specify minimum and maximum acceptable lengths. Again, the value has to satisfy the constraints specified by all of the fields that are filled in.

Class: *Number Validity Check*		
Field Name	Alternative Values	Description
Min Value	A number	Smallest acceptable value.
Max Value	A number	Largest acceptable value.

Number Validity Checks requires values that are numbers. The two fields can reject values that are too low or too high. Here, numeric comparison is used, so that 3 is less than 20. Again, the value has to satisfy the constraints of both fields if values are filled in for both.

1.3.3 Sorting

Class: *Field Order Format*		
Field Name	Alternative Values	Description
Field	[Field]	Which field to use for sorting
Direction	ASCENDING ‖ DESCENDING	For ascending order, smallest values go first.

Field Order Formats are used to sort objects. They specify fields from which to extract values for comparison and a direction for sorting. Only the first value in the field is used for sorting. For date, string, or number values of the same type, the comparison is made using the comparison operator for those values. If a pair of text values, prompts, or a pair of pointer values are compared, they are declared equal. Finally, if values of different types are compared, the result is determined by an ordering of the types, regardless of the values. The type ordering places dates first, then numbers, digit strings, text, prompts, and finally links. These details assure that the sort ordering is well defined, but are rarely relevant. In all of the application prototypes presented in this thesis, the values in the specified fields are always of the same type, either dates or numbers.

1.3.4 Item Formats

Class: *Item Format*		
Field Name	Alternative Values	Description
Object Class	[Class]	The class of object for which this specification is intended.
Field Formats	[Field Format]*	Which fields to play in which order
Select Action	[Select Action]*	What to do if the item is selected

Item Formats determine how the information from individual objects will be arranged for playback, and what action to take if the object is selected as a menu option. Each Field Format picks out one field of the object that should be played back. By ordering the Field Formats, the programmer can specify different ways to play back the information contained in objects. By leaving some fields out, the programmer can omit information that might be redundant or inappropriate to hear in a particular context.

Class: *Field Format*		
Field Name	Alternative Values	Description
Field	[Field]	The field from which to extract values for playback
Field Name Spoken?	YES ‖ NO	Whether to recite the name of the field before playing its contents
Linked Item Format	[Item Format]	If the contents of the field is a link to another object, an item format that specifies how to play back that item.

The 'Field Name Spoken?' parameter determines whether or not a recording of the field name will be played before the contents of the field. In some situations, playing the field name will orient callers to the kind of information that is about to be played. In other situations, the caller is already oriented and the field name would just get in the way. The programmer can specify independently for each field whether or not to play the field name. The 'Linked Item Format' is used when a field contains a pointer to another object. The Item Format determines which fields of that object to play back.

Class: *Select Action*		
Field Name	Alternative Values	Description
Action Type	GOTO OBJECT ‖ GOTO FIELD 'Field' ‖ RETURN ME ‖	If this object is a List Presentation, then GOTO OBJECT will cause it to be presented. GOTO FIELD will cause the List Presentation in the specified field to be presented. RETURN ME will cause the selected object to be returned as a value.
Action After Returning	RESET ‖ STAY AT CURRENT ITEM	When you return from the excursion to that List Presentation, should you reset to the beginning of the current list, or stay on the current item? Used only in conjunction with GOTO OBJECT and GOTO FIELD action types.
Relationship	text	Used to generate the prompt for selecting the item.

A Select Action object is also linked into the Item Format. Its 'Action Type' field determines what will happen if the item is selected from the list. The GOTO OBJECT value is used only when the Item Format is for a List Presentation object. Then, if the object is selected, the system will go to presenting that object. That is how navigation menus are usually implemented. For other types of objects, the GOTO FIELD 'Field' picks out a field of the object that should contain a link to a List Presentation object. For example, a lesson plan object has a field, 'Questions', that contains a pointer to a List

Presentation object for questions about the lesson plan. In that case, the first field of the Select Action would have the value GOTO FIELD 'Questions'. Finally, if the value is RETURN ME, then the current object is returned as a value. This is useful in picklists for selecting an object to link into a field of a telephone form, as will be discussed under Field Edit Formats below.

The 'Action After Returning' field is only relevant when the first field has the value GOTO OBJECT or GOTO FIELD. In those cases, the caller will go to another list and be able to return to the current list after finishing with the new one. This field determines where in the current list the caller will return to. If the value is RESET, the caller will return to the beginning, the list header for the current list. With a value of STAY AT CURRENT ITEM, the caller will return to the item that was selected.

When the 'Action type' is GOTO FIELD, the 'Relationship' parameters provides text to generate the prompt for select objects of this type. It indicates the relationship between the current object and the linked List Presentation. For example, in the teachers' curriculum line, the phrase, "Questions about" for this parameter generated the prompt, "For questions about this lesson plan, press..."

Note that it is possible for an Item Format to contain more than one Select Action. This makes it possible to branch to more than one other List Presentation from each item in a list. All of the prototype applications presented in this thesis, however, use only a single Select Action for each Item Format.

1.3.5 Adding Objects

A List Action specifies how a caller can add new objects. Note that a List Format may include more than one List Action, so that callers have options about the kinds of objects to add or where the objects will be added. Of the prototype applications presented in Chapter 2, only the Question and Answer program utilizes this feature. In that case, it

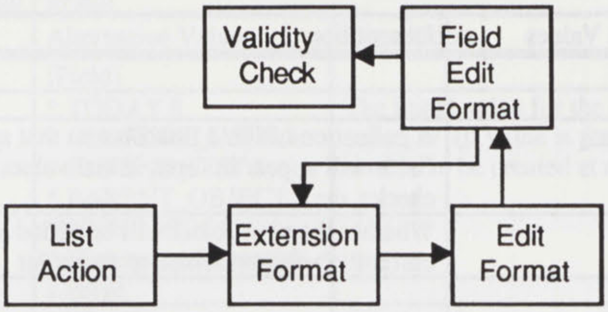


Figure A.6 The classes that specify the addition of new objects

allows system administrators to create an information tree over the phone, by adding either interior nodes or leaf nodes anywhere in an existing tree.

Class: *List Action*		
Field Name	Alternative Values	Description
Privileges Required	NONE, ADDING, EDITING	Determines what privileges are required in order to execute this action
Button Location	HEADER, FOOTER, ITEMS, ALL	From where in the list can this action be initiated
Extension Format	[Extension Format]	What kind of object will be added and how it will be edited.

The first parameter of a List Action object determines who will be able to add new objects. As described above under Login procedures, each caller is assigned a privilege level. If the caller's privilege level is lower than that required, then the caller will not be able to execute this List Action. Typically, a caller can initiate entry of a new object from anywhere in the list, but the Button Location parameter can be used to restrict that action to just the header of the list, just the footer, or just the list items.

The List Action specification also includes an Extension Format specification, which determines what kind of object will be created, how it will be edited, and where it will be added. The current list being presented is an implicit variable that the Extension Format makes use of. When the caller finishes editing the new object, if the 'Add to Current List?' field is set to YES, then the object will be added to whatever list is currently being presented. There may be reason to add the new object to other lists in addition to or

Class: *Extension Format*		
Field Name	Alternative Values	Description
Class of Object to Add	[Class]	
Edit Format	[Edit Format]	A collection of Field Edit Format that specify which fields will appear in form, initial values, validity checks, etc.
Add to Current List?	YES, NO	Whether the new object will be added to the list currently being presented to the caller
Other Lists to Add To	[List]*	
Location to Add	END, BEGINNING	Add the new object at the end or the beginning of the list(s) to which it will be added.

instead of the current one, hence the 'Other Lists to Add to' field. The 'Location to Add' field determines whether the new object will be added at the ends or at the beginnings of the lists to which it is added.

Class: *Edit Format*		
Field Name	Alternative Values	Description
Field Formats	[Field Edit Format]*	Linked objects that determine the entry blanks of the telephone form.
'Header Prompt'	Prompt	Used in the header that describes the telephone form to the caller.

The Edit Format contains pointers to Field Edit Formats, one for each field of the object, which determine initial values for the fields and various other details of how the field will appear in the telephone form for editing the object. The form will present a sequence of entry blanks, one for each field that can be edited. Callers will navigate between the entry blanks and enter or change values in the fields. Callers will begin on a header node that explains that this is a telephone form for editing a particular kind of object. As part of the form header , the 'Header prompt' is also played.

The 'Initial Value' can either be a value, or a specification of how to generate a value at run time. * TODAY fills in the current date. * CURRENT USER fills in a link to the User object that was selected during the login process. * PARENT LIST fills in a link to the List Presentation object that is currently being presented. * PARENT OBJECT fills in a link to the object that was selected in order to make the current list be presented, if there

Class: *Field Edit Format*		
Field Name	Alternative Values	Description
Field	[Field]	
Initial Value	* TODAY ll * CURRENT_USER ll * PARENT_LIST ll * PARENT_OBJECT ll * NEW LIST ll * [Extension Format] ll a value	The initial value for the contents of the field. Either a value, or (if value is preceded by *) the specification of a value to be created at runtime.
Include in Form?	YES, NO	Whether the contents of the field are editable by the user.
Data Type	VOICE ll LINK ll DIGIT STRING ll DATE ll NUMBER	The type of data that a caller can add to this field
Accept Multiple Values?	YES ll NO	Accept entry of more than one value, or just one?
Picklist	[List Presentation]	For POINTER data type, a list of alternatives from which to make a selection.
Max Length	An integer	Maximum number of seconds for a recording, if VOICE is the data type, or maximum number of digits to collect, if DIGIT STRING is the data type.
Validity Check	[Validity Check]	Predicate that checks whether a value that the caller inputs is valid.
Field Description	Text	In the entry blank for the field, this prompt will be played if no value has been filled in yet.

was one; typically, this is an object to which the current list contains responses. * followed by a pointer to an Extension Format object causes sets the field to be pointer to a new object that is created and filled in according to the Extension Format. In this way, embedded data structures can be generated automatically.

The 'Include in Form?' field determines whether or not an entry blank will appear in the telephone form for editing the current field. If a field has an initial value and does not appear in the form, that becomes the final value. If the field appears in the form, then the caller will have a chance to edit it. The rest of the fields are relevant only if this one gets the value YES.

The 'Data Type' determines what kind of values will be entered into the field. If voice data is accepted, the caller will record. Digit strings, dates and numbers will be typed in using the telephone keypad. Pointers are selected from a list of objects specified in the 'Picklist' field.

The 'Picklist' field points to a List Presentation object, which should be a list that allows selection, and the Item Format of the selected item should have the 'Select Action' field value RETURN ME. In short, the caller can select an object from the list specified in the List Presentation object. A pointer to that object will be returned and added as a value to the field of the object being edited.

The 'Max Length' field determines the maximum length of typed-in input. It is relevant only for numbers and digit strings.

The 'Validity Check' field restricts the values that will be accepted as inputs into the field. For example, dates might be restricted to be no more than one year from the date of entry. After a value is entered, it is compared against the linked Validity Check object. If it does not pass the check, the system notifies the caller that the value has been discarded and allows him or her to try again.

Finally, the 'Field Description' parameter contains a prompt that is played as part of the entry blank, if the field has no values. It tells the caller what kind of value is expected in the entry blank, and, in some cases, what the value will be used for. If the field already has one or more values filled in, the field description is omitted and the system instead plays the contents of the field.

1.4 SUMMARY

The List Presentation is the fundamental unit in a HyperVoice program. It specifies the presentation of some or all of the object in a list. Callers can navigate forward and back through the objects in a list and can follow links to other List Presentations. They can also create new objects and add them, either to the list currently being presented, or to other lists.

This concludes the presentation of the HyperVoice language primitives. These primitives can be roughly divided into three categories. First, the Login, User and List Jump classes determine what, if any, registration procedures will be used at the start of each call, what privileges different users will have, and any special jumps that will be available throughout the application. Second, list presentation formats for lists determine the selection and ordering of information to be presented. Third, editing formats, including the List Action, Extension Format, Edit Format, and Field Edit Format classes, determine how new objects will be added over the phone. All of the primitives are implemented as classes in the same object system used for storing the messages that callers add over the phone.

1.5 MACROS

In implementing the programs described in Chapter 2, some standard programming clichés have become apparent. Rather than always copy and edit large code fragments, I have added two macros to the programming language. These macros hide the similarities among the code fragments and parameterize the differences. One macro expands into a List Presentation and related objects that acts as a navigation menu, to which new options can be added by phone. The second macro expands into a Field Edit Format that creates a response list and links it into a field. Both macros are presented below, along with the code fragments that they expand into.

1.5.1 Extensible Navigation Menu

The first macro is a navigation menu for choosing which list to go to. The destination lists can themselves be navigation menus, or just lists of information objects. A system administrator can add additional destination lists as options in the menu. For example, in an events calendar, callers would select from a menu of event categories and system administrators could create new categories by phone.

Class: *Menu Macro*		
Field Name	Alternative Values	Description
Title	Prompt	Played in the list header, and when returning to the list after visiting somewhere else.
Description	Prompt	Played in the list header.
Menu Prompt	Prompt	Played when this object is an option in a menu.
Contents	[List Presentation]*	Initial items in the menu
Name for Objects	Text	Used in generating prompts
Privileges Required for Adding	NONE ǁ ADDING ǁ EDITING	The privilege level required to add a new option to the menu.
Form Header Prompt	Prompt	Part of what will be played in the header of the telephone form for editing the new menu option.
List Format for New Menu Options	[List Format]	What the List Format will be for the new List Presentation.

Menu Macro: *Sample Menu*	
Field Name	Value
Title	Event categories
Description	Please select a category
Menu Prompt	For event listings
Contents	--
Name for Objects	Category
Privileges Required for Adding	EDITING
Form Header Prompt	There are entry blanks, the title, description, and menu prompts for the new event category
List Format for New Menu Options	[Events List Format]

The same notation used for object classes also applies to macros. Unlike the object classes described above, however, instances of class Menu are not interpreted directly, but are instead "expanded" (just as a macro in Lisp can be thought of as involving textual substitution) into collections of instances of the built-in classes. The easiest way to understand the working of the Menu Macro is to follow an expansion of the sample instance above, which specifies an initially empty menu of event categories. In the expansion below, all of the instance-specific information is highlighted in **bold**. From the relative scarcity of bold text, it is easy to see why the Menu Macro contributes to more compact programs.

List Presentation:	
Field Name	Value
List	[List 1]
Filter	--
Format	[List Format 1]
Title	**Event categories**
Description	**Please select a category**
Menu Prompt	**For event listings**

Three prompts are taken directly from fields of the macro. A new list and a new List Format are created

List: *List 1*	
Field Name	Value
Contents	**--**

The list has as its initial contents specified in the macro. In this case, the menu will start without any options in it.

List Format: *List Format 1*	
Field Name	Value
Sort Order	--
Filter	--
Item Formats	[Default Menu Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	POSITIONAL
List Actions	[List Action 1]
Name for Objects	**Category**
Response List?	NO
Say Item Count?	NO

All the items in the list will be presented (no filter) in the order in which they are stored (no sort order). The Item Format is specified below. Callers will press buttons to skip through hearing the categories (SKIP advancement) and will have a single select button to select the category they are hearing (POSITIONAL) selection). The prompts for navigating through the menu will use the word "category", as specified in the macro.

Item Format: *Default Menu Item Format*	
Field Name	Value
Object Class	[List Presentation]
Field Formats	[Play Menu Prompt]
Select Action	[Goto Item]

Each option in the menu will be a List Presentation, the presentation of a category of event announcements.

Field Format: *Play Menu Prompt*	
Field Name	Value
Field	'Menu Prompt'
Field Name Spoken?	NO
Linked Item Format	--

The system will play only the 'Menu Prompt' field of each option in the menu.

Select Action: *Goto Item*	
Field Name	Value
Action Type	GOTO OBJECT
Action After Returning	RESET

When the caller selects an option, the system will go to that List Presentation. When the caller returns to this menu, the system will reset to the beginning.

List Action: *List Action 1*	
Field Name	Value
Privileges Required	**EDITING**
Button Location	ALL
Extension Format	[Extension Format 1]

Callers with editing privileges can add new options from anywhere in the menu. The privileges required are specified in the macro.

Extension Format: *Extension Format 1*	
Field Name	Value
Add to Current List?	YES
Other Lists to Add to	--
Location to Add	END
Class of Object to Add	[List Presentation]
Edit Format	[Edit Format 1]

Each event category is an object of type List Presentation. Thus, to create a new event category, a new [List Presentation] object is created. After it is edited, it will be added to the end of the current list. That is, it will become the last option in the menu.

Edit Format: *Edit Format 1*	
Field Name	Value
Field Formats	[Record Title] [Record Description] [Record Menu Prompt] [Make New List] [Edit Field Format 1]
Form Header Prompt	**There are entry blanks the title, description, and menu prompts for the new event category**

The caller will record three prompts for the new category, as specified in the first three Field Edit Formats below. The system will automatically create a new list, to hold the event announcements for the new category. The system also automatically fills in the List Format. The List Format to use was specified in the menu macro.

Field Edit Format: *Record Title*	
Field Name	Value
Field	[Title]
Initial Value	--
Include in Form?	YES
Data Type	VOICE
Max Length	30
Picklist	--
Validity Check	--
Field Description	--
Field Header Prompt	--

Field Edit Format: *Record Description*	
Field Name	Value
Field	[Description]
Initial Value	--
Include in Form?	YES
Data Type	VOICE
Max Length	60
Picklist	--
Validity Check	--
Field Description	--
Field Header Prompt	--

Field Edit Format: *Record Menu Prompt*	
Field Name	Value
Field	[Menu Prompt]
Initial Value	--
Include in Form?	YES
Data Type	VOICE
Max Length	30
Picklist	--
Validity Check	--
Field Description	--
Field Header Prompt	--

Field Edit Format: *Make New List*	
Field Name	Value
Field	[List]
Initial Value	* NEW LIST
Include in Form?	NO
Data Type	LINK TO OBJECT
Max Length	--
Picklist	--
Validity Check	--
Field Description	--
Field Header Prompt	--

Field Edit Format: *Field Edit Format 1*	
Field Name	Value
Field	[List Format]
Initial Value	**[Events List Format]**
Include in Form?	NO
Data Type	LINK TO OBJECT
Max Length	--
Picklist	--
Validity Check	--
Field Description	--
Field Header Prompt	--

Overall, then, the expansion of the menu macro fills in the details of how the menu will be presented to callers (e.g., automatic advance and positional selection) and generates the extension and editing formats for adding new event categories to the menu. It provides a compact way to specify a List Presentation of a particular kind, and can be used anywhere that a List Presentation object would be used.

1.5.2 Make Response List

The second macro is for filling in a field with a response list. For example, whenever someone adds a new comment to an issue discussion application, the system will create a new List Presentation, a place for other callers to add responses to the comment. The new list presentation needs a new list as one of its fields and will use a List Format specified in the macro. Again, a sample macro instance is presented along with the expansion of that instance into a program fragment.

Field Name	Alternative Values	Description
Field	[Field]	The field into which the new response list will be linked.
Response List Format	[List Format]	The List Format that should be used for presenting the new list of responses.

Make Response List: *Sample Instance*	
Field Name	Value
Field	'Responses'
Response List Format	[Sample List Format]

Field Edit Format:	
Field Name	Value
Field	**'Responses'**
Initial Value	* [Extension Format 2]
Include in Form?	NO
Data Type	LINK TO OBJECT
Max Length	--
Picklist	--
Validity Check	--
Field Description	--
Field Header Prompt	--

The initial value for the 'Responses' field will be a pointer to an object that is generated at run-time from the specification in the object below.

Extension Format: *Extension Format 2*	
Field Name	Value
Add to Current List?	--
Other Lists to Add to	--
Location to Add	--
Class of Object to Add	[List Presentation]
Edit Format	[Edit Format 2]

The linked object will be a [List Presentation] and will not be added to any lists.

Edit Format: *Edit Format 2*	
Field Name	Value
Field Formats	[Make New List] [Set Response List Format]

Field Edit Format: *Make New List*	
Field Name	Value
Field	'List'
Initial Value	* NEW LIST
Include in Form?	NO
Data Type	POINTER

One field of the new object will contain a pointer to a new List.

Field Edit Format: *Set Response List Format*	
Field Name	Value
Field	'List Format'
Initial Value	**[Sample List Format]**
Include in Form?	NO
Data Type	POINTER

The other field contains a pointer to the List Format specified in the macro.

Overall, the Make Response List macro provides a compact way of specifying that a field of an object should be filled in automatically with a new List Presentation object containing a new List and a List Format specified in the macro. It can be used anywhere that a Field Edit Format would be used.

2 CONCLUSION

This appendix has served as a programmer's manual for the HyperVoice language, describing the primitives and means of combination of the language primitives, and an informal description of their semantics. For listings of sample programs written in the HyperVoice language, see Appendix C. Appendix B discusses the pre-processor and interpreter for the language, which provide more details of the language semantics.

B The HyperVoice Interpreter

While appendix A served as a programmer's manual for HyperVoice users, this appendix is an implementor's manual for those who wish to replicate some or all of the features of the HyperVoice application generator. It describes in detail the two lower abstraction levels, the event and state machine layers. Then it shows how application layer program fragments are translated into state-machine layer programs.

1 THE EVENT LAYER

The event layer consists of procedure calls *play_file, stop_playing, record_file, stop_recording, beep, stop_beeps* ,and *silence*. These procedures are part of a library that is provided by the manufacturer of the hardware add-on card that digitizes speech and interfaces to a telephone line[5]. The event layer has a background process that enqueues tokens when certain events occur. These events include the end of playback of a file (either because it is finished or because the *stop_playing* command was issued), the end of beeps, and the beginnings and ends of touch-tones.

Programs can be written directly at this abstraction layer, but they will be long and complex. The programs will constantly monitor for and respond to low-level events such as files finishing playing or touch-tones interrupting the dialogue. The state-machine abstraction defines a uniform way of handling those low-level events and isolates that handling in an interpreter that is shared among all state-machine programs.

[5]The hardware card is Watson from Natural Microsystems and they call the software library the VAR Interface Driver.

2 THE STATE-MACHINE LANGUAGE AND INTERPRETER

As mentioned in the body of the thesis, the state-machine abstraction conceives of a telephone dialogue as a caller navigating through a graph of sounds. State-machine programs have *nodes* and *actions*, which either transition between nodes or modify the state-machine program in some way. The state-machine layer presented here is novel in its use of variables for the contents of sound_bytes and its use of subgraphs as subroutines. These features enable the creation of more modular, understandable programs.

The state-machine interpreter keeps track of a current state, namely a graph node and how much of it has been played so far. In the absence of button presses from the caller, the graph interpreter asks the event layer to play the recordings associated with the current node. If the event interpreter enqueues a button press, the state-machine interpreter first cancels any ongoing operation, then looks up the appropriate action to execute.

The remainder of this section describes the state machine language and interpreter in more detail. A kernel set of node contents and actions are introduced and then several extensions are made to the kernel. The first extension is that contents of nodes can be determined dynamically from variables and additional actions can set the contents of the variables. Second, conditional actions can act differently depending on the values of variables. Third, a call and return mechanism allows subgraphs to be treated as subroutines. Fourth, operations are introduced to read and write values from a database. Finally, several additional actions provide hooks for integration with the application layer.

<action> :=	
Rewind(int) ||	**Kernel**
Repeat ||	
Requeue-digit ||	
Record(filename) ||	
Hangup ||	
Goto(Graph-node) ||	
{<action>} ||*	
Set-variable(variable-name, <var_expr>) ||	**Variables**
Clear-variable ||	
If(<pred_expr>, <action>, <action>) ||	**Conditional Actions**
Call_subgraph(Graph-node) ||	**Subroutines**
Return(<var_epxr>) ||	
Add-field-value ([Object] , 'Field', <var_expr>) ||	**Database Operations**
Delete-field-value ([Object], 'Field', <var_expr>) ||	
Clear-field([Object], 'Field') ||	
Expand([List Presentation])	**Application Layer Actions**
Begin-form([Extension Format])	
Save-form	
Cut-item	
Paste-item	

Figure B.1 Summary of the syntax of all state-machine language actions. The right hand column indicates the subsections in which the actions are described.

2.1 KERNEL

There are two kinds of primitives in the graph programming language, graph nodes and actions. Each node consists of one or more sounds to be played, hereafter referred to as *sound_bytes*, a *transition table*, indicating what action to take on each of the twelve possible button presses, and a *timeout action*, to be taken if all of the sound_bytes are played without a button being pressed. A sound_byte can be a recording, a number, a string of digits, a date, a time, or a text string. When played back, dates, times, monetary values, numbers, and digit strings are "expanded" into sequences of voice recordings. For example, the date "062192" is expanded to the following recordings, "June," "twenty," "one," "nineteen," "ninety," "two." A text string would be played back through a text-to-speech synthesizer, though I have not hooked one up in the current implementation of HyperVoice.

There are two means of combination in the graph programming language, the *Goto* action and the *multi_action*. The goto action combines two nodes: callers can press a button to follow a link from one node to another. The multi_action combines two or more other actions, in much the same way that PROGN does in LISP or BEGIN...END blocks do in PASCAL: the component actions are executed sequentially, not in parallel. State machine programs shown in this chapter follow the C language notation for multi_actions, simply enclosing the component actions in {curly braces}.

In addition to the two means of combination, there are five additional actions: *rewind*, *repeat*, *requeue-digit*, *record*, and *hangup*. *Rewind* goes back a specified number of seconds in the playback of the current node's sounds. *Repeat* restarts at the beginning of the current node's sounds. *Requeue_digit* puts the button press that initiated this action back on the beginning of the digit queue. Requeue_digit is usually used as the first action in a multi_action, in order to execute the other actions without consuming a digit from the queue. *Record* initiates recording of whatever sounds next come over the phone, into a specified file. *Hangup* exits the graph, causing the current phone call to be terminated: no further input from the caller is accepted after the hangup action is executed.

Variable name	Values	Description
board_status	IDLE, PLAYING, RECORDING, BEEPING, SILENCE	What, if any, operation is ongoing
button_depressed?	TRUE, FALSE	Has a touch-tone started but not finished yet?
digit_queue		Touch-tones that have been detected, but not yet processed.
frame_position		How many frames of the current sound_byte have been played?

Table B.2 Variables used by the state machine interpreter to keep track of the event layer state of the dialogue.

In order to execute a graph program, the graph interpreter keeps track of the current graph node, the current sound_byte from that node, and the state variables that are described in Table B.2. When the graph interpreter asks the events layer to play or record, it sets the *board_state* variable to PLAYING or RECORDING, then waits while the events layer processes incoming board-level events. If the beginning of a touch-tone is detected, the state-machine interpreter sets the *button-depressed?* variable to TRUE, cancels any ongoing operation, and executes the appropriate action for that touch-tone. The rewind and repeat actions involve changing the current sound_byte and position within the current node, while links involve changing the current node. Before resuming play, the state machine interpreter waits until the end of the tone is detected.[6]

2.2 VARIABLES

<action> :=
 Set-variable(variable-name, <var_expr>) ||
 Clear-variable

The kernel graph language treats all button presses as navigation actions that cause transitions to different graph states. In many telephone dialogues, however, callers use touch-tones to enter values, such as zip codes or dates. To accommodate that, I introduce the notion of variables that hold values of any of the types that make acceptable sound_bytes (a file containing a voice recording, a number, a string of digits, a date, or a text string.)

[6] Note the analogy between beginning and end tone events and mouse down and up events in window systems. Here, the separation into two separate events improves efficiency: by the time the caller has finished the touch-tone press, the system has already processed the tone and is ready to start playing at the new location.

Two additional actions are added to the language, *clear_variable* and *set_variable*. The clear_variable action makes the specified variable unbound, while set_variable sets a variable to the result of a variable expression.

Sound_bytes for a graph node can be variable expressions rather than just constant values. The syntax for variable expressions is given in Figure B.3. Variable expressions can be constant values, direct lookups of variable values, or constructed from functions that return values can be used in variable expressions. *Strlen* is one such function. It returns the number of digits in a variable that is represented as a string. *Append_digit* returns the result of adding a digit to the end of another string value. Section B.2.5 will introduce the function calls *count_values*, *get_ith_value* and *get_matching_value* that retrieve values from a database. Section B.2.4 will introduce the function *call_subgraph*, which passes control to a subgraph that is expected to return a value.

Conceptually, variable expressions are re-evaluated continuously: whenever a variable's value changes, all variable expressions that depend on it are re-evaluated. For implementation efficiency, variable expressions are re-evaluated just before they are used.

<var expr> := constant || variable name || <function call>

<function call> :=
    strlen(<var expr>) ||
    append_digit(<var expr>) ||
    count_values([Object], 'Field') || **database operations**
    get_ith_value([Object], 'Field', <var expr>) ||
    get_matching_value([Object], 'Field', <var expr>, 'Field')
    call_subgraph(<graph node>) **subroutines**

Figure B.3 The syntax of variable expressions. The right column indicates the sections where the functions are described.

2.3 CONDITIONAL ACTIONS

<action> :=
    If(<pred_expr>, <action>, <action>) ||

The state-machine language includes a conditional construct. Suppose a caller is asked to type in a date, which will be stored in a variable, but is given the option of canceling entry of the date. The program should act differently depending on whether a complete date is entered. The *if* construct provides that facility. It takes three parameters, the predicate expression to check, the action to take if the predicate expression is satisfied, and an alternative action to take if it is not satisfied.

Predicate expressions examine the contents of one or more variables. The syntax of the legal predicates is specified by the equations in Figure B.4. Intuitively, a predicate expression compares two values, which can be constants, variables, or function calls. The semantics of the predicates follow the obvious intuitions. The predicate symbols are overloaded: the symbol < is interpreted differently when the two values being compared are dates than when they are numbers.

<pred expr> :=
    (<pred expr> AND <pred expr>) ||
    (<pred expr> OR <pred expr>) ||
    <var expr> <pred sym> <var expr>

<pred sym> := < || <= || == || >= || >

Figure B.4 The syntax of predicate expressions.

2.4 CALL AND RETURN

<action> :=
    Call_subgraph(Graph-node) ||
    Return(<var_epxr>)

<function call> :=
    Call_subgraph(<graph node>)

The graph language and interpreter presented thus far are useful, but do not facilitate the construction of modular programs because they do not allow the creation of sub-routines. It is not possible, for example, to create a single pause node that can be accessed from every other node in the system, because once the graph interpreter transitions to the pause node, it will not remember where to continue at the conclusion of the pause. Similarly, it is not possible to create a subgraph that collects and returns a date, then use that subgraph anywhere that it is appropriate. In this section, I present extensions to the graph language to accommodate a call and return mechanism that treats subgraphs as subroutines, complete with parameter passing and returned values.

Two new actions are included in the language: ***call_subgraph*** and ***return***. The call_subgraph action transitions to the destination node, and passes parameters, but keeps a marker, so that when a subsequent return action is performed, there will be a transition back to the current node, whence processing can resume. The return action, as well as returning to the last marker, can pass back a value, so that call_subgraph can be used as a function call in any variable expression.

To illustrate the advantages of a call and return mechanism, suppose that touch-tone button **8** is the pause button throughout an application. When a caller presses **8**, the system should say, "Paused," and then should wait until the callers presses **8** again to continue. Other button presses should end the pause and initiate whatever action is associated with that button in the transition table of the current graph node. One pause node containing two sound_bytes implements this functionality (see Figure B.5). The first sound_byte is the recording, "Paused," and the second is a very long period of silence. Button **8** in the pause node's transition table is a return action, with no returned value. Other button presses cause a return from the pause node without consuming the button press. All of the regular graph nodes have button **8** set to a call_subgraph action, with the pause node as the destination. Thus, for example, when **8** is pressed from node X, there is a transition to the pause node. If **8** is pressed again, it returns to node X and continues playing. If some other button, say **9**, is pressed, then it returns to node X and executes the action associated with **9** in node X.

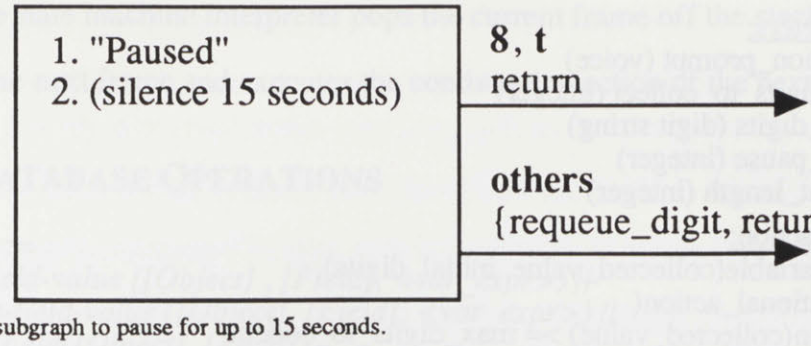


Figure B.5 A subgraph to pause for up to 15 seconds.

The advantages of parameter passing and returned values are best illustrated by a subgraph that collects a string of digits from the caller. This subgraph takes several parameters, including the number of digits to be collected, as shown in Figure B.6. Because of the input parameters, this subgraph can be used in many different data collection situations. For example, by specifying a different prompt, the caller can be informed of what kind of value to enter and what it will be used for. The timeout length can be varied to give callers more or less time between button presses before assuming that the caller is not going to enter any more digits. An initial value can be passed, if the caller is supposed to append digits to an existing value. Finally, a caller need not enter a delimiter to end data entry: data entry is terminated automatically after the maximum number of digits have been collected.

The returned value makes it possible to call this subgraph as a function inside any variable expression. Thus, it is possible, for example, to create an add_field_value action that adds the value returned from a call to this subgraph. It is also possible to call this subgraph inside a multi_action. For example, an action might first set a variable to be the result of calling the subgraph in Figure B.6. The next action in the multi_action could then do a conditional branch on the value of the variable. In fact, in section B.3.5 below, on telephone forms, just such a multi_action is used to collect values: if an acceptable value is collected, it is then written into the database.

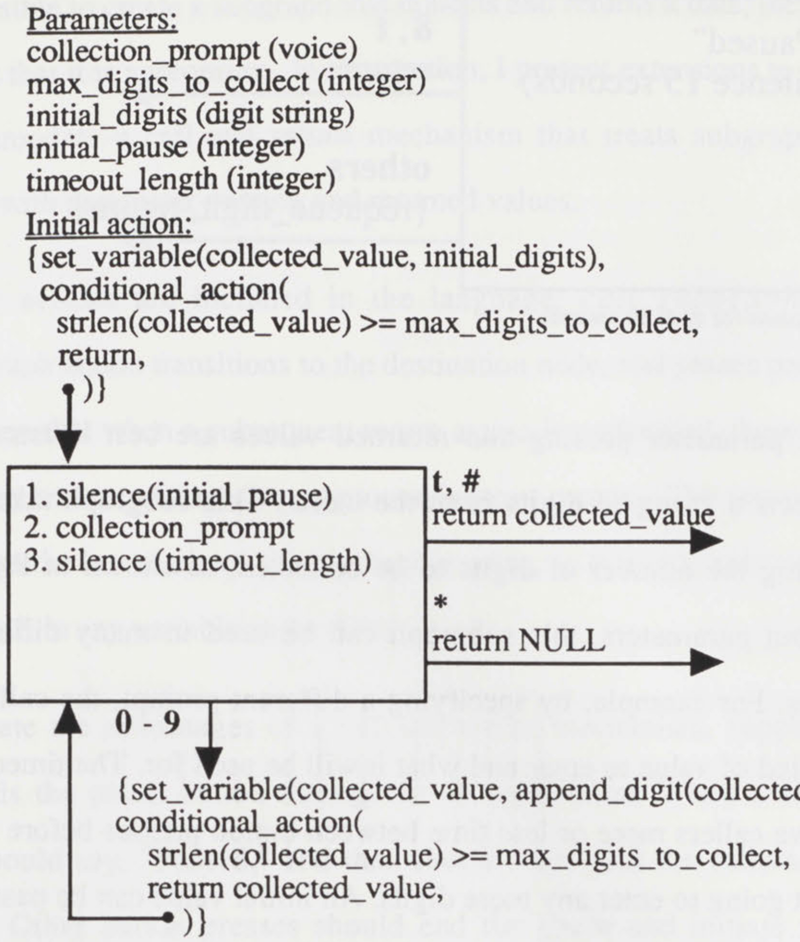

Parameters:
collection_prompt (voice)
max_digits_to_collect (integer)
initial_digits (digit string)
initial_pause (integer)
timeout_length (integer)
Initial action:
{set_variable(collected_value, initial_digits),
conditional_action(
strlen(collected_value) >= max_digits_to_collect,
return,
)}
1. silence(initial_pause)
2. collection_prompt
3. silence (timeout_length)
t, #
return collected_value
*
return NULL
0 - 9
{set_variable(collected_value, append_digit(collected_value)),
conditional_action(
strlen(collected_value) >= max_digits_to_collect,
return collected_value,
)}

Figure B.6 A subgraph that collects digits. The value collected is returned from the subgraph.

To accommodate the new language primitives, the graph interpreter has to be changed. Rather than keeping track of a single current graph node, it keeps a stack of graph frames. Each stack frame contains a graph node, a continuation action, and a place for a returned value. The node in the top frame on the stack is treated as the current node, and its contents are played. To execute a goto action, the interpreter replaces the graph node in the top stack frame. To execute a call_subgraph action, it sets a continuation action for the current stack frame and pushes the destination node onto to the stack. The continuation action is necessary because the call_subgraph action may come in the middle of a sequence of actions or be used as a function call. In either case, processing should continue where it left off after a return from the subgraph. To execute a return action, the state machine interpreter pops the current frame off the stack, sets the returned value of the next frame and executes the continuation action of the next frame.

2.5 DATABASE OPERATIONS

```
<action> :=
    Add-field-value ([Object] , [Field], <var_expr>) ||
    Delete-field-value ([Object], [Field], <var_expr>) ||
    Clear-field([Object], [Field])

<function call> :=
    count_values([Object], 'Field') ||
    get_ith_value([Object], 'Field', <var expr>) ||
    get_matching_value([Object], 'Field', <var expr>, 'Field')
```

Several functions and actions read and write values from a database. The database is object-oriented as described in Chapter 4. Briefly, a class specifies a set of fields that each instance will contain. An instance (also called an object) has zero or more values of any supported data type in each of the fields. The supported data types are recording, text, date, integer, and pointer to another object.

Three new functions are provided for reading values from fields. *Count_values* returns a count of the number of values currently stored in a specified field. *Get_ith_value* retrieves the i^{th} value from a field, where i is a parameter. *Get_matching_value* is available only when the values of a field are pointers to other objects. It takes as parameters an object, the field from which to retrieve a value, a variable expression that is used as the key for the search, and a field of the linked objects to compare against the key. Get_matching_value returns the first object whose specified field has a value that matches the key value. All of these functions can be used in any variable expression.

Three actions change the values of fields of objects. *Add_field_value* evaluates a variable expression and adds the resulting value in the ith position in the list of values of a specified field. To add a value at the end of the list, the parameter i can be computed as count_values on the field. *Delete_field_value* removes the ith value from the list of values for a specified field. *Clear_field* removes all the values for a specified field.

As an example, consider a graph node that plays back the closing stock price of the ACME shoe repair and financial services Corporation ("one-stop shopping for the investor who does his own legwork."). Assume that the object ACME in the database has a field called 'closing price' that is always updated by some outside source to contain a single value, the most recent closing price. The graph node contains two sound_bytes. The first is a recording of the company name, "ACME Hardware." The second sound_byte is the variable expression, *get_ith_value(ACME, 'closing price', 1)*, which retrieves the first value from the 'closing price' field of the object ACME. This example illustrates how some useful changes to the space of possible dialogues can be accommodated by setting sound_bytes of graph nodes to be retrieved values from a database.

2.6 CALLBACKS TO THE APPLICATION LAYER

The state-machine layer includes few actions that involve application layer constructs. All of these actions change the state-machine program, and change the state of the state-machine interpreter. *Expand* converts a List Presentation object into a state-machine subgraph for browsing through the information in a list of objects. *Cut-item* and *Paste-item* are used to cut and paste objects in lists, then to regenerate the graph for browsing through the contents of the list. *Begin-form* generates a telephone form subgraph for editing a new object, *process-validity-check* checks a value entered against a Validity Check object, *reset-field-graph* regenerates part of the form graph after the contents of a field of the object have been changed, and *save-form* saves the new object that has been edited and exits from the telephone form subgraph. These actions are the subject of section B.3.

2.7 RELATED RESEARCH

Chapter 4 already discussed audio interface toolkits that are based on the state-machine layer. Visual hypermedia systems such as HyperCard are also based on a variant of this model, though graph nodes consist of text or graphics to display rather than sounds to play back. While some of these systems provide a richer set of functions and actions in their scripting languages, including iterative constructs, none treat subgraphs as subroutines with parameters and returned values.

2.7.1 Visual hypermedia

Visual hypertext systems have explored more of the space of possible graph models. HyperCard is the best known entry in this category [Goodman 1990]. Cards in HyperCard serve the dual role of graph node and database record. For example, an action can set the value of a variable (or the field of another card) to be the contents of a particular field of a card. The current contents of a card can also be displayed on the screen. HyperCard does not, however, allow cards to automatically get their values from a variable or another card. HyperCard programmers can accomplish a subgraph call by pushing the current card on a system-provided stack and then branching to the subgraph. The calling and called cards can communicate through global variables, but there is no provision for parameter passing or returned values.

KMS [Akscyn, et al. 1988] and the Symbolics Document Examiner [Walker 1987] separate the underlying database from the graph nodes. Graph nodes can automatically look up their contents in an underlying database.

Often, visual hypermedia systems present more than one graph node simultaneously, in tiled or overlapping windows. For example, Trigg includes the notion of a Tabletop Card [Trigg 1988]: when it is opened, a whole set of nodes are opened at once. In that case, the state machine model breaks down somewhat, since the user can interact with a window of her own choosing. Two papers presented a formalism to handle this situation [Hill 1987, Wellner 1989] by keeping multiple active state machines.

Zellweger adds a full procedural programming language to a hyperdocument system [Zellweger 1989]. The language is not concerned with the contents of nodes, however, but only the sequencing of them. It does not allow the contents of nodes to be variables from programs or values retrieved from a database.

2.7.2 Window and Mouse Systems

State-machine abstractions are one popular substrate for screen-based User Interface Management Systems [Green 1986]. When screen-based systems switched from command-line to window and mouse interfaces, however, the popularity of the state-machine model declined. One difficulty is that many icons are typically available on the screen. Interactions with some of the icons are local, in that they will not affect other portions of the screen. One solution to this problem is to allow several state machines to run concurrently [Hill 1987, Wellner 1989].

Another problem with state-machine specifications of direct manipulation interfaces is that state-changes from even low-level events such as mouse-down and mouse-up require behavior that depends on application-level constructs. For example, clicking on an icon may cause the representation of an information object somewhere else on the screen to be highlighted, in a manner that depends on the current contents of the object. Hence, the state-machine abstraction does not accomplish the goal of separating the user interface specification from the application specification. To some extent, this problem carries over to the use of state-machine representations of telephone dialogues: the state-machine representation intermingles information about the application (what information is being presented and whether it can be modified) with information about how callers execute actions. This is one advantage of automatically generating the state-machine representations from application-layer specifications that exclude information about how callers will execute actions, as will be discussed in the next section.

3 THE APPLICATION INTERPRETER

With the state-machine abstractions as background, this section turns to a description of the application layer interpreter. It includes both static and dynamic components. The static pre-processor generates the text of prompts that need to be recorded before any telephone calls can be handled. The dynamic component interprets HyperVoice programs at run-time. It runs as a two-step process: first a part of the program is translated into a state-machine program, then the state-machine interpreter runs the translated program. When callers add new data objects by phone, parts of the state-machine layer program are regenerated.

The remainder of this section presents the translation of application program fragments into state machine programs. Throughout, nodes include recordings whose texts were generated by the pre-processor. The method of generating those prompts is noted whenever the prompts are used. Thus, the descriptions mix the actions of the pre-processor and the interpreter.

Since the complete state-machine program for even a simple graph-layer program would be difficult to fit on a single page, I present the translation in modular pieces. I begin with login procedures. Then, I proceed to lists, where each object in the list is assigned to a single graph node. Next, the graph node for a single object is expanded into a group of nodes, one for each field of the object, plus one help node for each possible command that is available. Then, I show how menus are implemented. Finally, I show how editing formats are used to generate telephone form graphs.

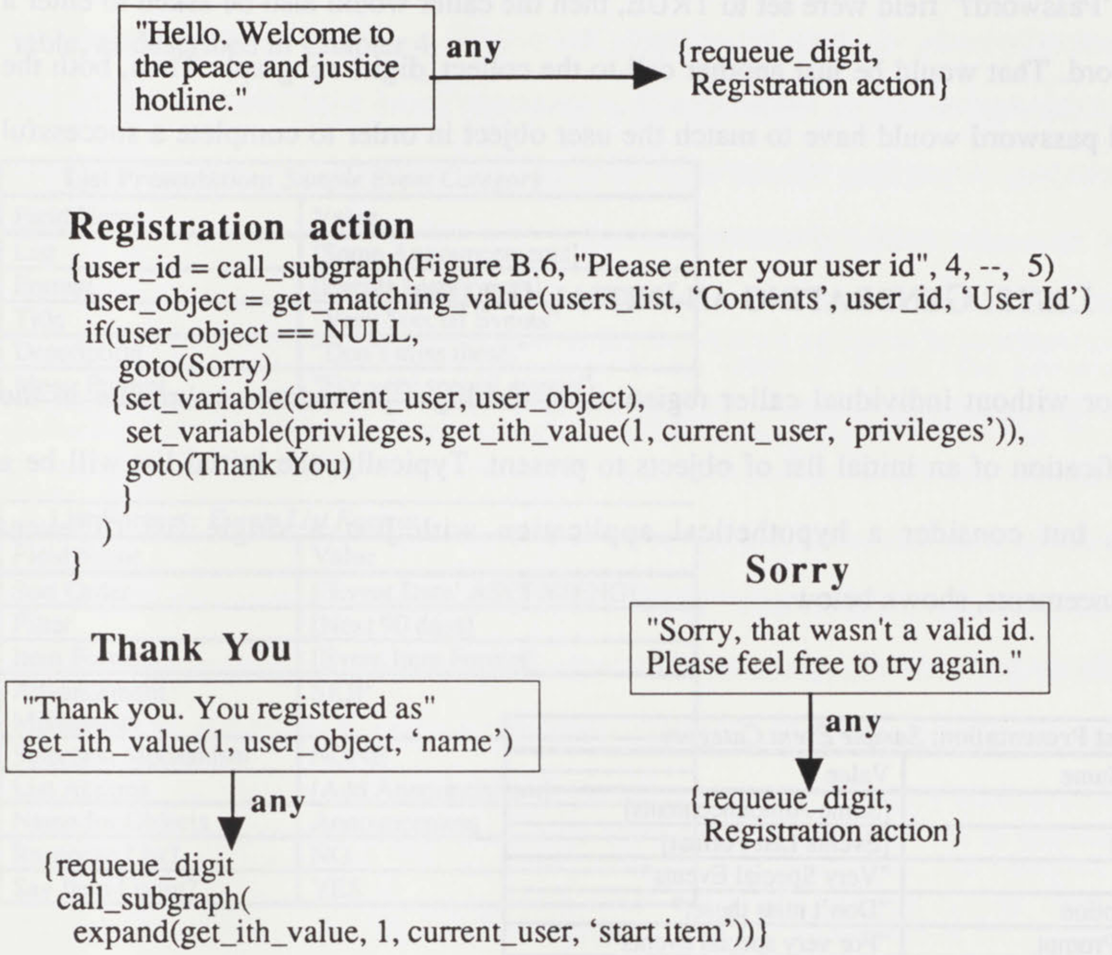
3.1 LOGIN PROCEDURES

An application with no access controls, such as the Boston Peace and Justice Events calendar, begins directly with the presentation of a menu or other list of information objects. This is specified by a Login object whose 'Register?' field is set to TRUE. If, on the other hand, registration is required, then a state-machine program like that in Figure B.7 is generated. Consider the Login specification from the teachers' curriculum line application, shown below. The complete program listing for that application is shown in Appendix C.

Login: *Curriculum Line*	
Field Name	Value
Register?	TRUE
User Id Length	4
Users List	[All Teachers]
Password?	FALSE
Greeting	Hello, welcome to the Math-cubed curriculum line.
Start Item	[The Main Menu]
Special List Jumps	[Goto all questions]

The initial graph node contains a recording of the prompt from the 'Greeting' field. After playing that prompt, the subgraph from Figure B.6 is called, to collect 4 digits. When the subgraph has finished collecting the four digits, they are returned as a string and the variable user_id is bound to that string. Next, a database operation searches in the list [All Teachers] for a User object that has a value in the 'User Id' field that matches the string that was collected. If one is found, the user_object variable is bound to it. The next state-machine action that is executed is a conditional on whether a matching User object was found. If none was found, then it goes to an error message node, called *Sorry*, and then begins the process again by collecting a four-digit string.

If a User object was found that matched the four digits entered, then the system sets the current_user variable and sets the privileges variable to be the value in the 'privileges' field of the User object. Then, the state-machine interpreter transitions to a node that confirms a successful login. From that node, it initiates the translation of the List Presentation object contained in the 'start item' field of the User Object.


"Hello. Welcome to the peace and justice hotline."
any
{requeue_digit, Registration action}
Registration action
Sorry
"Sorry, that wasn't a valid id. Please feel free to try again."
any
{requeue_digit, Registration action}
Thank You
"Thank you. You registered as" get_ith_value(1, user_object, 'name')
any

```
{user_id = call_subgraph(Figure B.6, "Please enter your user id", 4, --,  5)
 user_object = get_matching_value(users_list, 'Contents', user_id, 'User Id')
 if(user_object == NULL,
    goto(Sorry)
    {set_variable(current_user, user_object),
     set_variable(privileges, get_ith_value(1, current_user, 'privileges')),
     goto(Thank You)
     }
   )
}
```

```
{requeue_digit
 call_subgraph(
   expand(get_ith_value, 1, current_user, 'start item'))}
```

Figure B.7 The graph with which a caller interacts in order to register at the beginning of a call.

There are three interesting things to note about this state-machine graph. First, the expert caller can immediately start typing in her user id without waiting to listen to the initial greeting: the digit that interrupts the greeting will be requeued and used as the first digit of the user id that is collected. The ability to type-ahead at any time is an important feature of the interface style that is built into the application interpreter. Second, the generic subgraph for collecting a string of digits facilitates modularity. Third, the database operations also facilitate modularity. For example, the user's name to play in the Thank You node and the initial list to present can be looked up. Without that, it would be necessary to generate separate login confirmation graph nodes for each potential user.

If the 'Password?' field were set to TRUE, then the caller would also be asked to enter a password. That would be just another call to the collect_digits subgraph. Then, both the id and password would have to match the user object in order to complete a successful login.

3.2 LISTS: GENERATING AN INITIAL GRAPH

With or without individual caller registration, the login procedures culminate in the identification of an initial list of objects to present. Typically, the initial list will be a menu, but consider a hypothetical application with just a single list of event announcements, shown below.

List Presentation: *Sample Event Category*	
Field Name	Value
List	[Some Announcements]
Format	[Events List Format]
Title	"Very Special Events"
Description	"Don't miss these."
Menu Prompt	"For very special events"

The presentation of a list of objects follows a navigation metaphor, where the caller starts at the beginning of the list, and presses buttons to move forward and back between objects. I present the translation of a List Presentation object in two stages. This subsection shows the coarse structure of the state-machine program, in which each object is presented in one graph node. The following section then shows a finer structure, where the node for each object is split into one node per field of the object, plus one node to prompt for each possible action to take.

In the diagrams that follow, words rather than numbers label the transitions between nodes. This emphasizes how callers are expected to think about what the transition does. It de-emphasizes the particular mapping of the word labels to buttons on the telephone keypad. The interpreter looks up the mapping of word labels to telephone buttons in a table, as described in Chapter 4.

List Presentation: *Sample Event Category*	
Field Name	Value
List	[Some Announcements]
Format	[Events List Format]
Title	"Very Special Events"
Description	"Don't miss these."
Menu Prompt	"For very special events"

List Format: *Events List Format*	
Field Name	Value
Sort Order	['Event Date' ASCENDING]
Filter	[Next 90 days]
Item Formats	[Event Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	NONE
List Actions	[Add Announcement]
Name for Objects	Announcement
Response List?	NO
Say Item Count?	YES

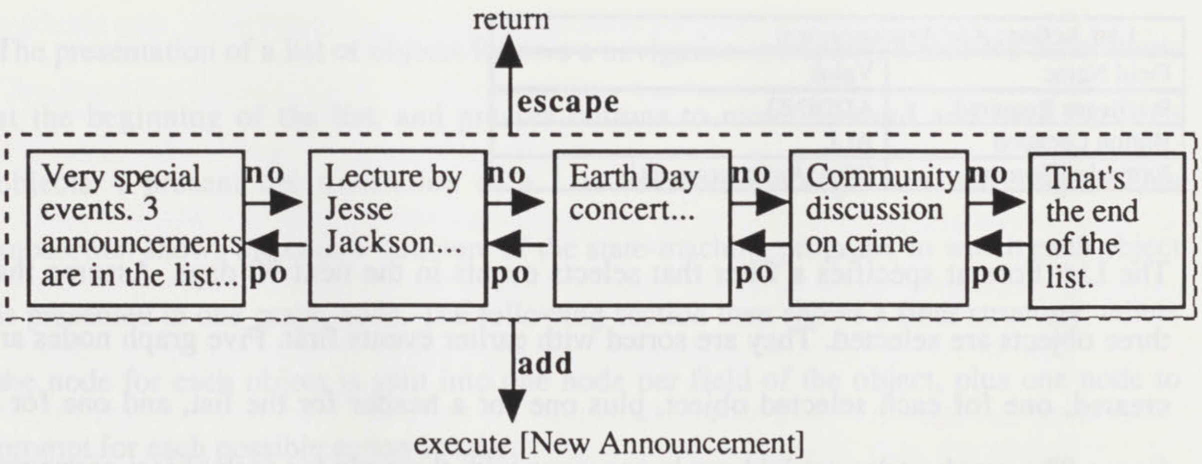
List Action: *Add Announcement*	
Field Name	Value
Privileges Required	ADDING
Button Location	ALL
Extension Format	[New Announcement]

The List Format specifies a filter that selects events in the next 90 days. Assume that three objects are selected. They are sorted with earlier events first. Five graph nodes are created, one for each selected object, plus one for a header for the list, and one for a footer. The graph nodes are laid out in sequence with the node for each object connected only to its neighbors, as shown in Figure B.8, repeated from Chapter 4.

The 'advancement mechanism' field of the List Format determines how the nodes in a list will be linked. If it is set to SKIP, then the caller will use actions **next object** (**no**) and **previous object** (**po**) to move forward and back in the list. If it is set to AUTO, then callers advance to the next item by waiting until the current item is finished playing. In state-machine terms, the links from Figure B.8 labeled **po** would be eliminated and the ones labeled **no** would be relabeled **t** (for **timeout**). If the field is set to BOTH, then either waiting or pressing buttons will work. The **po** links would remain, and the links labeled **no** would be labeled with both **no** and **t**.

For convenience of notation, boxes in the diagrams inherit links from the outside in. That is, any link emanating from an enclosing box is available from any of the nodes inside it. For example, the caller can press **escape** from anywhere in the list to return to the previous list that was presented, or press **add** from anywhere in the list to start the addition of a new announcement. By the way, the fact that **add** is available anywhere, rather than just attached to the header of the list, or to the nodes for the objects, is determined by the 'Button Location' field of the List Action object above.

The contents of the list header are generated by piecing together several sound_bytes. The first sound_byte is the contents of the 'Title' field of the List Presentation. If 'Say Item



return
escape
Very special events. 3 announcements are in the list...
no
po
Lecture by Jesse Jackson...
no
po
Earth Day concert...
no
po
Community discussion on crime
no
po
That's the end of the list.
add
execute [New Announcement]

Figure B.8 The initial expansion of the event announcements into a graph. There is one node for each of the event announcements, plus one each for the list header and list footer nodes.

Count?' is set to TRUE, as it is in the example, then the next sound_bytes are a numeric value for the count of how many announcements passed the filter, and the prompt, "announcements are in the list." The pre-processor generated the latter text by combining the word "announcement," taken from the field 'Name for Items' in the List Format, with the phrase, "are in the list." That sound_byte is followed by the contents of the 'Description' field of the List Presentation. Finally, there is the prompt, "A hint for getting through these announcements quickly: interrupt the spoken voice at any time. **<next object>** skips ahead to the next announcement, and **<commands>** will tell you about other helpful commands." **<next object>** and **<commands>** in this text stand for the button numbers to which the two actions are mapped. Again, the pre-processor generates the text of this prompt, using the word "announcement" from the 'Name for Items' field.

If the field 'Response List?' were set to TRUE instead of FALSE, two additional sound_bytes would be inserted in the list header, just after the 'Title' recording. The first would be the text of the prompt in the 'Response List?' field. The second would be the contents of the field specified, taken from the last object that was visited before the current list was presented. For example, this feature is used in the opinion forum applications, where each comment has a pointer to a List Presentation for responses. A caller listens to the comment, then follows the link to the responses. That List Presentation has no 'Title' recording, but the list is well described by playing, "responses to:" and the then the headline of the original comment.

3.3 SUBDIVIDING THE CONTENTS NODES

Thus far, the contents of the nodes for each object have been treated as black boxes. In fact, a caller can navigate can skip back and forth between hearing the contents of the fields of an object as well as skipping back and forth between objects. In addition, no mention has been made thus far of navigation prompts that inform callers of what actions are available. These are also attached to the node for each object and callers can skip through them as well.

The contents of the node for the Earth Day concert from Figure B.8 are determined by the Item Format object, which selects some of the fields, orders them, and specifies whether the field names will be played back before the contents. Figure B.9 shows the internal structure of the node for that event announcement, with one subnode for each field that is played back, and one subnode to prompt for each action that is available. Note that if a field's contents were empty, the subnode for it would be omitted.

Note several important things. First, if the caller does not press any buttons, the system will play the contents of the sub-nodes, one after the other, and then play all of the help prompts, one after the other. After completing playback of the help nodes, the timeout arrow back to the headline indicates that it will start replaying the whole event

Item Format: *Event Item Format*	
Field Name	Value
Object Class	[Event Announcement]
Field Formats	['Headline' NO] ['Dates' YES] ['Time' YES] ['Location' YES] ['Sponsoring Organization' YES] ['Contact Number' YES] ['Details' YES]
Select Action	--

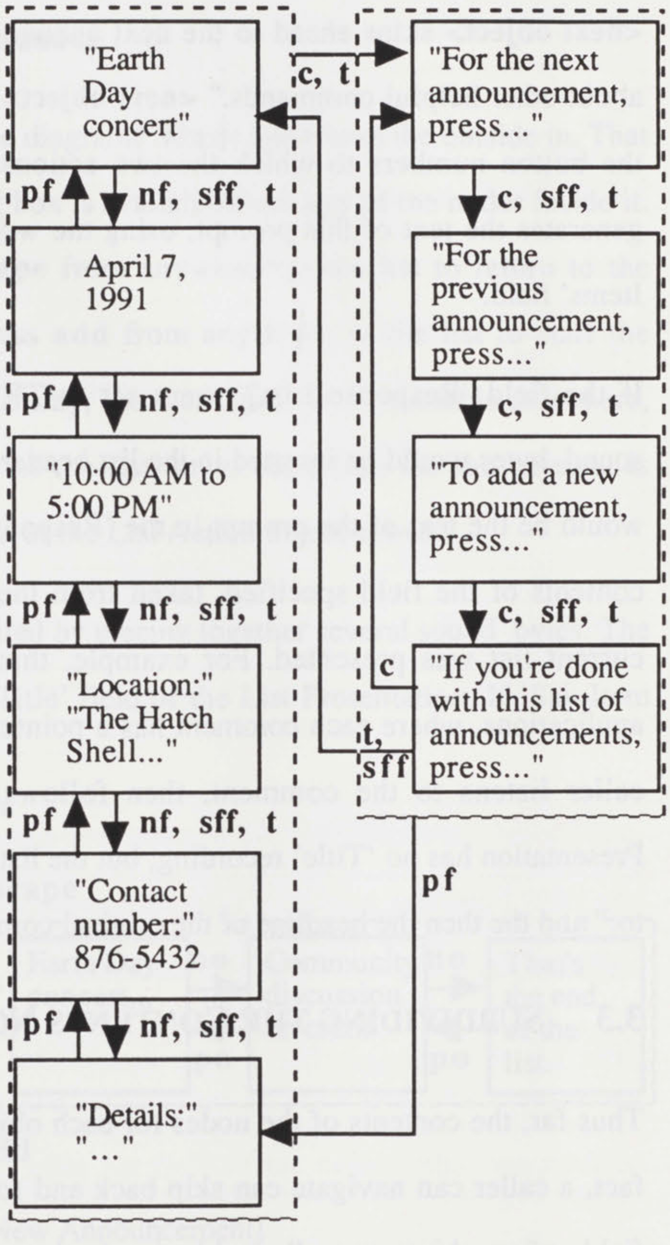

"Earth Day concert"
pf
nf, sff, t
April 7, 1991
pf
nf, sff, t
"10:00 AM to 5:00 PM"
pf
nf, sff, t
"Location:" "The Hatch Shell..."
pf
nf, sff, t
"Contact number:" 876-5432
pf
nf, sff, t
"Details:" "..."
c, t
"For the next announcement, press..."
c, sff, t
"For the previous announcement, press..."
c, sff, t
"To add a new announcement, press..."
c, sff, t
c
"If you're done with this list of announcements, press..."
t, sff
pf

Figure B.9 The substructure of one event announcement.

announcement. Second, the pre-processor generates the prompts for navigation, using the word "announcement" taken from the List Format specification, and using the mapping of commands to telephone buttons. For example, if the **next object** command is mapped to **9** on the telephone keypad, then the first navigation prompt would be, "For the next announcement, press 9." This ensures a uniform style for the prompts, and because it ensures that the prompts suggest the button presses that actually execute the actions. Third, there are no prompt that tell callers about the **next field**, **previous field**, and **commands** actions.

This last point is an example of a more general phenomenon, that if an action can be accomplished by timeouts or by explicit command, then prompting people to use the explicit commands will interrupt the flow that the automatic advance with timeouts attempts to create. Suppose, for example, that the 'Advancement Mechanism' field of the List Format were set to BOTH. Then, there would be automatic advance from one event announcement to the next, and prompting for the **next object** and **previous object** commands would interrupt the flow. In cases where skipping operations between nodes are optional, either the flow has to be interrupted, or callers need to be taught the skipping operations separately rather than prompting for them whenever they are available.

This was an important consideration in Chapter 3, on the Skip and Scan interface style, which argued that there is a fundamental tradeoff between how quickly callers will learn to use skipping actions and the additional costs that are incurred before they have learned to skip. Chapter 3 also argued that with navigation prompts is better err on the side of reduced initial costs. The navigation prompts generated by the application interpreter conform to this guideline: there is both automatic advance and skipping between navigation prompts.

3.4 Selection: Menus

The List Format above does not allow for selection, but others do. This section describes how menus, which are lists with selection, are generated. HyperVoice accommodates two selection mechanisms, positional and numeric. Chapter 5 addressed the relative merits of these mechanisms. With positional selection, a single select button selects whatever object the caller is positioned on. Figure B.10 shows the translation of a menu with positional selection. The **select** action for each object is attached to the graph node for that object: to select an object, the caller goes to the node for that object and then presses the **select** button. Whether the action returns the object as a value or expands another List Presentation object is determined by the Select Action object, and is not important here.

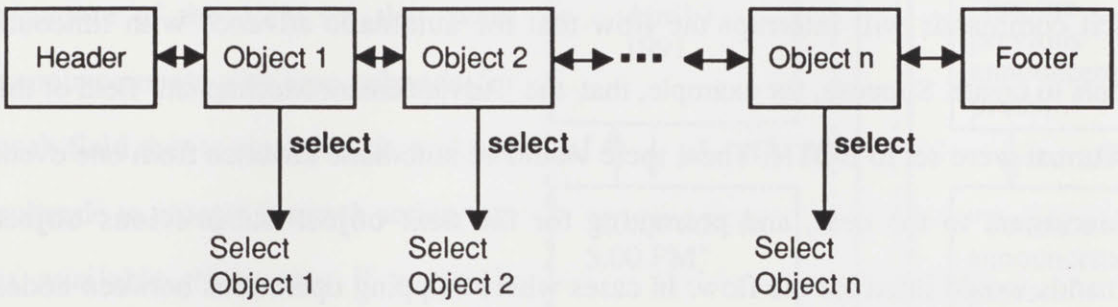
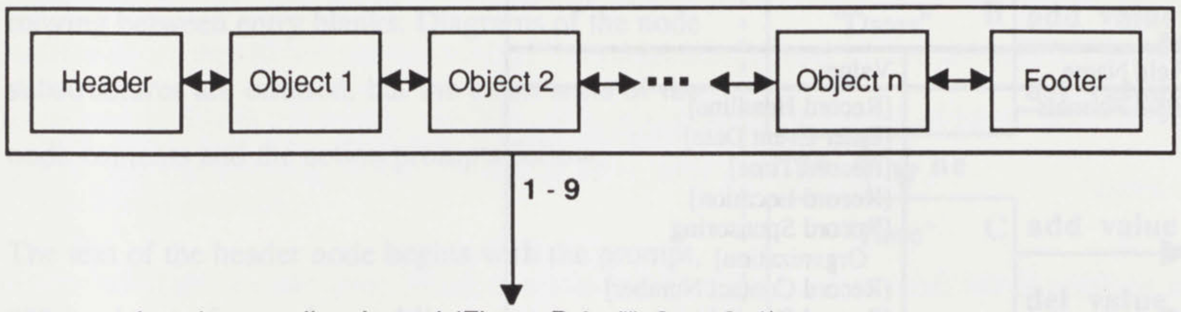


Figure B.10 A menu with positional selection.

With numeric selection, the caller enters a number to make a selection. One selection action is still created for each object in the list, but callers need not be listening to the node for the selected object when they enter numeric selections. Figure B.11 illustrates a numeric selection menu. Note that a box is drawn around the whole list, to indicate that the action to initiate entering a number is available throughout the list. This will only work if the **next object** and **previous object** actions are mapped to *****, **0**, or **#**, since **1** through **9** are used to begin entry of numeric selections.

The generic digit string collection subgraph is used again, this time to collect enough digits to make a selection. If there are fewer than ten objects, then one digit is sufficient, and the digit will be returned immediately from the subgraph. When collecting more than one digit, as in Figure B.11, no collection prompt is given and collection is terminated after a short timeout (1 second). This has the drawback that people have to type fast to make two digit selections, but the alternative of ending single digit selections with a delimiter such as # is even less palatable. The digit collection subgraph returns a string of digits that is used to pick out one of the actions from a nested if statement. If the number entered is larger than the number of objects to select from, it calls a standard error subgraph (not shown) to notify the caller that no selection was made.



```
{number = call_subgraph(Figure B.6,  "", 2, --, 0, 1)
  if(number == "1", Select Object 1,
    if (number == "2", Select Object 2,
      ...
        if (number == "n", Select Object n)
          call_subgraph(invalid-action(number))...)}
```

Figure B.11 A menu with numeric selection from a list of 10-99 options.

3.5 FORMS

Consider again the events calendar example used to illustrate the translation of a List Presentation object. While listening to the announcements, a caller can press **add** to initiate execution of the [New Announcement] Extension Format. The interpreter creates a new object of type Event Announcement and uses the Field Edit Formats to determine any initial values for fields. Then, it creates a telephone form, which is a subgraph for editing the object, and calls that subgraph. The subgraph has one major node for each field of the object that some Field Edit Format says to include in the form, plus a header node and a footer node, as shown in Figure B.12. It also sets a local variable *new_object*, so that actions in the form subgraph can edit the fields of the new announcement object.

Extension Format: ***New Announcement***

Field Name	Value
Add to Current List?	YES
Other Lists to Add to	[All Announcements]
Location to Add	END
Class of Object to Add	[Event Announcement]
Edit Format	[Edit New Announcement]

Edit Format: ***Edit New Announcement***

Field Name	Value
Field Formats	[Record Headline] [Enter Event Date] [Record Time] [Record Location] [Record Sponsoring Organization] [Record Contact Number] [Record Details] [Fill in Category]
Header Prompt	"You'll be asked to record a headline, enter the date, and record several other important pieces of information. Feel free to skip any entry blanks that are irrelevant to your announcement, but please be sure to include a contact phone number."

In Figure B.12, the commands **next entry blank (ne)** and **previous entry blank (pe)** initiate movement between entry blanks. In all of the field trials described in this thesis, **next entry blank** and **next object** have been mapped to the same key, **9**. Thus, a caller who has learned to skip through the objects in a list can use the same buttons to skip through the entry blanks in a form. Perhaps a better mapping, though, would map **next entry blank** and **next field** to the same key, so that the same button would advance between fields during recording and playback.

The nodes in the diagram have substructure analogous to the substructure given for list nodes above in Figure B.9. In a telephone form, however, the node for each entry blank contains only one subnode for the contents of the field. The remainder of the subnodes are for prompts as to commands that are available, including saving or canceling the form, adding and deleting values from the field, and moving between entry blanks. Diagrams of the node substructures are omitted, but the exact texts of the node contents and the action prompts follow.

The text of the header node begins with the prompt, "This is a form for adding a new event announcement. It works just like a paper form, but instead of writing the information, you'll be asked to record it or enter it using the buttons on your telephone." The pre-processor generates this text by splicing the phrase "event announcement", which is the name of the object class, into the rest of the text, which is shared among all telephone forms. The next sound-byte is a recording of the 'Header prompt'. The programmer is responsible for writing the text of that prompt.

Three actions are available from the form header, and hence three help prompt subnodes are generated. The first help prompt states, "To go to

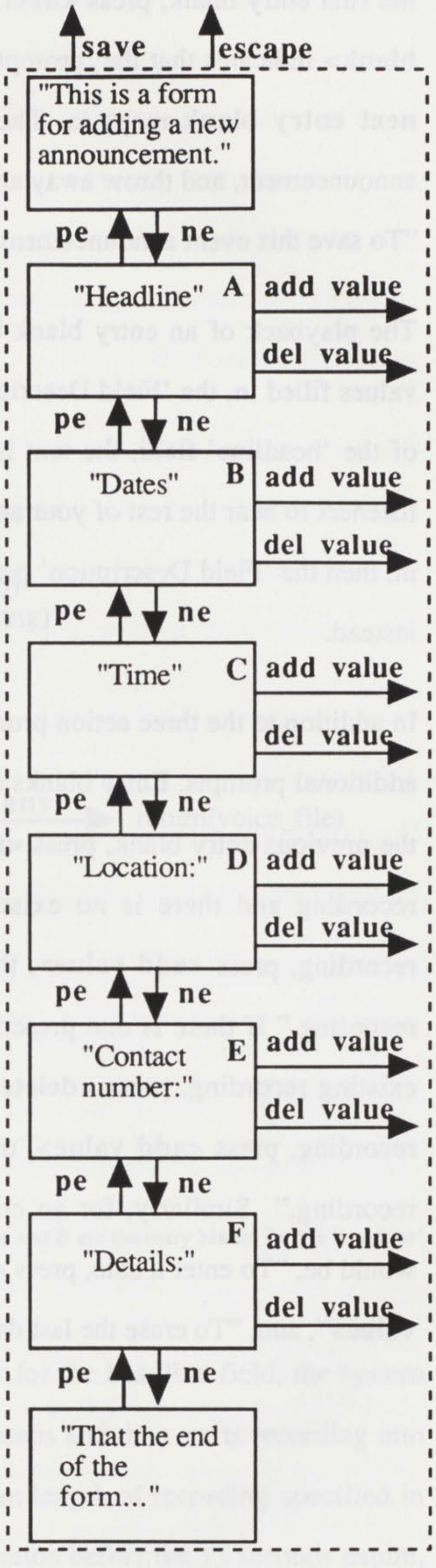


Figure B.12 A telephone form graph for a new event announcement.

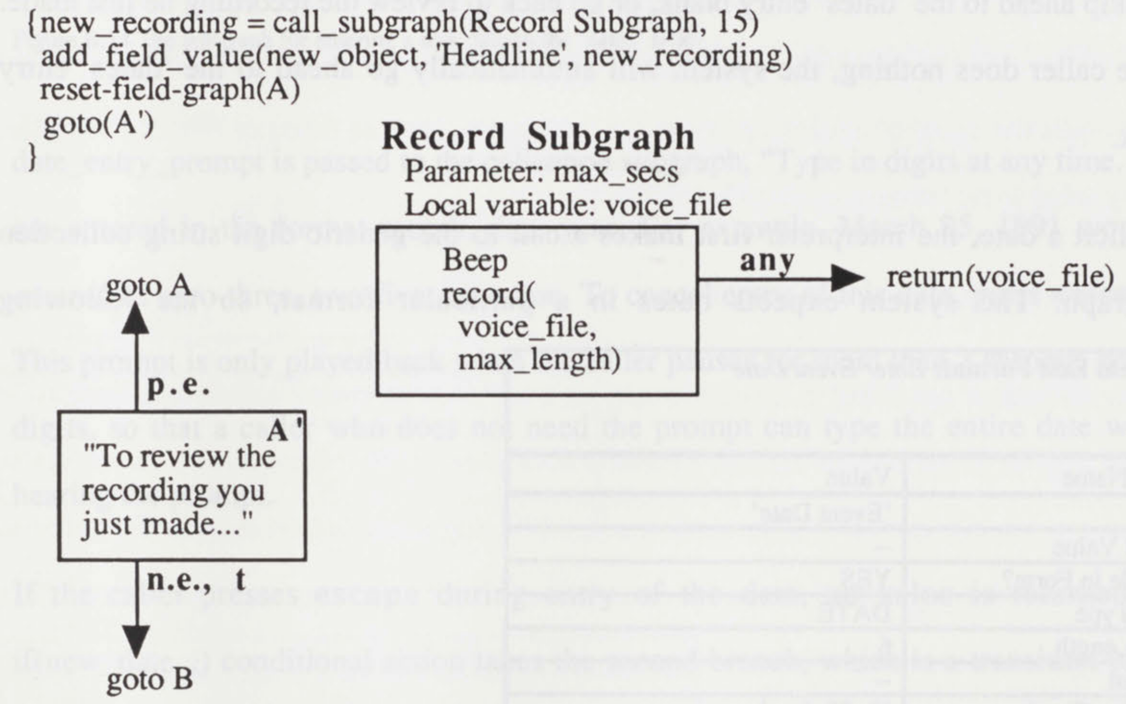
the first entry blank, press **<next entry blank>**." As before, the notation **<next entry blank>** indicates that the prompt actually ends with the word nine, or whatever button **next entry blank** maps to. The second help subnode states, "To cancel this event announcement, and throw away anything that you recorded, press **<escape>**." The last is, "To save this event announcement, press **<save>**."

The playback of an entry blank begins with the field name. Then, if the field has no values filled in, the 'Field Description' from the Field Edit Format is played. In the case of the 'headline' field, the text is, "Think of this like a newspaper headline, to entice listeners to hear the rest of your announcement." If the field has one or more values filled in, then the 'Field Description' prompt is omitted and the contents of the field are played instead.

In addition to the three action prompts for the header node, entry blanks can have several additional prompts. Entry blanks after the first one in the form include the prompt, "For the previous entry blank, press **<previous entry blank>**." If the entry blank expects a recording and there is no existing recording, one help subnode states, "To begin recording, press **<add value>**, then wait for the beep. Press any button when finished recording." If there is one present already, then two help prompts state, "To erase the existing recording, press **<delete value>**" and, "To append to the end of the existing recording, press **<add value>**, then wait for the beep. Press any button when finished recording." Similarly, for an entry blank that expects dates, the equivalent prompts would be, "To enter a date, press **<add value>**", "To enter an additional date, press **<add value>**", and, "To erase the last date entered, press **<delete value>**".

3.5.1 Data Entry Subgraphs

Field Edit Format: ***Record Headline***	
Field Name	Value
Field	'Headline'
Initial Value	--
Include in Form?	YES
Data Type	VOICE
Max Length	15
Picklist	--
Validity Check	--
Field Description	"Think of this like a newspaper headline, to entice listeners to hear the rest of your announcement."
Field Header Prompt	--



```
{new_recording = call_subgraph(Record Subgraph, 15)
 add_field_value(new_object, 'Headline', new_recording)
 reset-field-graph(A)
 goto(A')
}
```

Figure B.13 The data entry subgraph for recording the headline. Nodes A and B are the entry blanks for the 'headline' and 'dates' fields, as shown in Figure B.12.

When a caller initiates recording from the entry blank for the headline field, the system makes a call to the Record Subgraph. That subgraph beeps and then starts recording into a empty voice file, for up to 15 seconds, the maximum length of recording specified in the Field Edit Format above. If the caller presses any button before the 15 seconds expire, that also will terminate recording, as will a couple of seconds of silence. The recording is returned from the subgraph, and is added as a value to the 'Headline' field. Since the entry blank node was generated from the old contents of the 'Headline' field, its contents are now incorrect. The *reset-field-graph* action regenerates that node, using the rules described above to determine the node's contents and the editing actions that are available. For example, the erase value action may be prompted for now that there is a recording for the field.

Then the system transitions to graph node A'. A' can be viewed as a node between the entry blank for the headline and the entry blank for the dates (node B). From A', a caller can skip ahead to the 'dates' entry blank, or go back to review the recording he just made. If the caller does nothing, the system will automatically go ahead to the 'dates' entry blank.

To elicit a date, the interpreter first makes a call to the generic digit string collection subgraph. The system expects dates in a particular format, so the following

Field Edit Format: *Enter Event Date*	
Field Name	Value
Field	'Event Date'
Initial Value	--
Include in Form?	YES
Data Type	DATE
Max Length	6
Picklist	--
Validity Check	[0- 90 days]
Field Description	One or more typed-in dates for this event.
Field Header Prompt	--

Date Validity Check: *0- 90 days*	
Field Name	Value
Min Value Relative to Today	0
Min Value Absolute	--
Max Value Relative to Today	90
Max Value Absolute	--

```
{new_date = call_subgraph(Figure B.6, date_entry_prompt, --, 2,  5)
 if(new_date,
    if (process_validity_check(new_date, [0-90 days])
          {add_field_value(new_object, 'Dates', new_date)
           reset-field-graph(B)
             goto(B')
          }
         goto(B)
       )
   goto(B)
   )
}
```

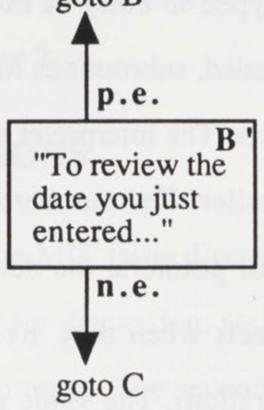


Figure B.14 The subgraph for entering a new date in the 'dates' field.

date_entry_prompt is passed to the collection subgraph, "Type in digits at any time. Dates are entered in the format month, day, year. For example, March 25, 1991 would be entered as zero-three, two-five, nine-one. To cancel entry of this date, press **<escape>**." This prompt is only played back when the caller pauses for more than 2 seconds between digits, so that a caller who does not need the prompt can type the entire date without hearing the prompt.

If the caller presses **escape** during entry of the date, no value is returned. The if(new_date...) conditional action takes the second branch, which is a transition back to the entry blank for entering dates. If a date is entered, then it checks the value against the Validity Check object, [0 - 90 days]. If the value is valid, then the validity check returns TRUE, the value is added to the 'Dates' field, and the interpreter transitions to node B'. If the value is not valid, then the system transitions to an error node (not shown) before returning from the validity check operation. The error node will state that the value is not a date and explain the format expected for dates, or it will state that the date is too early or too late, and say what the earliest or latest acceptable date is. If the caller presses a button to interrupt the error message node, the button press is requeued and the interpreter returns from the error graph. In this case, that would mean returning to the 'Dates' entry blank (B) and then executing the action associated with the button press.

Event announcements do not require typed-in entry of numbers, strings, or pointers to other objects. When such values are needed, subroutines for entry of these data types are implemented analogously to that for dates. The interpreter makes a call to a subgraph that collects a candidate value from the caller. For numbers and strings, the digit string collection subgraph would be used. For pointers, the data entry subgraph would be a menu that is configured to return objects when they are selected (a parameter of the Select Action object does this configuration). The value returned from the subgraph is checked for validity and then added to the appropriate field. This same scheme would also handle the input of additional primitive data types, such as times or monetary values, if they were to be integrated into HyperVoice.

3.5.2 Saving and Canceling Forms

Before saving or canceling a form, the system transitions to a confirmation subgraph, to ensure that these operations are not executed accidentally. The same button that initiates the save or cancel operation also confirms it, while any other button returns the caller to the telephone form graph. If the caller cancels the form, then the system just returns from the telephone form subgraph without adding the edited object to any lists. If the caller saves the form, then the save-form action is initiated, which looks in the Extension Format to determine the lists to which the new object should be added, and the locations in those lists. If the object is added to the current list, then the state-machine program for presenting that list will no longer be current. Hence, the system regenerates the graph for the current List Presentation, and positions the caller at the same location in the list from which he initiated the addition of the new event announcement.

C Sample Application Programs

This appendix lists HyperVoice program code for three of the most complex application prototypes presented in chapter 2.

1 ISSUE DISCUSSIONS

Two versions of U-TALK, the MIT issue discussion application, are shown below. The first presented a single topic for discussion, so that the first list of objects presented to callers was the top-level comments. The second version presented several topics in the top-level list. Only the system administrator could add new topics. Both version are presented below.

1.1 U-TALK I

1.1.1 Classes

Class Name	Fields
Comment	'Date Added' 'Headline' 'Contents' 'Responses'

1.1.2 Instances

Login: *UTALK-1*	
Field Name	Value
Register?	FALSE
Greeting	Hello, welcome to U-TALK, where you can make your voice heard.
Start Item	[Academic Honesty]
Special List Jumps	--
Ask for Last Date?	FALSE

The opinion forum accepts comments from anyone, so no registration procedure is required. At the beginning of each phone call, the system will play the greeting, then present the top-level list, [Academic Honesty].

List Presentation: *Academic Honesty*	
Field Name	Value
List	[Top-Level Comments List]
Format	[Top-Level Comments List Format]
Title	"Academic honesty at MIT."
Description	"Is cheating rampant? What should be done about it? Are professors making the rules clear? What about course bibles?"
Menu Prompt	--

As in the jokes application, no menu prompt is needed since this is the top-level menu.

List : *Top-Level Comments List*	
Field Name	Value
Contents	-

The top-level comments list is initially empty. Callers will add to it.

List Format: *Top-level Comments List Format*	
Field Name	Value
Sort Order	--
Filter	--
Item Formats	[Comment Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	POSITIONAL
List Actions	[Add Comment]
Name for Objects	Comment
Response List?	NO
Say Item Count?	YES

Comments are not filtered or sorted. Callers have to press a button to skip through hearing the top-level comments. A single selection button is provided for positional selection. As the [Comment Item Format] will specify, selection takes a caller to the response list for a comment.

Since this list is not treated as a response list, the header prompt is generated by concatenating the title and description prompts from the List Presentation object. This is what a caller will hear immediately after the greeting above. Then, the number of comments in the list is recited.

List Format: *Comments Response List Format*	
Field Name	Value
Sort Order	--
Filter	--
Item Formats	[Comment Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	POSITIONAL
List Actions	[Add Comment]
Name for Objects	Comment
Response List?	Comments in Response to [Headline]
Say Item Count?	YES

The difference between this and the previous List Format is in the 'Response List?' parameter. In this case, the list of comments are responses to another one. Hence, the header prompt at the beginning of playback of the list is a concatenation of a recording of the phrase "Comments in response to" and the headline of the comment it is in response to. This points another advantage of the headlines. While it would be cumbersome to play back the entire original comment, its headline serves as a good place holder to help callers keep track of what the current comments are responses to.

Item Format: *Comment Item Format*	
Field Name	Value
Object Class	[Comment]
Field Formats	['Headline' NO --] ['Contents' NO --] ['Date Added' YES --]
Select Action	[Goto Responses]

The 'headline', 'contents', and 'date added' fields of each comment are played back, in that order. Only the field name "Date added" is played back prior to playing the field's contents.

Select Action: *Goto Responses*	
Field Name	Value
Action Type	GOTO FIELD [Responses]
Action After Returning	STAY AT CURRENT ITEM
Relationship	Responses to

When a comment is selected, this determines that the system will present the contents of the field 'Responses'. That field should contain a link to a List Presentation object; the editing formats below will ensure that. After a caller is finished listening to the responses, she will be able to return to this list, and will then hear the current comment again, as specified by the value, STAY AT CURRENT ITEM.

List Action: *Add Comment*	
Field Name	Value
Privileges Required	ADDING
Button Location	ALL
Extension Format	[New Comment]

From anywhere in the list of comments, anyone with ADDING privileges can initiate addition of a new comment.

Extension Format: *New Comment*	
Field Name	Value
Add to Current List?	YES
Other Lists to Add to	--
Location to Add	BEGINNING
Class of Object to Add	[Comment]
Edit Format	[Edit New Comment]

If the caller saves the comment, it will be added at the beginning of the current list. This means that comments will be in reverse chronological order, which will help repeat callers to find new comments without having to skip through all of the ones they've already heard. On the other hand, it means that the list of comments do not flow coherently in response to each other, but that is what response lists are for. To respond to a comment, one adds to its response list, not to the list it is in.

Edit Format: *Edit New Comment*	
Field Name	Value
Field Formats	[Record Headline] [Record Contents] [Fill in Date Added] [Create Response List Presentation] (expand macro)

Field Edit Format: *Record Headline*	
Field Name	Value
Field	'Headline'
Initial Value	--
Include in Form?	YES
Data Type	VOICE
Max Length	15
Picklist	--
Validity Check	--
Field Description	--
Field Header Prompt	--

The headline recording is restricted to 15 seconds.

Field Edit Format: *Record Contents*	
Field Name	Value
Field	'Contents'
Initial Value	--
Include in Form?	YES
Data Type	VOICE
Max Length	300
Picklist	--
Validity Check	--
Field Description	--
Field Header Prompt	--

The contents field accepts a longer recording, of up to 300 seconds.

Field Edit Format: *Fill in Date Added*	
Field Name	Value
Field	'Date Added'
Initial Value	* TODAY
Include in Form?	NO
Data Type	DATE
Max Length	--
Picklist	--
Validity Check	--
Field Description	--
Field Header Prompt	--

The 'date added' field is filled in automatically, and is not included in the form, to prevent accidental editing of it.

Make Response List: *Create Response List Presentation*	
Field Name	Value
Field	'Responses'
Response List Format	[Comments Response List Format]

This macro does all the complicated work of automatically adding a new list of responses each time a new comment is added. It creates a new List Presentation object and a new, empty list to hold the responses. It sets the list format for the new List Presentation to be the [Comments Response List Format], specified above.

1.2 U-TALK II

The second version of this opinion forum functions nearly identically to the first, but allows discussion on several topics, with each topic presented as a question to which callers are invited to respond. Only the changed portions of the program are shown here.

Login: *UTALK-II*	
Field Name	Value
Register?	FALSE
Greeting	Hello, welcome to U-TALK, where you can make your voice heard.
Start Item	[Random Topics]
Special List Jumps	--
Ask for Last Date?	FALSE

List Presentation: *Academic Honesty*	
Field Name	Value
List	[Topics List]
Format	[Topics List Format]
Title	"Random topics."
Description	"Find a question that interests you, listen to other people's responses to it, then YOU TALK. Suggestions for new topics can be added in response to the last question. Have fun!
Menu Prompt	--

The 'Title' and 'Description' fields for the initial list have changed, to reflect the fact that the system now handles several topics at once.

List Format: *Topics List Format*	
Field Name	Value
Sort Order	--
Filter	--
Item Formats	[Comment Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	POSITIONAL
List Actions	[Add Topic]
Name for Objects	Question
Response List?	NO
Say Item Count?	YES

Topics are just comments, so the [Comment Item Format] still specifies how to present them. In this case, however, the prompts call them questions rather than comments. Thus, the navigation prompts will be of the form, "For the next question, press 9" instead of, "For the next comment, press 9."

List Action: *Add Topic*	
Field Name	Value
Privileges Required	EDITING
Button Location	ALL
Extension Format	[New Comment]

The only other change is that EDITING privileges are required to add a new topic. This gives the system administrator some control over the topics, which are the first things that new callers will hear. By seeding the topics list appropriately, the system administrator may influence the tone of discussions that will ensue, although that influence will be limited.

2 TEACHERS' CURRICULUM LINE

This program incorporates aspects of issue discussion, an event calendar, and questions and answers. The issue discussion component consists of two lists of comments, neither of which include response lists. The question and answer portions are attached to lesson plans. Unlike the question and answer application listed above, questions here are made public in addition to being added the expert's mailbox (in this case there is only one expert, the head teacher).

2.1 CLASSES

Class Name	Fields
Lesson Plan	'Lesson Number' 'Objectives' 'Materials' 'Dilemma' 'Alternate Forms' 'Projections' 'Related Research' 'Lab' 'Drill' 'Homework' 'Home Projects' 'Questions'
Comment	'Date Added' 'Added By' 'Headline' 'Contents'
Meeting Announcement	'Headline 'Dates' 'Details'
Question	'Date Added' 'Added By' 'Headline' 'Contents' 'Answers' 'Linked Lesson'
Answer	'Date Added' 'Added By' 'Headline' 'Contents'
Success Story	'Date Added' 'Added By' 'Headline' 'Contents'

2.2 INSTANCES

Login: *Curriculum Line*

Field Name	Value
Register?	TRUE
User Id Length	4
Users List	[All Teachers]
Password?	FALSE
Greeting	Hello, welcome to the Math-cubed curriculum line.
Start Item	[The Main Menu]
Special List Jumps	[Goto all questions]

Head Teacher's Access to All Questions

List Jump: *Goto All Events*	
Field Name	Value
To Jump To	[Head Teacher's Question List Presentation]
Code	9999
Privileges Required	EDITING

The head teacher, who will have editing privileges, can jump to the presentation of all of the questions. This is her private mailbox.

List Presentation: *Head Teacher's Question List Presentation*	
Field Name	Value
List	[All Questions]
Format	[Moderator's Question List Format]
Title	"All questions"
Description	--
Menu Prompt	--

List Format: *Moderator's Question List Format*	
Field Name	Value
Sort Order	['Date Added' DESCENDING]
Filter	--
Item Formats	[Moderator's Question Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	NONE
List Actions	--
Name for Objects	Question
Response List?	NO
Say Item Count?	YES

The questions are sorted with newest ones coming first in the list.

Item Format: *Moderator's Question Item Format*	
Field Name	Value
Object Class	[Event Announcement]
Field Formats	['Headline' NO --] ['Contents' YES --] ['Added by' YES --] ['Date Added' YES --] ['Linked Lesson' YES Play Lesson Plan Headline]]
Select Action	[Goto Responses]

Item Format: *Play Lesson Plan Headline*	
Field Name	Value
Object Class	[Lesson Plan]
Field Formats	['Lesson number' YES --] ['Objectives' YES --]
Select Action	--

The lesson number and objectives of the lesson plan that this question is about are played. Note that these are not played back to the other teachers, since they access questions by selecting a lesson plan.

Select Action: *Goto Responses*	
Field Name	Value
Action Type	GOTO FIELD [Responses]
Action After Returning	STAY AT CURRENT ITEM
Relationship	Responses to

When listening to a question, the head teacher can select it in order to go to the list of responses to it. The teacher can add an answer to the question, and then return to the current item in the list of all questions.

The Main Menu

List Presentation: *Head Teacher's Question List Presentation*	
Field Name	Value
List	[Main menu options]
Format	[Menu Format]
Title	"The main menu"
Description	Please choose an option
Menu Prompt	--

List: *Main menu options*	
Field Name	Value
Contents	[Kindergarten Lessons] [First Grade Lessons] [Second Grade Lessons] [Third Grade Lessons] [Success Stories] [Meeting Announcements] [Comments and Bug Reports]

The main menu provides seven options: four lists of lesson plans, one for each grade; a list of success stories; an events calendar; and a list of general comments.

List Format: *Menu Format*	
Field Name	Value
Sort Order	--
Filter	--
Item Formats	[Default Menu Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	POSITIONAL
List Actions	--
Name for Objects	Option
Response List?	NO
Say Item Count?	YES

Presenting a Lessons List

List Presentation: *Kindergarten Lessons*	
Field Name	Value
List	[K Lesson Objects]
Format	[Lessons List Format]
Title	"Kindergarten Lesson Plans"
Description	
Menu Prompt	"Kindergarten Lessons"

The List Presentations for the four grade levels are similar, so only one is presented. The prompt in the main menu is given in the last field. After the object is selected, the 'Title' prompt is played as part of the list header.

List Format: *Lessons List Format*

Field Name	Value
Sort Order	['Lesson Number' ASCENDING]
Filter	--
Item Formats	[Lesson Plan Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	NONE
List Actions	[New Lesson Plan]
Name for Objects	Lesson Plan
Response List?	NO
Say Item Count?	YES

Item Format: *Lesson Plan Item Format*

Field Name	Value
Object Class	[Event Announcement]
Field Formats	['Lesson Number' YES --] ['Objectives' YES --] ['Materials' YES --] ['Dilemma' YES --] ['Alternate Forms' YES --] ['Projections' YES --] ['Related Research' YES --] ['Lab' YES --] ['Drill' YES --] ['Homework' YES --] ['Home Projects' YES --]
Select Action	[Goto Questions]

Select Action: *Goto Questions*

Field Name	Value
Action Type	GOTO FIELD 'Questions'
Action After Returning	STAY AT CURRENT ITEM
Relationship	Questions about

When a teacher selects a lesson plan, the system goes to the questions associated with the lesson plan.

Adding New Lesson Plans

List Action: *Add Lesson Plan*	
Field Name	Value
Privileges Required	EDITING
Button Location	ALL
Extension Format	[New Lesson Plan]

The head teacher, who has EDITING privileges, can add a new lesson plan.

Extension Format: *New Lesson Plan*	
Field Name	Value
Add to Current List?	YES
Other Lists to Add to	--
Location to Add	END
Class of Object to Add	[Lesson Plan]
Edit Format	[Edit New Lesson Plan]

The new lesson plan will be added at the end of the current list.

Edit Format: *Edit New Lesson Plan*	
Field Name	Value
Field Formats	[Enter Lesson Number] [Record Objectives] [Record Materials] [Record Dilemma] [Record Alternate Forms] [Record Projections] [Record Related Research] [Record Lab] [Record Drill] [Record Homework] [Record Home Projects] [Create Questions List]

Most of the fields are filled in with recordings: the Field Edit Formats specifying them are omitted here. Only the first and last Field Edit Formats are shown here.

Field Edit Format: *Enter Lesson Number*	
Field Name	Value
Field	'Lesson Number'
Initial Value	--
Include in Form?	YES
Data Type	NUMBER
Max Length	3
Picklist	--
Validity Check	--
Field Description	--
Field Header Prompt	--

Up to a three-digit number will be accepted as a value for the 'Lesson Number' field.

Make Response List: *Make Questions List*	
Field Name	Value
Field	'Questions'
Response List Format	[Questions List Format]

The macro defined in Appendix A comes in handy for specifying that each new lesson plan should have a new list of responses to it. That response list should be presented using the [Questions List Format].

Presenting a Questions List

List Format: *Questions List Format*	
Field Name	Value
Sort Order	--
Filter	--
Item Formats	[Questions Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	POSITIONAL
List Actions	[Add Comment]
Name for Objects	Question
Response List?	Questions about Lesson 'Lesson Number'
Say Item Count?	YES

Item Format: *Questions Item Format*			
Field Name	Value		
Object Class	[Question]		
Field Formats	['Headline' ['Contents' ['Added By'	NO NO YES	--] --] --]
Select Action	[Goto Responses]		

The 'headline', 'contents', and 'added by' fields of each question are played back, in that order. Note that the 'Date added' and 'Linked lesson' fields are not played back, although those fields are played back when the head teacher accesses them, as specified in the [Moderator's Question Item Format] above. The [Goto Responses] Select Action object is reused here.

List Action: *Add Question*	
Field Name	Value
Privileges Required	ADDING
Button Location	ALL
Extension Format	[New Question]

From anywhere in the list of comments, anyone can initiate addition of a new question.

Extension Format: *New Question*	
Field Name	Value
Add to Current List?	YES
Other Lists to Add to	[All Questions]
Location to Add	BEGINNING
Class of Object to Add	[Question]
Edit Format	[Edit New Question]

New questions are added to the [All Questions] list, as well as to the current list.

Edit Format: *Edit New Question*	
Field Name	Value
Field Formats	[Fill in Date Added] [Fill in Author] [Record Headline] [Record Contents] [Create Answer List] [Fill in Lesson Plan]

Field Edit Format: *Fill in Author*	
Field Name	Value
Field	'Added By'
Initial Value	* CURRENT_USER
Include in Form?	NO
Data Type	LINK

Make Response List: *Create Answer List*	
Field Name	Value
Field	'Answers'
Response List Format	[Answers List Format]

The 'Answers' field will contain a link to a new List Presentation that uses the [Answers List Format].

Field Edit Format: *Fill in Lesson Plan*	
Field Name	Value
Field	'Linked Lesson'
Initial Value	* PARENT_OBJECT
Include in Form?	NO
Data Type	LINK

The 'Lesson' field is set to the parent object, which is the lesson plan that the current list of questions is responding to.

The [Answers List Format] is omitted from this program listing, since it is so similar to many of the List Formats from earlier programs. It allows anyone to add new answers, and those answers do not have response lists attached to them.

3 TASK TRACKING

Three object types are required for this application. Each person on the project is represented by a User object. The class User as presented in chapter 4 is specialized to include additional fields, including phone number, email and fax address. Tasks and Status Reports are two new object classes.

3.1 CLASSES

Class Name	Fields
Person	'Name' 'User Id' 'Password' 'Privileges' 'Start Object' 'Phone Number' 'Fax Number' 'Email address' 'US mail address'
Task	'Persons Responsible' 'Date due' 'Priority' 'Description'
Status Report	'Date added' 'Added by' 'Task' 'Status' 'Details'

There is one master List of status reports, [All status reports] and one master List of Tasks [All tasks]. Project members only accesses the information about tasks for which they are responsible, while project coordinators can access all of the information. In the remainder of this section, the specifications required for one project member are described, and then the specifications for a project coordinator.

3.2 PROJECT MEMBER INSTANCES

Consider a single project member, John Doe. After John logs in, the system presents him with a menu of four options. That menu includes hearing all tasks for which he is responsible, sorted either by date due or by priority, all status reports that he has recorded, or all status reports for tasks for which he is responsible.

Person: *John Doe*

Field Name	Value
Name	Recorded
User Id	9999
Password	0000
Privileges	ADDING
Start Object	[John Doe's Choices]
Phone Number	1234567
Fax Number	7654321
Email Address	Recorded
US mail address	Recorded

List Presentation: *John Doe's choices*

Field Name	Value
List	[John Doe's choice list]
Filter	--
Format	[Menu Format]
Title	"John Doe's options"
Description	
Menu Prompt	"John Doe"

List Format: *Menu Format*	
Field Name	Value
Sort Order	--
Item Formats	[Default Menu Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	POSITIONAL
List Actions	--
Name for Objects	Option
Response List?	NO
Say Item Count?	NO

List: *John Doe's choice list*	
Field Name	Value
Contents	[JD tasks by date due] [JD tasks by priority] [JD's status reports] [JD's tasks' status reports]

List Presentation: *JD tasks by date due*	
Field Name	Value
List	[All tasks]
Format	[Sort JD tasks by date first]
Title	"John Doe's tasks sorted by date"
Description	
Menu Prompt	"Tasks sorted by date"

The first option: all of the tasks for John Doe is responsible, sorted by date due, with priority used as a tie breaker for tasks due on the same date.

List Format: *Sort JD's tasks by date first*	
Field Name	Value
Sort Order	['Date Due' ASCENDING] ['Priority' DESCENDING]
Filter	[JD Responsible]
Item Formats	[Default Task Format]
Advancement Mechanism	DEFAULT
Selection Mechanism	DEFAULT
List Actions	[Add Task]
Name for Objects	Option
Response List?	NO
Say Item Count?	NO

Filter: *JD responsible*

Field Name	Value
Or Filters	--
And Filters	--
Class required	[Task]
Field Filters	'Persons Responsible' [=JD]

Object Validity Check: *=JD*

Field Name	Value
Field Cannot be Empty?	--
Equals	[John Doe]
Member of	--
Filter	--

The selected tasks are those that pass the filter: only tasks which have [John Doe] linked into the 'Persons Responsible' field are selected.

Item Format: *Default Task Format*

Field Name	Value
Object Class	[Task]
Field Formats	['Description' NO --] ['Date Due' YES --] ['Priority' YES --] ['People responsible' YES --]
Select Action	[Go to Status Reports]

Four fields of each selected task are played back.

Select Action: *Go to Status Reports*

Field Name	Value
Action Type	GOTO FIELD 'Status Reports'
Action After Returning	STAY AT CURRENT ITEM
Relationship	Status reports about

John Doe can select any of the tasks in order to hear all of the status reports associated with the task.

List Action: *Add Task*

Field Name	Value
Privileges Required	ADDING
Button Location	ALL
Extension Format	[New Task]

Extension Format: *New Comment*

Field Name	Value
Add to Current List?	YES
Other Lists to Add to	--
Location to Add	BEGINNING
Class of Object to Add	[Task]
Edit Format	[Edit New Task]

New tasks are added to the list of all tasks, which is the current list.

Edit Format: *Edit New Comment*

Field Name	Value
Field Formats	[Record Headline] [Record Description] [Fill in Author] [Enter Date Due] [Enter Priority] [Enter Persons Responsible] [Create Status Reports Response List]

The first two Field Edit Formats are omitted, since they are similar to previous specifications of fields that accept recordings. Likewise, the third Field Edit Format is similar to previous specifications that set a field to be a link to the current User object but do exclude the field from the telephone form so that the field is not editable by the caller.

Field Edit Format: *Enter Date Due*

Field Name	Value
Field	'Date Due
Initial Value	--
Include in Form?	YES
Data Type	DATE
Max Length	6
Picklist	--
Validity Check	--
Field Description	The date by which this task needs to be completed.
Field Header Prompt	--

Any valid date will be accepted for the 'Date Due' field, even dates that have already passed!

Field Edit Format: *Enter Priority*	
Field Name	Value
Field	'Priority'
Initial Value	--
Include in Form?	YES
Data Type	NUMBER
Max Length	1
Picklist	--
Validity Check	--
Field Description	The priority of the task on a scale of one to nine.
Field Header Prompt	--

The caller will enter a single digit for the priority.

Field Edit Format: *Enter Persons Responsible*	
Field Name	Value
Field	'Persons Responsible'
Initial Value	--
Include in Form?	YES
Data Type	LINK
Max Length	--
Picklist	[Pick a user]
Validity Check	--
Field Description	Links to one or more persons responsible for this task
Field Header Prompt	--

The caller can select user objects for the 'Persons Responsible' field, using the [Pick a user] selection menu below.

List Presentation: *Pick a user*	
Field Name	Value
List	[All Users]
Format	[Users Picklist Format]
Title	"People"
Description	
Menu Prompt	--

List Format: *Users Picklist Format*

Field Name	Value
Sort Order	--
Filter	--
Item Formats	[Say User Name; return]
Advancement Mechanism	SKIP
Selection Mechanism	POSITIONAL
List Actions	--
Name for Objects	Person
Response List?	NO
Say Item Count?	YES

Item Format: *Say User Name; return*

Field Name	Value
Object Class	[User]
Field Formats	['Name' NO --]
Select Action	[Return]

Select Action: *Return*

Field Name	Value
Action Type	RETURN ME
Action After Returning	--

The menu options are the names of the users. When a selection is made from the menu, the User object is returned as a value, which is then linked into the 'Persons Responsible' field of the new task object.

Make Response List: *Create Status Reports Response List*

Field Name	Value
Field	'Status Reports'
Response List Format	[Status Reports List Format]

This macro does all the complicated work of automatically adding a new list of status reports each time a new task is added. It creates a new List Presentation object and links it into the 'Status Reports' field. It sets the list format for the new List Presentation to be the [Status Reports List Format], specified below, and creates a new list to hold the status reports.

List Format: *Status Reports List Format*	
Field Name	Value
Sort Order	--
Filter	--
Item Formats	[Default Status Report Item Format]
Advancement Mechanism	SKIP
Selection Mechanism	NONE
List Actions	[Add Status Report]
Name for Objects	Status Report
Response List?	Status reports about 'Headline'
Say Item Count?	YES

Since this is a response list, the header prompt for a list of status reports states that these are status reports about a particular task, which is identified by playing the 'Headline' field of the Task object. The [Default Status Report Item Format] just plays the contents of the 'Status', 'Date Added', and 'Added By' fields: the program listing is omitted here. The [Add Status Report] List Action is also similar to previous List Action specifications and is omitted here. It allows the caller to add a new status report to both the current list and to the list of [All Status Reports]. The 'Task' field is filled in automatically using the * PARENT_OBJECT initial value, which sets the field to be a link to the object that the current list is responding to.

List Presentation: *JD tasks by priority*	
Field Name	Value
List	[All tasks]
Format	[Sort JD tasks by priority]
Title	"John Doe's tasks sorted by priority"
Description	
Menu Prompt	"Tasks sorted by priority"

The second option available to John Doe: all of the tasks for which he is responsible, sorted by priority, with date due used as a tie breaker for tasks with the same priority.

List Format: *Sort JD's tasks by priority*	
Field Name	Value
Sort Order	['Priority' DESCENDING] ['Date Due' ASCENDING]
Filter	[JD Responsible]
Item Formats	[Default Task Format]
Advancement Mechanism	DEFAULT
Selection Mechanism	DEFAULT
List Actions	[Add Task]
Name for Objects	Option
Response List?	NO
Say Item Count?	NO

This is nearly identical to the List Format for the first option, except that the two sort orderings are reversed, to reflect sorting on 'Priority' first rather than 'Date Due'.

List Presentation: *JD status reports*	
Field Name	Value
List	[All Status Reports]
Format	[All recorded by JD]
Title	"Status reports recorded by John Doe"
Description	
Menu Prompt	"John Doe's recorded status reports"

The third option available to John Doe: all of the status reports recorded by him.

List Format: *All recorded by JD*	
Field Name	Value
Sort Order	--
Filter	[JD recorded]
Item Formats	[Default Status Report Format]
Advancement Mechanism	SKIP
Selection Mechanism	NONE
List Actions	--
Name for Objects	Status Report
Response List?	NO
Say Item Count?	NO

No provision is made for adding new status reports from here, because the system would not know which task the status report is associated with.

Filter: *JD recorded*	
Field Name	Value
Or Filters	--
And Filters	--
Class required	[Status Report]
Field Filters	'Added by' [=JD]

This filter picks out only those status reports that were recorded by John Doe. The [=JD] Object Validity check was already shown above, as part of the filter for selecting only tasks for John Doe was responsible.

List Presentation: *JD status reports*	
Field Name	Value
List	[All Status Reports]
Format	[All recorded by JD]
Title	"Status reports recorded by John Doe"
Description	
Menu Prompt	"John Doe's recorded status reports"

The fourth and final option available to John Doe: all of the status reports about projects for which he is responsible. Note that these status reports might have been recorded by anyone.

List Format: *All reports JD responsible for*	
Field Name	Value
Sort Order	--
Filter	[JD responsible for task]
Item Formats	[Default Status Report Format]
Advancement Mechanism	SKIP
Selection Mechanism	NONE
List Actions	--
Name for Objects	Status Report
Response List?	NO
Say Item Count?	NO

Filter: *JD responsible for task*	
Field Name	Value
Or Filters	--
And Filters	--
Class required	[Status Report]
Field Filters	'Task' [JD Responsible]

This filter picks out only status reports about tasks for which John Doe is responsible. It embeds the [JD Responsible] Filter specified earlier for selecting tasks that JD is responsible for. This illustrates one advantage of the Filter constructs of the HyperVoice language. In a relational query language such as SQL [Date 1986] this operation would first require a join of the two tables representing tasks and status reports, based on a task-id field, then a filter operation on the joined table, then a projection back to select only the status report fields. In the HyperVoice program above, the filter on the status reports simply specifies that the object linked into the 'Task' field must satisfy another filter.

3.3 PROJECT LEADER INSTANCES

The project leader may choose to access the tasks and status reports by person responsible or may choose to access them all together. The objects below specify the initial menu accessed by the project leader.

Person: *Project Leader*	
Field Name	Value
Name	Recorded
User Id	8888
Password	0000
Privileges	EDITING
Start Object	[Project Leader Choices]
Phone Number	1234567
Fax Number	7654321
Email Address	Recorded
US mail address	Recorded

List Presentation: *Project Leader Choices*	
Field Name	Value
List	[PL's choice list]
Format	[Menu Format]
Title	"Project Leader's options"
Description	
Menu Prompt	"Project Leader"

List: *PL's choice list*	
Field Name	Value
Contents	[All tasks by date due] [All tasks by priority] [All status reports] [Tasks and Status Reports by Person]

The Project Leader has four options. The first three are similar to the first three options seen by project members. The difference is that no filters are specified in the List Format objects, so that all tasks or status reports are selected rather than a subset being selected. Due to their similarity to objects above, the objects for the first three menu choices are omitted.

List Presentation: Tasks and Status Reports by Person	
Field Name	Value
List	[Users List]
Format	[Choose Person for Access]
Title	"Choose a person"
Description	
Menu Prompt	"Access tasks and status reports for a particular person"

The project leader can also access information relevant to any one member of the project, by selecting that person from a menu of all the people.

List Format: Choose Person for Access	
Field Name	Value
Sort Order	--
Filter	--
Item Formats	[Say User Name; navigate]
Advancement Mechanism	SKIP
Selection Mechanism	POSITIONAL
List Actions	--
Name for Objects	Option
Response List?	NO
Say Item Count?	YES

Item Format: *Say User Name; navigate*	
Field Name	Value
Object Class	[User]
Field Formats	['Name' NO --]
Select Action	[Goto Start Item]

Each menu option is just the name of a person, which comes from the 'Name' field of the User object.

Select Action: *Goto Start Item*	
Field Name	Value
Action Type	GOTO FIELD 'Start Item'
Action After Returning	STAY AT CURRENT ITEM

If a User object is selected, the system presents the List Presentation in the User object's 'Start Item' field. Note that this is the same thing that the project member hears after first logging in. Thus, when the project leader makes a selection from this menu, the project leader has all the same options that a project member has, as described above.

This program illustrates some of the limitations of the HyperVoice language. First, all of the presentation formats are pre-specified, which does not allow for ad-hoc queries, such as finding all of the tasks for which one of two people are responsible. Second, the language does not have an abstraction mechanism for creating new primitives. The program listing only shows the List Presentations and List Formats for one of the project members, John Doe. Similar ones would have to be created for each additional project member. Instead, there should be a way to create a new macro or a new primitive with a few variables for the information that differs among the Presentation Formats and List Formats for the different project members. These limitations and extensions were discussed in the future research section of chapter 4.

D MENU EXPERIMENT DETAILS

In the second menu experiment, subjects made selections from two different menu trees. This appendix contains enough of the details of the experiment so that, together with the material from Chapter 5, other researchers could replicate it. The appendix first presents the text of the prompts in both trees. Then, it gives the prompts that explained each of the twelve selection tasks that subjects performed. Finally, a 12x12 Latin Square determines the order in which subjects performed the set of tasks.

1 THE MENU TREES

Below are the text of the prompts for the options in the two applications. Submenus are indented. These prompts were the same with all of the menu selection styles. Thus, in the top-level menu, with the standard menu style, the prompts for selecting sports scores is: "Sports scores and highlights, press 2" With positional selection, the "press 2" is omitted and the caller is instead prompted to press 1 for the current selection. If the subject then selects the sports option, the system plays the menu header, "The sports report. Where Budweiser brings you all the action." The subject can then make a selection of pro sports, local college, NCAA, or olympic highlights.

1.1 NEWS AND INFORMATION APPLICATION

Welcome to PhoneLine. Your only source for up to the minute news and information. Brought to you by GTE, where the power is on.

1. News reports from around the country and around the globe.

The News Desk. Brought to you by Ford motor company, where quality is job one.

1. State news. Keeping you up to date on Massachusetts.
2. National News. Brought to you by Turner Broadcasting Company.
3. World headlines. Late breaking international news.
4. The business report, provided by the Wall Street Journal.

2. Sports scores and highlights.

The sports report. Where Budweiser brings you all the action.

1. The Pro Sports reports, sponsored by Budweiser.

1. The NFL this week. Off season trades and news.
2. The NBA Report. Countdown to the finals.
3. The major league baseball report. Sponsored by Budweiser. This Bud's for you.
4. NHL action. Game and strike news.

2. Local college action, covering BU, BC, Harvard and Northeastern.
The local college sports scene, brought to you by WFNX, the cutting edge of rock
1. Boston College, home of the Eagles
2. Boston University, home of the Terriers
3. Harvard University. Home of the Crimson.
4. Northeastern University, home of the Wildcats

3. The NCAA report, with college scores from around the nation.
Rundown on the NCAA, brought to you by Barnes and Noble Bookstores.
1. Basketball scores from the NCAA-sponsored tournaments.
2. Football news from around the NCAA
3. Hockey scores and news from around the NCAA

4. Olympic highlights. The latest on Albertville.
Selected Olympic Results.

1. Figure skating results from Albertville, France.
2. Olympic hockey results.
3. Alpine and Nordic skiing results from Albertville, France.

3. Horoscopes. By noted astrologer Jeane Dixon.
Horoscopes. Don't make a move without consulting the stars
1. Aquarius, January 20th to February 18th.
2. Pisces, February 19th to March 20th.
3. Aries, March 21st to April 19th.
4. Taurus. April 20th to May 20th.
5. Gemini, May 21st to June 21st.
6. Cancer, June 22nd to July 22nd.
7. Leo, July 23rd to August 22nd.
8. Virgo, August 23rd to September 22nd.
9. Libra. September 23rd to October 23rd.
10. Scorpio, October 24th to November21st.
11. Sagittarius, November 22nd to December 21st.
12. Capricorn, December 22nd to January 19th

4. Entertainment news and happenings around town.
The entertainment report. Where the Boston Phoenix helps you make your leisure time count.
1. Hot videotape releases. The top renting videos in your area.
Videotape Hits. Supplied by Blockbuster videos, with a store near you.
1. The Rocketeer
2. Regarding Henry
3. Return of Blue Lagoon
4. Deadlock
5. Miles From Nowhere
6. Over Her Dead Body

2. Movie reviews by Gene Aubrey of the New York Times.
Now playing at your local theater.
1. Shining Through
2. Fried Green Tomatoes
3. Grand Canyon
4. Freejack

5. Final Analysis
6. Naked Lunch

3. Television Tonight. Highlights of this evening's programs.

Tonight's TV Highlights, brought to you by Heritage Cable TV.

1. On ABC this evening
2. On CBS tonight.
3. On NBC this evening
4. Fox network offerings tonight
5. Highlights of the cable networks this evening

4. Music: what's selling and what's playing around town.

The music scene is brought to you by the Phoenix, Boston's #1 weekly newspaper.

1. Billboard's top ten singles. Brought to you by Strawberries records in Framingham.
2. The top 10 albums according to Billboard magazine
3. Rock Talk. MTV VeeJay Martha Quinn brings you the scoop.
4. Concerts in Boston and MetroWest.

1.2 WEATHER APPLICATION

Welcome to WeatherLine. Forecasts for over 50 U.S. and international cities. Brought to you by Thomas Cook Travel, where we bring the world to you.

1. Asian and Pacific Rim cities

Choose the Asian or Pacific Rim city you will be traveling to:

1. Beijing. Mainland China's premier city.
2. Tokyo, Japan. The gateway to the orient.
3. Bangkok. Capital of Thailand.
4. Seoul. Capital of South Korea.

2. Western European Cities and regions.

Western European weather: select the region you are traveling to.

1. German and Austrian cities.

Germany and Austria, served by Lufthansa Air.

1. Berlin, and Eastern Germany.
2. Bonn, Germany, and the northern Rhine valley.
3. Munich, Germany and the German Alps.
4. Vienna and the Austrian forecast.

2. French and Belgian cities.

France and Belgium, served by Air France.

1. Bordeaux, France and the wine country.
2. Brussels, capital of Belgium.
3. Paris, France. City of lights.
4. Marseilles, France and the Mediterranean coast.

3. Spanish and Portuguese Cities.

Spain and Portugal, served by Ibena Airways.

1. Barcelona, Spain. Home of the 1992 summer olympics.
2. Lisbon, Portugal's capital.
3. Madrid, and Central Spain.

4. Italian and Grecian cities

Italy and Greece, served by Alitalia Airways.

1. Athens, Greece and the Grecian islands.
2. Milan, Italy and the Italian Alps.
3. Rome, Italy and the Vatican City.

3. Canadian city and regional forecasts.

Provincial capitals and major cities of Canada.
1. Calgary and the Canadian Rockies.
2. Edmonton, and the Central Alberta forecast..
3. Halifax, Nova Scotia, and Prince Edward Island.
4. Montreal. Old world charm right next door.
5. St. John and New Brunswick province
6. Ottawa, Canada's capital city.
7. Southern Newfoundland, including the provincial capital, St. John's.
8. Saskatoon, and the Canadian plains.
9. Toronto, and Eastern Ontario.
10. Vancouver, and southern British Columbia.
11. Yukon and the Northwest territories.
12. Winnipeg, and southern Manitoba.

4. Major United States cities by region.
United States weather: select the region you are traveling to.
1. The West and Southwest, including all pacific coast and rocky mountain states.
Major cities in the western and southwestern states.
1. Albuquerque, New Mexico. Sitting on top of the continental divide.
2. Los Angeles, and Southern California.
3. Phoenix, and southern Arizona.
4. Portland, and western Oregon.
5. San Francisco, California, and the Silicon Valley.
6. Seattle, Washington and Puget Sound.

2. The midwest, including all Plains, Great lakes, and Ohio River Valley states.
Major cities in the midwestern states.
1. Chicago, Illinois and the lake Michigan shore.
2. Cincinnati, and southwestern Ohio
3. Des Moines, Iowa and the agricultural heartland.
4. Detroit, Michigan and the Great Lakes.
5. Minneapolis, Minnesota and the Northern Plains region.
6. Saint Louis and eastern Missouri.

3. The northeast region, New England and coastal states from Maine to Pennsylvania.
Major cities of the northeastern United States.
1. Albany, and central New York.
2. Boston, Massachusetts and the Cape.
3. New York City and the greater metropolitan area.
4. Philadelphia, Pennsylvania and central New Jersey.
5. Portland, Maine, and the New England ski report.

4. The Southern Region, including all Mid-Atlantic, Gulf coast, and South Central states.
Major cities in southern and middle Atlantic states.
1. Atlanta, and the north Georgia area.
2. Miami, and the South Florida report.
3. New Orleans, Louisiana, and the Mississippi Delta.
4. Washington, DC, the nation's capital.

2 THE TASKS

Each subjected completed twelve selection tasks from each menu tree, but different subjects completed the tasks in different orders. Since the two menu trees have the same structure, the set of tasks can be determined by paths through the tree. For example, the path 1-3 means to select the first item from the top-level menu, the third item from the second menu. In the news application, this picks out the path, "News; Word headlines." In the weather application, this picks out the path, "Asian cities; Bangkok." The twelve paths are lettered A through L and listed below with the tasks presented to subjects.

A. 1-2

You heard that President Bush announced a new tax proposal last night. You would like to see what PhoneLine has to say about it.

Check the weather in Tokyo.

B. 2-1-3

You heard that the Boston Red Sox and the New York Yankees just made a big trade. Where would you look for it?

Check the weather in Munich, Germany.

C. 2-2-3

Did Harvard win their hockey game last night against Yale?

Check the weather in Paris. (Not Texas, France!)

D..2-3-1

The NCAA basketball tournament is about to begin, and you want to see if University of Kentucky got a bid this year.

Check the weather in Barcelona

E. 2-4-3

You want to know who won the women's downhill at the Olympics. Go ahead.

What is the weather report for Rome, Italy?

F. 3-4

You are a Taurus. What is you horoscope?

What is the weather forecast for Montreal?

G. 3-9

Your friend is a Libra. What is her horoscope?

What is the weather forecast for Toronto?

H. 4-1-2

You are planning to rent a videotape tonight, and you want to here the review for "Regarding Henry". Find it on the service.

What is the weather forecast for Los Angeles, CA?

I. 4-2-1

You think Melanie Griffith is in the new movie, "Shining Through", but you want to check the review to make sure before heading out to the theater. Do it.

Check the weather in Chicago, IL.

J. 4-2-5

Find the review for the new movie, "Final Analysis", now playing at a theater near you.

You want to know how much snow is predicted for Minneapolis, MN. Go ahead.

K. 4-3-2

What shows are on CBS TV tonight?

What is the five day forecast for Boston, MA?

L. 4-4-1

You are interested in knowing which songs are at the top of the charts. Find the list of singles.

What is the weather forecast for Atlanta, GA?

3 TASK ORDERING

A 12x12 matrix used to order tasks for the twelve subjects. The letters A through L represent the twelve possible tasks. Each row shows the order of tasks for one subject.

A	B	C	D	E	F	G	H	I	J	K	L
B	I	A	G	C	H	E	J	F	L	D	K
C	A	E	B	F	D	I	G	K	H	L	J
D	G	B	H	A	J	C	L	E	K	F	I
E	C	F	A	D	B	K	I	L	G	J	H
F	H	D	J	B	L	A	K	C	I	E	G
G	E	I	C	K	A	L	B	J	D	H	F
H	J	G	L	I	K	B	D	A	F	C	E
I	F	K	E	L	C	J	A	H	B	G	D
J	L	H	K	G	I	D	F	B	E	A	C
K	D	L	F	J	E	H	C	G	A	I	B
L	K	J	I	H	G	F	E	D	C	B	A

This matrix satisfies the Latin Square properties:

1) Each subject (row) performs every task exactly once

2) In every trial position (column) every task is executed by some subject.

3) For every pair of tasks, each one appears directly to the right of the other exactly once in the matrix.

This assignment of tasks to subjects has only one drawback. The first subject gets tasks A through H in order. For ease of presentation, those tasks progress from selections early in a menu to selections later in the menu. This might potentially give the first subject quite an advantage. To remedy this, we relabeled tasks A through H in a random order, thus preserving all three desirable Latin Square properties but eliminating the possibility that the first subject would gain an advantage from knowing that the tasks progress through the menu in an orderly way.

REFERENCES

Ackerman, M. S. and Malone, T. W. Answer Garden: A Tool for Growing Organizational Memory. In *Proceedings of COIS '90 Conference on Office Information Systems*. (Cambridge, MA, 1990). ACM, pp. 31-39.

Akscyn, R. M., McCracken, D. L. and Yoder, E. A. KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. *Communications of the ACM*. 31, 7 (1988), pp. 820-835.

Arons, B. Authoring and Transcription Tools for Speech-Based Hypermedia Systems. In *Proceedings of the Conference of the American Voice Input/Output Society*. (1991a), pp. 15-20.

Arons, B. Hyperspeech: Navigating in Speech-Only Hypermedia. In *Proceedings of Hypertext*. (1991b). ACM, pp.133-146

Arons, B. Techniques and Applications of Time-Compressed Speech. In *Proceedings of AVIOS '92: Conference of the American Voice Input/Output Society*. (Minneapolis, MN, 1992).

Arthur, W. B. Competing Technologies: An Overview. In *Technical Change and Economic Theory*. G. Dosi, Ed. Columbia University Press, New York, NY, 1987.

Beard, D., Palanlappan, M., Humm, A., Banks, D., Nair, A. and Shan, Y.-P. A Visual Calendar for Scheduling Group Meetings. In *Proceedings of CSCW '90: Conference on Computer-Supported Cooperative Work*. (Los Angeles, CA, 1990). ACM, pp. 279-290.

Bollobas, B. *Graph Theory: an Introductory Course*. Springer Verlag, New York, 1979.

Cahn, J. E. The Generation of Affect in Synthesized Speech. *Journal of the American Voice Input/Output Society*. July (1990).

Callahan, J., Hopkins, D., Weiser, M. and Shneiderman, B. An Empirical Comparison of Pie vs. Linear Menus. In *Proceedings of CHI '88 Conference on Human Factors in Computing Systems* (Washington, DC, 1988). ACM, pp. 95-100.

Carasik, R. P. and Grantham, C. E. A Case Study of CSCW in a Dispersed Organization. In *Proceedings of CHI '88 Conference on Human Factors in Computing Systems*. (Washington, D.C., 1988). ACM, pp. 61-66.

Chaiklin, S. and Schrum, L. Community-Based Telecommunications. In *Third Guelph Symposium on Computer Mediated Communication*. (Guelph, Ontario, Canada, 1990).

Chalfonte, B. L., Fish, R. S. and Kraut, R. E. Expressive Richness: A Comparison of Speech and Text as Media for Revision. In *Proceedings of CHI '91 Conference* *on Human Factors in Computing Systems*. (Seattle, WA, 1991). ACM, pp. 21-26.

Collier, G. Thoth-II: Hypertext with Explicit Semantics. In *Proceedings of Hypertext '87*. (Chapel Hill, NC, 1987). ACM, pp. 269-287.

Conklin, J. Hypertext: An Introduction and Survey. *IEEE Computer*. 20, 9 (1987), pp. 17-41.

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

Creecy, R. H., Masand, B. M., Smith, S. J. and Waltz, D. L. Trading MIPS and Memory for Knowledge Engineering. *Communications of the ACM*. 35, 8 (1992), pp. 48-64.

Date, C. J. *An Introduction to Database Systems*. Addison-Wesley, Reading, MA, 1986.

de Baar, D. J. M. J., Foley, J. D. and Mullet, K. E. Coupling Application Design and User Interface Design. In *Proceedings of CHI '92 Conference on Human Factors in Computing Systems*. (Monterey, CA, 1992). ACM, pp. 259-266.

Degen, L., Mander, R. and Salomon, G. Working with Audio: Integrating Personal Tape Recorders and Desktop Computers. In *Proceedings of CHI '92 Conference on Human Factors in Computing Systems*. (Monterey, CA, 1992). ACM, pp. 413-418.

Detweiler, M., Schumacher, R., Jr. and Gattuso, N., Jr. Alphabetic input on a Telephone Keypad. In *Proceedings of the Human Factors Society-- 34th annual meeting*. (Santa Monica, CA, 1990).

diSessa, A. A. and Abelson, H. Boxer: A Reconstructable Computational Medium. *Communications of the ACM*. 29, 9 (1986), pp. 859-868.

Dreyfus, H. L. *Being-in-the-world: A Commentary on Heidegger's Being and Time, Division I*. MIT Press, Cambridge, MA, 1991.

Dumais, S. T., Furnas, G. W., Landauer, T. K., Deerwester, S. and Harshman, R. Using Latent Semantic Analysis to Improve Access to Textual Information. In *Proceedings of CHI '88 Conference on Human Factors in Computing Systems*. (Washington, DC, 1988). ACM, pp. 281-288.

Dumais, S. T. and Landauer, T. K. Using Examples to Describe Categories. In *Proceedings of CHI '83 Conference on Human Factors in Computing Systems*. (1983). ACM, pp. 112-115.

Dumais, S. T. and Landauer, T. K. Describing Categories of Objects for Menu Retrieval Systems. Behavior Research Methods, Instruments, and Computers 16, 2 (1984), pp. 242-248.

Engelbeck, G. and Roberts, T. The Effects of Several Voice-Menu Characteristics on Menu Selection Performance. Technical Report ST0401, US West Advanced Technologies, 1990.

Fast, L. and Ballantine, R. Dialing a name: Alphabetic Entry Through a Telephone Keypad. *SIGCHI Bulletin*. 20, 2 (1988), pp. 34.

Feiner, S. Seeing the Forest for the Trees: Hierarchical Display of Hypertext Structures. In *Proceedings of COIS '88 Conference on Office Information Systems*. (Palo Alto, CA, 1988). ACM, pp. 205-212.

Fichman, R. G. and Kemerer, C. F. Adoption of Software Engineering Process Innovations: The Case of Object-Orientation. WP-242, MIT Center for Information Systems Research, 1992.

Foss, C. Effective Browsing in Hypertext Systems. In *Proceedings of RIAO '88*. (Cambridge, MA, 1988). ACM, New York, pp. 16-23.

Furnas, G. W. Experience With an Adaptive Indexing Scheme. In *Proceedings of CHI '85 Conference on Human Factors in Computing Systems*. (San Francisco, 1985). ACM, pp. 131-135.

Furnas, G. W. Generalized Fisheye Views. In *Proceedings of CHI '86 Conference on Human Factors in Computing Systems*. (Boston, 1986). ACM, pp. 16-23.

Furuta, R. and Stotts, P. D. Programmable Browsing Semantics in Trellis. In *Proceedings of Hypertext '89*. (Pittsburgh, PA, 1989). ACM, pp. 27-42.

Garzotto, F., Paolini, P., Schwabe, D. and Bernstein, M. Tools for Designing Hyperdocuments. In *Hypertext/Hypermedia Handbook*. E. Berk and J. Devlin, Ed. McGraw-Hill, New York, 1991, pp. 179-207.

Gaver, W. W. Auditory Icons: Using Sound in Computer Interfaces. *Human-Computer Interaction*. 2, 2 (1986), pp. 167-177.

Goodman, D. *The Complete HyperCard 2.0 Handbook*. Bantam, New York, 1990.

Gould, J. D. and Boies, S. J. Human Factors Challenges in Creating a Principal Support Office System-- The Speech Filing System Approach. *ACM Transactions on Office Information Systems*. 1, 4 (1983), pp. 273-298.

Gould, J. D., Boies, S. J., Levy, S., Richards, J. T. and Schoonard, J. The 1984 Olympic Message System: A Test of Behavioural Principles of System Design. *Communications of the ACM*. 30, 9 (1987), pp. 758-769.

Green, M. A Survey of Three Dialog Models. *ACM Transactions on Graphics*. 5, 3 (1986), pp. 244-275.

Greif, I. and Sarin, S. Data Sharing in Group Work. *ACM Transactions on Office Information Systems*. 5, 2 (1987), pp. 187-211.

Grudin, J. Why Groupware Applications Fail: Problems in Design and Evaluation. *Office: Technology and People*. 4, 3 (1989), pp. 245-264.

Halasz, F. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM*. 31, 7 (July 1988), pp. 836-851.

Halstead-Nussloch, R. The Design of Phone-Based Interfaces for Consumers. In *Proceedings of CHI '89 Conference on Human Factors in Computing Systems.* (Austin, TX, 1989). ACM, pp. 347-352.

Harvey, G. *Understanding HyperCard.* SYBEX, Inc., Alameda, CA, 1988.

Hayes, P., Szekely, P. and Lerner, R. Design Alternatives for User Interface Management Systems Based on Experience with COUSIN. In *Proceedings of CHI '85 Conference on Human Factors in Computing Systems.* (San Francisco, CA, 1985). ACM, pp. 169-175.

Hill, R. D. Event-Response Systems-- A Technique for Specifying Multi-Threaded Dialogues. In *Proceedings of CHI + GI '87 Conference on Human Factors in Computing Systems and Graphics Interface.* (Toronto, Canada, 1987). ACM, pp. 241-248.

Hindus, D. and Schmandt, C. Ubiquitous Audio: Capturing Spontaneous Collaboration. In *Proceedings of CSCW '92 Conference on Computer Supported Cooperative Work.* (Toronto, Canada, 1992). ACM, to appear.

Hutchins, E. L., Hollan, J. D. and Norman, D. A. Direct Manipulation Interfaces. In *User Centered System Design.* D. A. Norman and S. W. Draper, Ed. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, pp. 87-124.

Johnson, J. Selectors: Going Beyond User-Interface Widgets. In *Proceedings of CHI '92 Conference on Human Factors in Computing Systems.* (Monterey, CA, 1992). ACM, pp. 273-279.

Kiger, J. L. The Depth/Breadth Trade-off in the Design of Menu-Driven User Interfaces. *International Journal of Man-Machine Studies.* 20, 2 (1984), pp. 201-213.

Kim, W. C. and Foley, J. D. DON: User Interface Presentation Design Assistant. In *Proceedings of UIST '90 ACM Symposium on User Interface Software and Technology.* (Snowbird, Utah, 1990). ACM, pp. 10-20.

Ladner, R., Day, R., Gentry, D., Meyer, K. and Rose, S. A User Interface for Deaf-Blind People (Preliminary Report). In *Proceedings of CHI + GI '87 Conference on Human Factors in Computing Systems and Graphics Interface.* (Toronto, 1987). ACM, pp. 75-80.

Lai, K.-Y., Malone, T. and Yu, K.-C. Object Lens: A "Spreadsheet" for Cooperative Work. *ACM Transactions on Office Information Systems.* 6, 4 (1988), pp. 332-353.

Landauer, T. K. and Nachbar, D. W. Selection from Alphabetic and Numeric Menu Trees Using a Touch Screen: Breadth, Depth, and Width. In *Proceedings of CHI '85 Conference on Human Factors in Computing Systems.* (1985). ACM, pp. 73-78.

Landow, G. P. Relationally Encoded Links and the Rhetoric of Hypertext. In *Proceedings of Hypertext '87.* (Chapel Hill, NC, 1987). ACM, pp. 331-343.

Landow, G. P. The Rhetoric of Hypermedia: Some Rules for Authors. *Journal of Computing in Higher Education.* 1, 1 (1989), pp. 39-64.

Laverson, A., Norman, K. and Shneiderman, B. An Evaluation of Jump-ahead Techniques in Menu Selection. *Behaviour and Information Technology*. 6, 2 (1987), pp. 97-108.

Lee, E. and MacGregor, J. Minimizing User Search Time in Menu Retrieval Systems. *Human Factors*. 27, (1985), pp. 157-162.

MacGregor, J., Lee, E. and Lam, N. Optimizing the Structure of Database Menu Indexes: a Decision Model of Menu Search. *Human Factors*. 28, 4 (1986), pp. 387-399.

Magnum Software Corporation. TFLX Quick Manual: A Simple Overview of Magnum TFLX and its Picture Programming System. 21115 Devonshire Street, Suite 337, Chatsworth, CA 91311, 1990.

Malone, T. W., Grant, K. R., Lai, K.-Y., Rao, R. and Rosenblitt, D. Semi-structured Messages are Surprisingly Useful for Computer-Supported Coordination. In *Computer-Supported Cooperative Work: A Book of Readings*. I. Greif, Ed. Morgan Kaufmann, San Mateo, CA, 1988, pp. 311-331.

Malone, T. W., Lai, K.-Y. and Fry, C. Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. In *Proceedings of CSCW '92 Conference on Computer Supported Cooperative Work*. (Toronto, CA, 1992). ACM , to appear.

Marics, M. How Do You Enter "D'Anzi-Quist" Using a Telephone Keypad? In *Proceedings of the Human Factors Society-- 34th annual meeting*. (Santa Monica, CA, 1990).

Markus, M. L. Towards a 'Critical Mass' Theory of Interactive Media: Universal Access, Interdependence and Diffusion. In *Perspectives on Organizations and New Information Technology*. J. Fulk and C. W. Steinfeld, Ed. Sage Publications, Newbury Park, CA, 1990, pp. 194-218.

Mehlenbacher, B., Duffy, T. M. and Palmer, J. Finding Information on a Menu: Linking Menu Organization to the User's Goals. *Human-Computer Interaction*. 4, 3 (1989), pp. 231-251.

Miller, D. P. The Depth/Breadth Tradeoff in Hierarchical Computer Menus. In *Human Factors Society 25th annual meeting*. (Santa Monica, 1981). pp. 296-300.

Muller, M. J. and Daniel, J. E. Toward a Definition of Voice Documents. In *Proceedings of COIS '90 Conference on Office Information Systems*. (Boston, MA, 1990). ACM, pp. 174-183.

Murray, J. A. H. and Burchfield, R. W. (Ed.) *The Oxford English Dictionary*. Clarendon Press, Oxford, 1933.

Nunamaker, J. F., Dennis, A. R., Valacich, J. S., Vogel, D. R. and George, J. F. Electronic Meeting Systems to Support Group Work. *Communications of the ACM*. 34, 7 (1991), pp. 40-61.

Ochsman, R. B. and Chapanis, A. The Effects of 10 Communication Modes on the Behavior of Teams During Co-operative Problem-solving. *International Journal of Man-Machine Studies*. 6, (1974), pp. 579-619.

Olsen, D. R., Jr. A Programming Language Basis for User Interface Management. In *Proceedings of CHI '89 Conference on Human Factors in Computing Systems*. (Austin, TX, 1989). ACM, pp. 171-176.

Olsen, D. R., Jr., McNeill, T. G. and Mitchell, D. C. Workspaces: An Architecture for Editing Collections of Objects. In *Proceedings of CHI '92 Conference on Human Factors in Computing Systems*. (Monterey, CA, 1992). ACM, pp. 267-272.

Paap, K. R. and Roske-Hofstrand, R. J. The Optimal Number of Menu Options per Panel. *Human Factors*. 28, 4 (1986), pp. 377-385.

Postman, N. *Amusing Ourselves to Death: Public Discourse in the Age of Show Business*. Viking Penguin, New York, 1985.

Remde, J. R., Gomez, L. M. and Landauer, T. K. SuperBook: An Automatic Tool for Information Exploration-- Hypertext? In *Proceedings of Hypertext '87*. (Chapel Hill, NC, 1987). ACM, pp. 175-187.

Repenning, A. and Sumner, T. Using Agentsheets to Create a Voice Dialog Design Environment. In *Symposium on Applied Computing*. (Kansas City, MO, 1992). ACM, pp. 1199-1207.

Resnick, P. and King, M. The Rainbow Pages: Building Community with Voice Technology. In *Directions and Implications of Advanced Computing*. (Boston, MA, 1990).

Resnick, P. and Virzi, R. A. Skip and Scan: Cleaning Up Telephone Interfaces. In *Proceedings of CHI '92 Conference on Human Factors in Computing Systems*. (Monterey, CA, 1992). ACM, pp. 419-426.

Resnick, P. HyperVoice: A Phone-Based CSCW Platform. In *Proceedings of CSCW '92 Conference on Computer-Supported Cooperative Work*. (Toronto, 1992). ACM. To appear.

Resnick, P. The Smart Fast Forward Button. In *Proceedings of the American Voice Input/Output Society*. (Minneapolis, MN, 1992). To appear.

Richards, J. T., Boies, S. J. and Gould, J. D. Rapid Prototyping and System Development: Examination of an Interface Toolkit for Voice and Telephony Applications. In *Proceedings of CHI '86 Conference on Human Factors in Computing Systems*. (New York, 1986). ACM, pp. 216-220.

Roberts, T. L. and Engelbeck, G. The Effects of Device Technology on the Usability of Advanced Telephone Functions. In *Proceedings of CHI '89 Conference on Human Factors in Computing Systems*. (Austin, TX, 1989). ACM, pp. 331-337.

Rosson, M. B. and Mellen, N. M. Behavioral Issues in Speech-Based Remote Information Retrieval. In *Proceedings of the American Voice Input/Output Society*. (San Francisco, CA, 1985).

Samuelson, P. Protecting User Interfaces Through Copyright: The Debate. In *Proceedings of CHI '89 Conference on Human Factors in Computing Systems*. (Austin, TX, 1989). ACM, pp. 97-104.

Schmandt, C. and Arons, B. A Conversational Telephone Messaging System. *IEEE Transactions on Consumer Electronics*. CE-30, (1984).

Shneiderman, B. Direct Manipulation: A Step Beyond Programming Languages. *Behavior and Information Technology*. 1, (1983), pp. 237-256.

Shneiderman, B. Designing Menu Selection Systems. *Journal of the American Society for Information Science*. 37, (1986), pp. 57-70.

Shneiderman, B. User Interface Design for the Hyperties Electronic Encyclopedia. In *Proceedings of Hypertext '87*. (Chapel Hill, NC, 1987). ACM, pp. 189-194.

Shneiderman, B., Kreitzberg, C. and Berk, E. Editing to Structure a Reader's Experience. In *Hypertext/Hypermedia Handbook*. E. Berk and J. Devlin, Ed. McGraw-Hill, New York, 1991, pp. 143-164.

Sisson, N., Parkinson, S. R. and Snowberry, K. Considerations of Menu Structure and Communication Rate for the Design of Computer Menu Displays. *International Journal of Man-Machine Studies*. 25, (1986), pp. 479-489.

Stotts, P. D. and Furuta, R. Petri-Net Based Hypertext: Document Structure with Browsing Semantics. *ACM Transactions on Information Systems*. 7, 1 (1989), pp. 3-29.

Strunk, W., Jr. and White, E. B. *The Elements of Style*. Macmillan, New York, 1979.

Sumner, T., Davies, S., Lemke, A. C. and Polson, P. G. Iterative Design of a Voice Dialog Design Environment. CU-CS-546-91, University of Colorado at Boulder, 1991.

Szekely, P. Template-based Mapping of Application Data to Interactive Displays. In *Proceedings of UIST '90 Symposium on User Interface Software and Technology*. (Snowbird, Utah, 1990). ACM, pp. 1-9.

Szekely, P., Luo, P. and Neches, R. Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. In *Proceedings of CHI '92 Conference on Human Factors in Computing Systems*. (Monterey, CA, 1992). ACM, pp. 507-515.

Travers, M. A Visual Representation for Knowledge Structures. In *Proceedings of Hypertext '89*. (Pittsburgh, PA, 1989). ACM, pp. 147-158.

Trigg, R. H. Tools for Communicating in a Hypertext Environment. In *Proceedings of CSCW '88 Conference on Computer Supported Cooperative Work*. 1988). ACM, pp. 216-226.

Utting, K. and Yankelovich, N. Context and Orientation in Hypermedia Networks. *ACM Transactions on Information Systems*. 7, 1 (1989), pp. 58-84.

Vander Zanden, B. and Myers, B. A. Automatic, Look-and-Feel Independent Dialog Creation for Graphical User Interfaces. In *Proceedings of CHI '90 Conference on Human Factors in Computing Systems*. (Seattle, WA, 1990). ACM, pp. 27-34.

Virzi, R. A., Resnick, P. and Ottens, D. Skip and Scan Telephone Menus: User Performance as a Function of Experience. In *Proceedings of the Human Factors Society*. (Atlanta, GA, 1992). Human Factors Society. To appear.

Voice Messaging User Interface Forum. Specification Document, Final Version. April 1990. Information Industry Association, Suite 800, 555 New Jersey Avenue, NW, Washington, DC 20001.

Walker, J. H. Document Examiner: Delivery Interface for Hypertext Documents. In *Proceedings of Hypertext '87*. (Chapel Hill, NC, 1987). ACM, pp. 307-324.

Wellner, P. D. Statemaster: A UIMS based on Statecharts for Prototyping and Target Implementation. In *Proceedings of CHI '89 Conference on Human Factors in Computing Systems*. (Austin, TX, 1989). ACM, pp. 177-182.

Wenzel, E. M. Localization in Virtual Acoustic Displays. *Presence*. 1, 1 (1991), pp. 80-107.

Weyer, S. A. and Borning, A. H. A Prototype Electronic Encyclopedia. *ACM Transactions on Office Information Systems*. 3, 1 (1985), pp. 63-88.

Wiecha, C., Bennett, W., Boies, S. and Gould, J. Tools for Generating Consistent User Interfaces. In *Coordinating User Interfaces for Consistency*. J. Nielsen, Ed. Academic Press, San Diego, CA, 1989, pp. 107-130.

Wiecha, C. and Boies, S. Generating User Interfaces: Principles and Use of ITS Style Rules. In *Proceedings of UIST '90 Symposium on User Interface Software and Technology*. (Snowbird, Utah, 1990). ACM, pp. 21-30.

Winograd, T. A Language/Action Perspective on the Design of Cooperative Work. In *Computer Supported Cooperative Work: A Book of Readings,*. I. Greif, Ed. Morgan Kaufmann, San Mateo, CA, 1988, pp. 623-653.

Wixon, D., Whiteside, J., Good, M. and Jones, S. Building a User-defined Interface. In *Proceedings of CHI '83 Conference on Human Factors in Computing Systems*. (Boston, MA, 1983). ACM, pp. 24-27.

Zellweger, P. T. Active Paths Through Multimedia Documents. In *Document Manipulation and Typography*. J. C. v. Vliet, Ed. Cambridge University Press, 1988.

Zellweger, P. T. Scripted Documents: A Hypermedia Path Mechanism. In *Proceedings of Hypertext '89*. (Pittsburgh, PA, 1989). ACM, pp. 1-14.

4 1/1 2 -70