Toward Effective Conversational Messaging

Matthew Talin Marx

B.S., Symbolic Systems
Stanford University, 1993

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences
at the
Massachusetts Institute of Technology
June 1995

Author:

Program in Media Arts and Sciences
12 May 1995

Certified by:

Christopher M. Schmandt
Principal Research Scientist, MIT Media Laboratory
Thesis Supervisor

Accepted by:

Stephen A. Benton
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Toward Effective Conversational Messaging

Matthew Talin Marx

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on May 12, 1995
in partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

The ubiquity of the telephone suggests it as an ideal messaging tool. The slow, serial output of speech, however, makes it difficult to find important messages quickly. MailCall, a telephone-based messaging system using speech recognition, takes a step toward effective conversational messaging with a combination of filtering, random access, and recognition error handling. Incoming voice and text messages are categorized and prioritized based on the user's current interests as inferred from the calendar, rolodex, etc. MailCall then updates the speech recognizer's vocabulary on the fly to support random access—so that the user can go to interesting messages directly rather than having to step through the list one message at a time. Inevitable recognition errors are handled by an interface algorithm which verifies potential mistakes and lets the user correct them quickly; further, touch-tone equivalents for several speech commands are provided. And the user can send replies to messages or place calls using the rolodex. The result is a system which retrieves the user's most important messages first, supports flexible browsing of the information space, and tolerates speech recognition errors.

Thesis Supervisor: Christopher M. Schmandt
Title: Principal Research Scientist

This work was supported by Sun Microsystems, Motorola, and Texas Instruments.

Thesis Committee

Thesis Supervisor:

Christopher M. Schmandt
Principal Research Scientist
MIT Media Laboratory

Thesis Reader:

Pattie Maes
Assistant Professor of Media Arts and Sciences
MIT Program in Media Arts and Sciences

Thesis Reader:

Nicole Yankelovich
Principal Investigator, Speech Applications Group
Sun Microsystems Laboratories

Acknowledgments

Chris Schmandt has supervised me during the past two years and played a critical role in shaping this project. He displayed patience as I adjusted to the technical demands of MIT and helped me to develop the perseverance to exhaust all possibilities. His emphasis on usefulness as well as usability has been a stabilizing force in the often eclectic discipline of interactional design.

Pattie Maes generously served as a reader for this thesis; I hope it has been worthy of her time. Two classes she taught, one on autonomous agents and one on information filtering, impacted this work and changed the way I think about computational systems.

Nicole Yankelovich of Sun Microsystems Laboratories sponsored the speech group and served as a reader for this thesis. I thank her along with the other members of the SpeechActs team—Stuart Adams, Eric Baatz, Gina-Anne Levow, and Paul Martin—for an exciting, stimulating summer and subsequent help in working with Swiftus.

Charles Hemphill at Texas Instruments developed Dagger, the speech recognizer used in this project. He responded quickly to my never-ending requests for new features and improved performance. The dynamic grammar loading capability he developed has made an incredible difference in this project.

Lisa Stifelman, my officemate, has had more impact on this project than she knows. Her critical review of various issues including discourse management helped me refocus my efforts. Lisa read an early draft of this thesis and gave many useful comments.

Jeff Herman, a fellow speech group student, helped me with finer details of spoken language interaction. He is one of the most pleasant people I've met and possesses a wealth of insight into interface design. Thanks to him for reading a draft of this thesis.

Jordan Slott cheerfully assisted me in myriad technical matters; without his help, many parts of this project would largely be a list of good intentions. He did much of the work on the text-to-speech server and made it possible to use ISDN for call control.

My father, Gary Marx, encouraged me to stay the course during my adjustment to MIT. Without his lifelong support of education, I would not be writing this thesis now.

My wife Heather has sustained me through much of the time spent at MIT, lifting my spirits when low and inspiring me in her inimitable style. Without her, this all would have been more trying, less enjoyable, and ultimately unsatisfying.

This work is dedicated to the memory of my mother.

Table of Contents

1. Introduction 10

1.1 Why conversational messaging? 10
1.2 Research challenges 11
1.2.1 Liabilities of speech input 11
1.2.2 Liabilities of speech output 12
1.2.3 Processing Messages Efficiently 13
1.3 MailCall: toward effective conversational messaging 13
1.3.1 Message Filtering and Prioritization 14
1.3.2 Presentation and Navigation 14
1.3.3 Avoiding Conversational Breakdown 15
1.4 Document Overview 15
1.4.1 Summary of chapters 15
1.4.2 Typographical conventions 16

2. Background 17

2.1 Multimodal Conversational Systems 17
2.1.1 Put That There 17
2.1.2 Conversational Desktop 17
2.1.3 Pegasus 18
2.2 Speech-only navigation 18
2.2.1 Hyperphone 18
2.2.2 Hyperspeech 19
2.2.3 VoiceNotes 19
2.3 Telephone-Based Messaging Systems 20
2.3.1 Voiced Mail 20
2.3.2 Phoneshell 20
2.3.3 Chatter 21
2.3.4 SpeechActs 21
2.3.5 Wildfire 22
2.4 Electronic Mail Filtering 22
2.4.1 Information Lens 23
2.4.2 Maxims 23
2.4.3 Tapestry 23
2.5 Analysis of Related Work 24

3. MailCall Design Overview 26

3.1 Functionality 26
3.1.1 Unified voice/text message retrieval 26
3.1.2 Sending messages 27
3.1.3 Voice dialing 27
3.2 User Interface 28
3.2.1 Interaction loop 28

3.2.2 Combining speech and touch-tones 29
3.2.3 An "implicit" discourse model 30
3.2.4 Vocabulary Subsetting 31
3.3 A sample MailCall session 32

4. Message Filtering and Prioritization 37

4.1 Capturing short-term or "timely" interests 37
4.1.1 The cost of authoring rules 37
4.1.2 The difficulty of capturing short-term interests with MBR 38
4.1.3 A low-effort, high-benefit approach: CLUES 38
4.2 Gathering clues from information sources 39
4.2.1 Extracting clues from the calendar 39
4.2.2 Correlating with the rolodex 41
4.2.3 Examining outgoing messages and phone calls 43
4.3 Generating rules from clues 46
4.3.1 Names 46
4.3.2 Proper Noun Phrases 46
4.3.3 Individual Words 47
4.3.4 Email addresses 48
4.3.5 Subject lines 49
4.3.6 Phone Numbers 50
4.4 Setup and configurability 50
4.4.1 Minimizing computational requirements 51
4.4.2 Customizing the user's CLUES 51
4.5 Integrating with static rules 51
4.6 Application-independence of CLUES 53
4.6.1 Phoneshell 53
4.6.2 HTMaiL 53
4.7 Two users' experiences 57
4.8 Limitations 59

5. Audio-only navigation and presentation 62

5.1 Limitations of sequential navigation 62
5.2 Category-level random access 63
5.3 Message-level random access 65
5.3.1 Location-based navigation 65
5.3.2 Content-based navigation 66
5.4 Summarizing messages within a category 68
5.4.1 Scanning headers 68
5.4.2 Clustering by sender 69
5.5 Supporting multiple navigation strategies 70
5.5.1 Sequential navigation 71
5.5.2 Content-based navigation 71
5.6 Updating the vocabulary on the fly 72
5.6.1 Updating vs. Subsetting 72
5.6.2 Meeting user expectations 73
5.6.3 Updating one-time, user-specific vocabulary 74
5.6.4 Updating and re-updating information about messages 74
5.7 Limitations 76
5.7.1 Specifying names 76
5.7.2 Other content on which to base random access 77

6. Avoiding conversational breakdown 79

6.1 Learning the system's vocabulary 79
6.1.1 Directive prompts 79
6.1.2 Time-out prompts 81
6.2 Context-sensitive help 82
6.2.1 Step 1: re-establish context 82
6.2.2 Step 2: list the current options 83
6.2.3 Step 3: offer a way out 84
6.3 Providing coherent, comprehensible feedback 85
6.3.1 Varying the speed of output 85
6.3.2 Adjusting speed for given vs. new information 86
6.4 Effectively handling recognition errors with ErrorLock 87
6.4.1 Progressive assistance for rejection errors 88
6.4.2 Identifying potential substitution errors 89
6.4.3 The language of verification 92
6.4.4 Facilitating one-step correction 93
6.4.5 Taking the blame for recognition difficulties 94
6.4.6 Dealing with consecutive substitution errors 95
6.5 Touch-tones 97
6.6 Summary 97

7. System Architecture 99

7.1 Hardware, operating systems and programming languages 99
7.2 Asynchronicity 100
7.3 Server Architecture 100
7.3.1 Existing speech group servers used in MailCall 100
7.3.2 Enabling dynamic grammars within the recognition server 101
7.3.3 Preparing a grammar for on-the-fly expansion 103
7.3.4 Creating a text-to-speech server 104
7.4 Third-party software 105
7.4.1 Speech recognition 105
7.4.2 Speech synthesis 106
7.4.3 Call control 106
7.4.4 Parsing 106
7.5 Run-time options 107

8. Conclusions 109

8.1 Distinguishing MailCall from previous work 109
8.2 Insights and Issues 109
8.2.1 Filtering 110
8.2.2 Speech as a control channel 110
8.2.3 Speech as feedback 111
8.3 Next Steps 111

Appendix A: Recognition Grammar (for Dagger) 113

Appendix B: Biographies of the Committee 117

References 118

List of Figures

Figure 1.1: A sample MailCall session....14
Figure 1.2: Formatting of conversational outtakes. 16
Figure 1.3: Formatting of recorded speech....16
Figure 1.4: Annotated transcripts....16
Figure 3.1: The interaction loop of MailCall. 28
Figure 3.2: The MailCall DTMF equivalents. 29
Figure 3.3: A finite-state diagram of MailCall interaction. 30
Figure 3.4: Vocabulary subsetting for each dialogue state. 32
Figure 3.5: Identifying the user by password....33
Figure 3.6: Asking for messages from a certain user....34
Figure 3.7: Hearing a summary and read messages from a specific sender. 34
Figure 3.8: Sending a reply to an incoming message. 35
Figure 3.9: Handling substitution errors. 35
Figure 3.10: Handling consecutive rejection errors and giving help....36
Figure 3.11: Voice dialing. 36
Figure 3.12: Terminating a MailCall session. 36
Figure 4.1: A set of user-authored filtering rules. 37
Figure 4.2: Sample calendar entries. 40
Figure 4.3: A sample rolodex card....42
Figure 4.4: Summary-level view of HTMaiL. 54
Figure 4.5: An email message formatted in HTML....55
Figure 4.6: Accessing voice messages through HTMaiL. 56
Figure 4.7: Sending a reply using a Netscape "mailto:" tag. 59
Figure 4.8: Explaining why a message was important. 60
Figure 5.1: Navigating messages sequentially in Chatter....62
Figure 5.2: Groupwise sequential navigation in SpeechActs. 64
Figure 5.3: Random access to categories using DTMF tones. 64
Figure 5.4: Category-based random access in VoiceNotes....65
Figure 5.5: Asking for messages from a specific sender. 67
Figure 5.6: Summarization and random access in MailCall. 67
Figure 5.7: Scanning headers in SpeechActs. 68
Figure 5.8: "Fuzzy" equivalents for message totals. 70
Figure 5.9: A clustered summary of a category. 70
Figure 5.10: Sequential navigation in MailCall. 71
Figure 5.11: Content-based querying. 72
Figure 5.12: Hearing a summary twice....72
Figure 5.13: "Graying out" portions of a recognition grammar for improved performance. 73
Figure 5.14: Returning to a message from a certain sender....76
Figure 5.15: Supporting reference to people by first name only. 77
Figure 5.16: Random access to subject lines. 77
Figure 5.17: Asking for messages related to events in the user's calendar....78
Figure 6.1: IVR-style listing of options....80
Figure 6.2: Sending a message with verbose prompts. 81
Figure 6.3 Sending a message with terse prompts. 81

Figure 6.4: Reestablishing context through help....83
Figure 6.5: Explaining the user's current options....84
Figure 6.6: Increasing the speech output rate....85
Figure 6.7: Slowing down while presenting new information....87
Figure 6.8: The "brick wall" effect of rejection errors....88
Figure 6.9: Avoiding the "brick wall" effect with progressive assistance....89
Figure 6.10: Asking too many verification questions....90
Figure 6.11: Verifying a command to hang up....90
Figure 6.12: an utterance structure....92
Figure 6.13: the grammar entry for the "hang up" command....92
Figure 6.14: Asking the wrong verification question....93
Figure 6.15: Just saying "yes" but not saying just "no." 93
Figure 6.16: Correcting a mistake in two steps....94
Figure 6.17: Progressive assistance for multiple substitution errors. 95
Figure 6.18: Making the same mistake twice in a row....96
Figure 6.19: Overcoming repetitiveness by tracking past attempts. 96
Figure 6.20: Exploiting N-Best in handling multiple substitution errors....96
Figure 6.21: ErrorLock, an interface algorithm for handling recognition errors....98
Figure 7.1: MailCall's servers and their interaction....101
Figure 7.2: run-time options for debugging. 108

1. Introduction

A critical part of being an effective colleague, spouse, parent, or friend is *communication*. While there is no substitute for a face-to-face conversation, geographical distribution and differing schedules necessitate the use of computational tools to stay connected. Keeping in touch is difficult enough while in the office and even more problematic when on the road. Besides the anxiety of feeling "out of touch," the incommunicado nomad may impede the progress of a work group, miss a chance to talk with a close friend, or fail to respond to a family emergency. Meanwhile, voice mail and electronic mail pile up, resulting in a personal information glut to be dealt with when returning to the office. The alternating unavailability and overflow of information can wreak havoc with one's productivity and peace of mind.

An effective tool for remote access will help the user put first things first by prioritizing messages based on the user's current interests. It will also leverage the information available in the user's desktop work environment in order to avoid needless, annoying duplication of data. And it will be usable in common situations while away from the desk—walking down the hall, sitting in someone else's office, riding the subway, driving a car, waiting at the airport, etc.

1.1 Why conversational messaging?

Today, the nomad must enlist several partial solutions to the problem of keeping in touch. Most voice mail systems and many personal answering machines allow remote retrieval of messages via the ubiquitous telephone, but such systems are usually self-contained; the user cannot send replies to people outside the local voice mail system. Further, such systems often fail to integrate with desktop tools like a rolodex, so calling someone who has left a message often involves hanging up and dialing again. Finally, since one often does not know whether someone will respond by calling or sending electronic mail, it is necessary to have remote access to electronic mail, as well.

Accessing electronic mail anywhere, anytime requires carrying a laptop computer or PDA equipped with a modem (which is not always easily connected to the network).

Even if the modem is cellular, the portable device can be inconvenient to transport and difficult to use in situations where the user's hands and eyes are busy, like driving.[1]

Ideally, one would have access to both voice and text messages with a ubiquitous, networked tool. The goal of the MailCall project is to provide an integrated messaging environment where voice and text messages are processed, prioritized, and presented uniformly, and where the user can respond in kind. Further, MailCall provides a level of integration with the user's desktop environment: for instance, rolodex entries can be accessed, and messages which correlate with events in the calendar are marked as important.

1.2 Research challenges

Although calling up and retrieving messages by voice is an appealing method of messaging, several liabilities of speech as an input/output modality make for a challenging design problem. The potentially large information space of incoming messages requires special attention to issues of audio-only navigation.

1.2.1 Liabilities of speech input

Voice mail systems offering remote access typically give the user a set of touch-tone commands for listening to messages, replying to them, and deleting unwanted ones. Touch-tone systems can be difficult to use, however, due to the impoverished interface of the twelve-button keypad. The paucity of input options leads to hierarchies of commands which can be difficult for the novice to learn, especially since the keys are rarely related to their associated commands: for instance, why should one press '2' for 'delete' and not '3'? The command layout cannot entirely avoid arbitrariness [Marx & Schmandt 1994b].

Speech recognition can overcome the liabilities of touch-tone systems and offer the user more flexible, conversational interaction. Speech is relatively effortless for the user and can be employed when hands are eyes are busy. Since command and control are accomplished through natural language, one does not have to associate arbitrary keys with commands. Further, the number of active commands is not limited by the number of keys but can grow with the capabilities of the recognizer. Finally, speech is an inherently rich means of communication and utterly familiar [Chalfonte et. al. 1991].

[1] Laws in some countries prohibit the dialing of cellular phones when a car is in motion.

Although speech recognition overcomes some of the liabilities of touch-tones, it introduces a few of its own. A major challenge is conveying to users what the system understands since speaking outside the vocabulary results in misrecognition and probably misunderstanding. Some users, when told that the system will recognize their speech, will speak as if conversing with a friend: spontaneously, full of stops and false starts, corrections, and backtracking. In order to guide users into staying within the vocabulary of the system, the designer must present the available options in a concise, effective way.

Despite one's best efforts, recognition errors will and do occur. If recognition errors are not handled in a consistent, helpful manner, users may lose trust in the system and form a distorted mental model of its behavior [Yankelovich et. al. 1995]. Recognition errors appear in a variety of forms, and each must be handled appropriately. Though people misunderstand each other once in awhile, they recover quickly and get on with the conversation. An effective speech-driven system must pay special attention to handling multiple recognition errors, since a machine which repeatedly makes the same mistake quickly loses the user's confidence.

Despite the drawbacks of touch-tones, they can complement speech for certain tasks and in certain contexts. A combination of the two may prove most effective, since a system may be used in a variety of situations. While driving, for instance, speaking is safer than using touch-tones, yet in a noisy environment DTMF detection will be more reliable than speech recognition.

1.2.2 Liabilities of speech output

Designing speech-only systems in any domain is challenging due to the lack of persistent, visual output. Audio is both temporal and serial; it is slow for the listener to consume large amounts of speech (hence the frustration with listening to long menus over the phone), and the user must rely upon short-term memory since audio does not persist like a visual display [Schmandt 1994a].

Informing the user of current options and giving feedback is thus especially difficult. The invisibility of speech requires more explanation and help for the user, yet the slow output rate raises the cost of doing so. Hence, it is crucial to know when to give assistance and when to remain terse. Yet it is not as if a single solution exists which will satisfy every user, or even satisfy a single user over a long period of time. Whereas the

novice requires hand-holding, the expert finds wordy explanations annoying. Striking and continually re-striking a balance between exposition and brevity is a challenge.

In the messaging domain, working without a visual display is especially difficult when a large number of messages are to be presented. With a GUI-based mailer, the user can visually scan the list of messages and select the ones which look interesting; in audio, however, scanning is more difficult since speech is slow and serial, and selecting is harder since speech is transient. Listening to a message header, choosing whether to read it, and moving to the next one if not can prove very tedious. Effectively manipulating a large information space in audio requires attention to issues of presentation and navigation.

1.2.3 Processing Messages Efficiently

Anyone who receives dozens of email messages each day knows the value of separating the wheat from the chaff and putting the most important messages at the top of the list. This is a boon at the desktop, for filtering can save the user time and be a great convenience, but it is simply indispensable when calling up to get messages over the phone. For one, digging through dozens of messages one by one to find what's important is very slow and can be extremely frustrating. In addition, when the user is calling long-distance or from a cellular phone—two situations in which telephone-based message retrieval is likely to be most useful—wasted time is wasted money.

At the same time, if the set-up and maintenance costs outweigh the benefits of filtering, the user may abandon the system altogether. The challenge is to produce a filtering and prioritization system which provides immediate, enduring benefit without incurring high setup and maintenance costs.

1.3 MailCall: toward effective conversational messaging

MailCall, the conversational messaging system described in this document, exploits the complementary attributes of speech and Dual-Tone Multi-Frequency (DTMF) tones for input. It offers telephone-based access to both email and voice mail, and allows the user to send voice messages or place outgoing calls in response or to anyone in the user's rolodex. It is designed to facilitate both rapid, goal-directed interaction and more casual browsing. MailCall enables transactions like the following:

MailCall:	Welcome to MailCall. Who's this?
User:	This is Matt Marx.
MailCall:	Matt Marx?
User:	Yes.
MailCall:	Hi Matt! Please say your six-digit password.
User:	Six four one nine seven zero.
MailCall:	Hold on while I get your messages....Okay! You have two timely messages, five personal ones, and sixty-six others.
User:	Is there anything from Chris Schmandt?
MailCall:	Timely messages from Chris Schmandt. Message 2 from Chris Schmandt about "thesis."
User:	Read it.
MailCall:	"Come on by if you want to talk about the thesis; I'll be here for another twenty minutes or so."
User:	Send a reply.
MailCall:	Record your voice message for Chris Schmandt.
User:	*Chris, this is Matt. I'm on my way and will be right there.*
MailCall:	Got it. Review the message, send it, or cancel?
User:	Send.
MailCall:	Message sent.
User:	That's all for now.
MailCall:	Did you say to hang up?
User:	Yes.
MailCall:	Matt, thanks for calling. Good bye!

Figure 1.1: A sample MailCall session.

MailCall enables very fluid interactions between people and machines using the ubiquitous telephone. It distinguishes itself from other messaging systems in three aspects: filtering, navigation, and error management.

1.3.1 Message Filtering and Prioritization

MailCall filters messages and presents them in categories defined by the user. In a special category called *timely* are placed messages deemed relevant because some line in the message header correlates with a calendar entry or because the sender is someone the user has been trying to contact recently. CLUES, the filtering and prioritization mechanism used by MailCall, scans personal information databases including one's calendar, to-do list, rolodex, and list of people one has recently sent email or tried to call. It catches relevant messages automatically, alleviating the need to write and maintain long lists of rules. See Chapter 4 for more details about CLUES.

1.3.2 Presentation and Navigation

Sorting messages into categories can help, but when there are a lot of messages within a category the user is again faced with the dilemma of how to find the most interesting

messages in a long list. MailCall facilitates rapid browsing by summarizing the contents of a message category by sender and then updating the vocabulary of the speech recognizer accordingly. As a result, the user can zero in on messages of interest: e.g., "read me the message from Lisa Stifelman." For more details on presentation and navigation strategies, see Chapter 5.

1.3.3 Avoiding Conversational Breakdown

MailCall uses an interface algorithm called ErrorLock to keep the conversation afloat despite recognition errors. It uses a confidence threshold to screen out potential misrecognitions (i.e., where the speech recognizer is only 50% sure that it is correct) and checks first with the user before executing potentially damaging utterances like "delete the message." It allows the user to correct mistakes in a single step rather than insisting on yes/no answers to questions. Finally, ErrorLock keeps a history of mistakes so that MailCall can avoid making the same mistake twice and apologize when it is having excessive difficulty. More details can be found in chapter 6.

1.4 Document Overview

An overview of the chapters and a list of typographical conventions are provided below.

1.4.1 Summary of chapters

Chapter 2, "Background," describes relevant work in the fields of conversational interaction, telephone-based messaging, audio-only navigation, and electronic mail filtering. Issues pertaining to the design of MailCall are raised.

Chapter 3, "Design Overview," describes the system's functionality and highlights features which distinguish MailCall from other conversational messaging systems. A sample session with the system is described.

Chapter 4, "Message Filtering and Prioritization," describes the CLUES filtering system in detail: how it deduces interesting items from personal information sources, builds rules from these items, and integrates its automatically generated rules with user-authored rules. HTMaiL, a World-Wide Web browser interface to reading email, is described, as are the experiences of two users using MailCall over several months.

Chapter 5, "Audio-only Navigation and Presentation," describes the MailCall approach to dealing with a large number of messages.

Chapter 6, "Avoiding Conversational Breakdown," details strategies for helping users to learn and use the system's vocabulary, and for handling recognition errors.

Chapter 7, "System Architecture," outlines the hardware and software used in the construction of MailCall. Various asynchronous servers are described, as are third-party tools for speech recognition, synthesis, and parsing. Debugging options are also described.

Chapter 8, "Conclusions," recapitulates the main points of the document, suggests extensions to MailCall, and outlines avenues for future work.

1.4.2 Typographical conventions

This document includes many transcripts of conversations between MailCall and the user. These outtakes are indented and bordered for easy reading.

User:	Is there a message from Jeff Herman?
MailCall:	Sorry, no messages from Jeff Herman.

Figure 1.2: Formatting of conversational outtakes.

Wherever voice messages are played or recorded, text is italicized, as in the following.

MailCall:	Send a voice message to Chris Schmandt.
User:	Record your voice message after the tone.
MailCall:	*Chris, this is Matt. Let me know if you have time to meet today.*

Figure 1.3: Formatting of recorded speech.

Occasionally the transcript will be annotated. This is in bracketed bold text:

MailCall:	What now?
User:	Go to my timely messages. **<MailCall hears "personal messages" by mistake>**
MailCall:	Going to your personal messages.

Figure 1.4: Annotated transcripts.

Code snippets or entries in personal information databases are listed in Courier font:

```
Dec 12 3pm meeting in Bartos
Dec 13 all day visit at Texas Instruments
```

2. Background

MailCall draws on a legacy of research in conversational systems, telephone-based messaging, browsing in audio, and mail filtering. Understanding the work upon which MailCall builds—and, in several cases, integrates—is key to measuring its contribution.

2.1 Multimodal Conversational Systems

Speech recognition has principally been used for keyboard or mouse replacement, such as for dictation [Danis et. al. 1994] or GUI command and control [Strong 1993]. Listed below are interactive projects which utilize speech both as input and output.

2.1.1 Put That There

The precocious Put That There project [Bolt 1980] combined spoken and gestural input with spoken and visual output to support natural interaction in a resource management simulation. Looking at a wall-size map of the world, the user could create and locate sea vessels in a natural and flexible way, giving a command "put", pointing to an object "that", and pointing to its desired location "there." The resolution of ambiguous anaphora with gestural deixis was both novel and powerful, and system engaged the user in a dialogue to complete tasks: "put that...where? there." Small-vocabulary, speaker-dependent, discrete-word recognition was used, so the system was only accessible by a small community of users under controlled conditions, but its conversational style demonstrated the potential of speech recognition in interactive systems.

2.1.2 Conversational Desktop

Richer conversational interaction was demonstrated by the Conversational Desktop [Schmandt 1987]. The Conversational Desktop integrated rolodex, calendar, and messaging, providing an experience not unlike that of having a live secretary. It could even tell if the user was addressing it or someone else in the room by measuring the output of multiple directional microphones.

Using recognition technology similar to Put That There, the Conversational Desktop was capable of processing partial recognitions. If, for instance, it heard "Schedule a meeting with _____ tomorrow," its repair mechanism re-prompted the user by saying "With whom do you wish to meet?" [Schmandt 1986] Although current speech recognizers usually return a full sentence or nothing at all, the attention paid to error correction proved crucial for subsequent systems.

2.1.3 Pegasus

The Pegasus system [Zue et. al. 1994] provided spoken language interaction for an on-line air travel reservation system. The system engaged the user in a goal-directed dialogue for making air travel reservations. Pegasus displayed flight information on the screen and asked supporting questions when necessary.

Pegasus tracked the dialogue as a finite-state diagram with approximately 30 different discourse states. In an effort to minimize the toll of recognition errors, users were offered an "undo" option: saying "scratch that" would undo the last command, and saying "clear history" would put the user back at the beginning, resetting all dialogue states. In this way, Pegasus was able to avoid slowing down the interaction by asking verification questions.

2.2 Speech-only navigation

While most interactive speech-recognition systems have employed a visual component for all or part of the feedback, a telephone-based system does not enjoy such a luxury. Described below are several early investigations into systems employing speech as the sole interactional medium.

2.2.1 Hyperphone

Hyperphone [Muller & Daniel 1990] is a speech-only document browser using speech recognition for input and speech synthesis for output. Typed links were constructed between different sections of the text to enable voice-only browsing; the user may follow a linear sequence of navigation known as a Parsed Voice String or interrupt a pre-structured sequence to follow a related topic. A history of user actions is kept as a set of virtual links so that one can retrace one's steps.

Because the speaker-independent speech recognizer used a precompiled vocabulary, and because so few technical words were available in its vocabulary, the user could not reference the content of an item by voice (i.e., "read me the one about 'metallurgical research'"); instead, *voice-direct manipulation* was so that the user could interrupt a list of several options by saying "that one." Interruptibility was identified as a necessary feature.

2.2.2 Hyperspeech

Similar to Hyperphone, the Hyperspeech project [Arons 1991] explored audio-only navigation. Here, however, the information space consisted of interviews with various people on a single topic, hand-parsed into voice snippets with typed links for following a speaker or topic.

Isolated-word recognition was used for navigation, with different commands for different types of navigation. To follow a topic, one could say "more," whereas saying "name" would go to the next snippet from the current speaker. Interruptibility was again singled out as an important feature due to the serial nature of output. It was claimed that an optimal speech interface is modeless, with the entire vocabulary active at all times.

2.2.3 VoiceNotes

Like Hyperphone and Hyperspeech, VoiceNotes [Stifelman 1992, Stifelman et. al. 1993] focused on navigating in an audio-only environment. Its premise was different, though: the goal was to provide a hand-held tool for recording spontaneous thoughts, reminders, etc. for later retrieval. The user could create categories and then record individual voice notes within those categories; for instance, the "things to do" category might contain the notes "pick up bread on the way home," "give Lisa a copy of my thesis to read," and "find summer housing." Notes could be created, played, deleted, or moved to another category.

The dynamic nature of the information space led to the exploration of navigation in a single-level hierarchy. Since the user defined a new category by speaking the category name, speaker-dependent recognition allowed the user to navigate among categories. Navigation within categories was sequential, however, with buttons mimicking the actions of pre-trained commands like "next" and "previous." The combination of voice

and buttons proved to be powerful, with buttons most useful in replacing repetitive, fine-grained actions like moving among notes within a category.

2.3 Telephone-Based Messaging Systems

As the telephone becomes recognized as a prime messaging tool, several systems offering telephone-based access to personal information and messaging capabilities have sprung up, including some commercial ventures. The section below describes recent work in telephone-based messaging systems.

2.3.1 Voiced Mail

Unified messaging (i.e., both text and voice) over the telephone was first explored in the Voiced Mail system [Schmandt 1984]. A user could call up and hear voice and text messages sorted by sender independent of medium. Interruptibility was identified as a key component of a speech-only system. A graphical component was later added [Schmandt 1985], resulting in a unified messaging system usable at the desktop or on the road.

2.3.2 Phoneshell

Mobile access to desktop applications was the focus of the Phoneshell project [Schmandt 1993]. Using any touch-tone telephone, the user could access voice messages, text messages, calendar entries, rolodex cards, and various news sources—much of the information available while sitting in front of the computer. Some data entry was possible, as well; the user could create calendar entries and rolodex cards using DTMF tones and type text replies to messages. Thus the Phoneshell user could maintain a high degree of productivity while on the road.

Despite the breadth and completeness of its functionality, Phoneshell is constrained by the limitations of the telephone keypad. Its exploitation of the twelve buttons was remarkable, though novices sometimes struggled to learn the commands. (It should be noted that Phoneshell offered lists of menu items as well as context-sensitive help as assistance.) Once key combinations were learned, however, the expert could execute actions quickly and with confidence since DTMF recognition is robust.

2.3.3 Chatter

An attempt to provide the functionality of Phoneshell while improving the interface, Chatter [Ly 1993] used continuous, speaker-independent speech recognition for input. It did not provide the full suite of applications that Phoneshell did, focusing instead on communication within a workgroup: reading electronic mail, sending voice messages to workgroup members, looking up people in the rolodex, and placing outgoing calls. Speech recognition enabled the simultaneous activation of many more commands than was possible with DTMF tones. A discourse manager based on the Grosz & Sidner model [Grosz & Sidner 1986] tracked the user's place in the dialogue and preserved context among separate tasks—after reading a message from someone, the user could look up their phone number in the rolodex and then call them.

The incorporation of memory-based reasoning (MBR) in a spoken language system was a major contribution of Chatter. [Ly & Schmandt 1994] Observing the user's history of actions, Chatter made predictions about the user's next action. If, for example, the user always saved messages from Eric Baatz, instead of asking what to do with a message from Eric Baatz, Chatter would suggest, "Save it?" Such adaptive behavior had a twofold advantage: one, it could increase user confidence in the system; two, by reducing much of the interaction to "yes" and "no" responses, the likelihood of recognition errors was diminished.

Recognition errors did not disappear, however. Since Chatter did not verify the output of the speech recognizer, it was prone to erroneous action like hanging up unpredictably during a session or deleting a message by mistake. Also, since the filtering system presented a relevance ranking but no categorization, navigating many messages sequentially ("next message...previous message") could become tedious.

2.3.4 SpeechActs

The SpeechActs project [Yankelovich & Baatz 1994] provides access to a set of applications similar to Phoneshell: voice/text messages, calendar, stock quotes, weather, and reminder calls. Like Chatter, it used speech recognition for command and control. To restrict the grammar size, however, it adopted Phoneshell's notion of switching between applications. Language management was facilitated by the Unified Grammar and Swiftus parser [Martin 1994], which allowed for recognizer-independent, generalized grammar specification, and which synchronized the recognition and parsing grammar.

SpeechActs paid attention to issues of recognition errors, and user studies [Yankelovich et. al. 1995] confirmed the importance of avoiding the "brick wall" effect of repeated mistakes. The mechanisms for doing so were distributed among the applications, so one interaction may have behaved differently from another when, for example, dealing with a substitution error. SpeechActs offered message filtering and categorization, and allowed the user to skip over or delete uninteresting groups of messages. Unlike Chatter, SpeechActs did not maintain an elaborate discourse model—it is not clear, however, how a more elaborate discourse model would have improved SpeechActs.

2.3.5 Wildfire

Wildfire [Wildfire 1995] renders the functionality of several MIT Media Lab speech group projects in a commercial product. Like the Phone Slave [Schmandt 1984], it acts as an interactive answering machine, interactively getting the caller's name, phone number, and message. It routes incoming calls to another number—either based on knowledge about the user [Wong 1991] or based on what the user tells the assistant—and allows the user to screen incoming calls by listening to the name of the caller. As with Chatter, the user can retrieve messages, check a personal rolodex, and call people back using voice dialing. Like the Conversational Desktop, the user can schedule reminder messages to be played at a certain time. Finally, the user can ask the whereabouts of other system users, similar to Chatter.

Due to the discrete utterance recognition used in the system, a Wildfire session can be less conversational than a system using continuous speech recognition. A combination of speaker-independent recognition for commands and speaker-dependent recognition for names leads the user through a menu-style interface with sometimes choppy dialogues, since the two types of recognition are not mixed. The user cannot say "call John at home" but must break up the command into "call," "John," and "at home," waiting between each for the system prompt. Its limitations aside, Wildfire is a brave first step towards bringing speech-driven messaging systems into the marketplace.

2.4 Electronic Mail Filtering

With current and future growth in messaging technologies, particularly electronic mail, the "information glut" [Postman 1992] is no longer simply a horizon too broad to be explored in a reasonable amount of time; it is a salesman knocking at the door, an old friend calling to ask a favor, and myriad nameless addresses petitioning for one's time.

2.4.1 Information Lens

One of the earliest attempts to filter electronic mail, Information Lens [Malone et. al. 1987, Malone et. al. 1989] relied on user-authored rules to filter out unimportant messages. Instead of sending messages to their recipients directly, users targeted communications at the *Anyone* server, which then processed the messages and redirected them where intended. Extra fields could be added to the message header to facilitate processing, similar to The Coordinator [Winograd & Flores 1987], which relied on users to label each message's conversational act.

The rule-based filtering pioneered by Information Lens has been popularized in both Procmail [van der Berg 1993] and Eudora [Dorner 1988]; Eudora provides a graphical user interface for writing and maintaining lists of rules.

2.4.2 Maxims

A learning-based approach was taken in hopes of reducing the amount of work a user had to perform in using a filtering system [Metral 1992]. Using Memory-Based Reasoning (MBR) [Stanfill & Waltz 1986], Maxims, an appendage to the Eudora electronic mail handler tried to anticipate the user's action for a given message. A generic framework for building learning interface agents was also presented.

One limitation cited was the time necessary for the filtering agent to develop a model of the user's interests. Subsequent work focused on helping agents to collaborate in order to overcome the steep ramp-up associated with a new learning agent [Lashkari et. al. 1994]. Improved performance was achieved when several agents cooperated in predicting the user's action, while still requiring no work on the part of the user.

2.4.3 Tapestry

Collaborative filtering was also the focus of the Tapestry project [Goldberg et. al. 1992], though the agents were not computational but human. System users annotate incoming documents, and those annotations are processed by Tapestry to inform other users of the message's potential for interest. Users are also able to "look over the shoulder" of others whose opinions they value. The system is most useful when multiple users are receiving identical messages, such as in a work group or where mailing lists are common; it does not, however, assist the individual dealing with messages unique to a sender.

2.5 Analysis of Related Work

MailCall synthesizes research in several different areas—conversational interaction, speech-only navigation, telephone-based access, and information filtering—meanwhile filling some gaps left by prior work.

Conceptually, MailCall might be viewed as a combination of Chatter, SpeechActs, Voiced Mail, and Phoneshell. It unifies the presentation of voice and text messages from Voiced Mail with the multimedia reply capabilities of Phoneshell, and adopts an conversational style inspired by Chatter and SpeechActs. Its contributions to conversational interaction are ErrorLock, a general interface algorithm for dealing with possibly erroneous input, and a sensitivity to the conversational context in generating feedback, especially through the use of ellipsis.

The application of certain speech-only navigation techniques to the messaging domain is novel. Although many telephone-based systems have offered remote access to voice and/or text messages, they have largely restricted the user to linear navigation through the list of messages. Projects such as Hyperspeech and VoiceNotes suggest a random access and the use of categories. Interestingly, these two projects have orthogonal information landscapes: with Hyperspeech, the nodes and links are authored by the system designer, and the user must discover them; with VoiceNotes, items are created and categorized by the user, so the information space is predictable. MailCall is unique in that the user is navigating an information space created by multiple people (the message senders) and which is different with each session. Since the user in a sense "starts fresh" each time, giving an overview of the information space and allowing rapid navigation is especially important.

Categorization of items is common to messaging systems. Like Phoneshell and SpeechActs, MailCall—or more specifically, the CLUES filtering system—sorts messages into categories based on user preferences. CLUES, however, fills a void in filtering systems which user modeling research [Orwant 1993] is beginning to address: capturing highly volatile, short-term user interests. Without any extra work on the part of the user, MailCall is able to present messages in order of relevance, singling out those which might be especially important due to factors not quickly captured by MBR or conveniently represented as a set of rules.

In summary, MailCall draws on the interactional style of other conversational systems, adding an interface formalism for handling recognition errors. Its filtering is inspired by

both rule-based and learning approaches, though its addition of timely filtering is novel. Finally, its exploitation of dynamic grammar loading opens up new possibilities for navigating an information space. The following chapter describes the general design of MailCall.

3. MailCall Design Overview

This chapter gives a high-level description of MailCall. Its functionality, user interface, and information flow are described in turn, with an annotated sample session at the end.

3.1 Functionality

The goal of MailCall is to help the user keep abreast of incoming voice and text messages when away from the desk, and to facilitate communication with those who have sent messages as well as those in the user's rolodex.

3.1.1 Unified voice/text message retrieval

The MailCall user can access voice and text messages in a unified manner. Electronic mail is kept in a spool file on a UNIX workstation; the spool file is parsed and run through the filtering/prioritization mechanism at runtime.[2] The user can ask for details on a certain message: the sender, subject, time it arrived, who it was addressed to, and what its priority is. Using touch-tones the user can jump to the next or back to the previous sentence of a message, or skip over blocks of included text.

MailCall is a multimedia messaging system. Messages with voice attachments are processed and played for the user as voice messages. In addition, a homegrown voice mail system [Stifelman 1991] sends the user email notification of incoming voice mail with the sender's name and phone number if known, as well as the path name of the sound file containing the message. So when MailCall encounters an email notification of voice mail, it plays the corresponding sound file instead of reading the notification email. Since this process is transparent to the user, voice and text messages are presented similarly.

[2] The code for parsing the mail spool file and processing voice attachments was adapted from Phoneshell; its authors include Chris Schmandt, Angie Hinrichs, Sumit Basu, Lisa Stifelman, and Eric Ly.

3.1.2 Sending messages

A voice reply can be sent to any message.[3] If the recipient is a local voice mail subscriber, the message will be delivered as voice mail with the appropriate email notification sent. If not, the messages will be encoded and sent as multimedia electronic mail over the Internet. MailCall supports the Sun Multimedia Format, NextMail, Mime, and uuencode. People in the in the user's rolodex as well as system users can also be sent messages. In an attempt to minimize repetitive interaction, MailCall checks first in the user's rolodex to see if a particular voice format has been stored for the recipient. If so, that format is used. If not, the user is prompted for a format, and the result is stored in the rolodex for use next time.[4]

While sending a voice reply is as simple as recording a voice message and choosing a format, sending text replies is trickier. Telephone-based speech recognition is not yet reliable enough to support dictation of text messages, and editing such messages presents an even greater interactional challenge. An alternative is to type messages letter by letter using DTMF tones. Two methods of typing text messages are available, one which requires two tones to explicitly specify each letter [Schmandt 1993], and another using one tone per letter and then disambiguating the input using information theoretic constraints [Rau & Skiena 1994]. The integration of these techniques into MailCall is not complete at time of this writing.

3.1.3 Voice dialing

Instead of sending a voice reply, the user may wish to establish contact by placing a call. If the sender is on the local voice mail system, or if the phone number is to be found in the rolodex, then that number is used. (And if both a work and home number are available, the user is asked to pick one or the other.) If not, the user may speak the number to be dialed or type it using DTMF tones. Call control is facilitated through a telephone server which connects to an Integrated Digital Services Network (ISDN) server, allowing software-based call control.

[3] Since the caller ID system is not comprehensive, there are some voice messages whose origin cannot be traced. Such messages are marked as being from "Unknown," and the user cannot reply to them. But the user can call or send a message to anyone in the rolodex, or dial any phone number, so the difficulty is mitigated.

[4] If no rolodex card exists for the message recipient, one is created so that the sound format can be stored for future use. Whether this leads to an unacceptably high number of rolodex cards remains to be seen.

3.2 User Interface

One of the challenges of designing a spoken language system is effectively setting and meeting user expectations. It is often the case that pre-design studies for speech interfaces suggest a conversational style which exceeds the capabilities of current speech recognizers [Yankelovich 1994, Herman 1995]. Consequently, the interactions possible with current technology are more simple and less rich than human-human conversation. At the same time, touch-tones can compensate for some of the weaknesses of the recognizer and even introduce new possibilities for interaction. This section describes the basic interaction strategy of MailCall, its combination of speech and touch tones, and its approach to modeling discourse.

3.2.1 Interaction loop

The interaction of MailCall mimics that of conversation, which consists of a series of exchanges often described as *adjacency pairs* [Coulthard 1977]. An adjacency pair is a set of two utterances, the first soliciting the second as a response. The most common adjacency pairs in a MailCall dialogue are *question/answer* ("What's your name?" "Matt Marx.") and *menu/selection* ("Do you want your personal messages or timely messages?" "Timely messages."). In the former, the interaction is system-driven, and in the latter it is user-driven. The following diagram shows the information flow in such exchanges; although the system issues the first prompt by asking the user's name, after that point the conversation cycles, and the reins of the conversation often change hands.

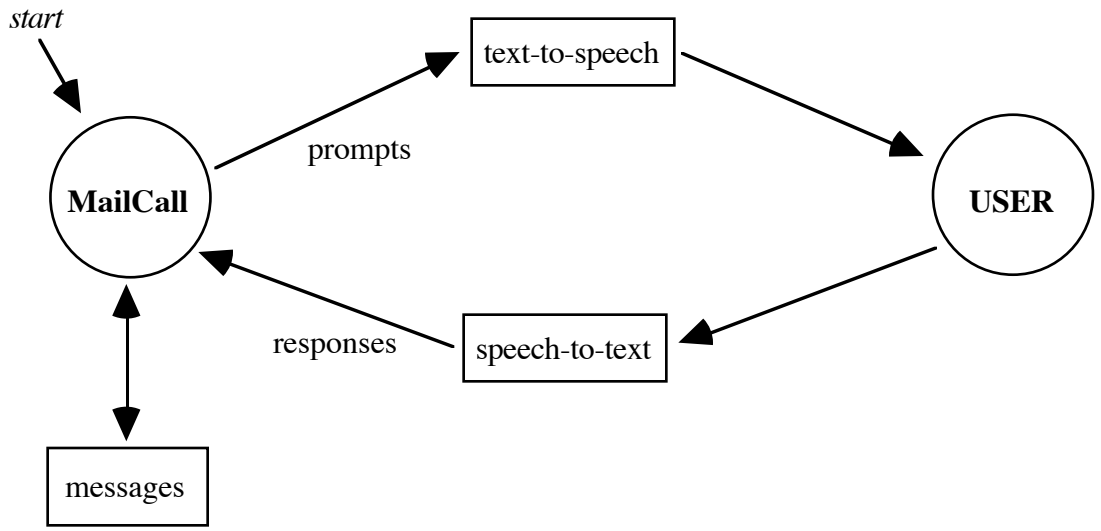

Figure 3.1: The interaction loop of MailCall.

3.2.2 Combining speech and touch-tones

Although touch-tones have many limitations, they can complement or even supersede speech depending on the task and the context [Stifelman 1992]. In a noisy environment, for instance, DTMF detection is likely far more accurate than speech recognition. It is more private, as well; one may prefer typing a password instead of broadcasting it by voice. In addition, the user can issue a series of repetitive commands more quickly with touch-tones, as when skipping over several messages in a row. Touch-tones may be the only method of interruption if barge-in is not supported (as is the case over the analog line), and even if barge-in is supported, some users may feel obligated to wait until the system is finished speaking before interrupting with voice [Stifelman 1992]. Touch-tones are also valuable for commands which cannot be executed quickly with voice, such as skipping back and forth among the text of a sentence. Saying "next sentence" and waiting for the recognizer to return an event might take a couple of seconds whereas a touch-tone can accomplish the command instantly.

The layout of the telephone keypad in MailCall resembles that of Phoneshell. When MailCall hears a touch-tone, it simulates the recognition of that command. (See Section 7.2 for a discussion of the event-driven architecture of MailCall.)

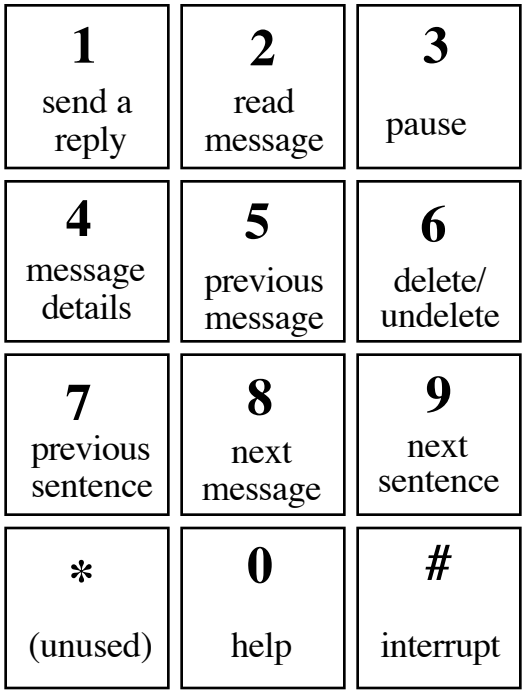

Figure 3.2: The MailCall DTMF equivalents.

3.2.3 An "implicit" discourse model

Given that recognition vocabularies are typically small, modeling human-computer conversation with complex discourse models is somewhat akin to driving in a thumbtack with a sledgehammer. Several linguistic discourse theories are designed to model complex interactions where the participants generally understand each other. Human-computer conversation, by contrast, is usually quite limited in scope but full of misunderstandings in the form of recognition errors.

Unlike Chatter but like SpeechActs, MailCall does not use an explicit discourse model based on a linguistic theory.[5] Thanks to the Swiftus parser [Martin & Kehler 1994], utterances in the recognizer's vocabulary can be labeled with individual discourse segment tags so that, unlike Chatter, MailCall itself does not infer which discourse segment should process incoming utterances—it simply passes the parsed utterance to the appropriate routine for handling that discourse segment. MailCall does, however, keep track of the dialogue, including the referents of anaphora[6] (for resolution of commands like "give her a call"), the user's previous actions (for tailoring prompts, as described in Section 6.1.1), and the list of currently available options. MailCall's interaction can be described as a finite-state diagram where each state represents the user's options at a given point.

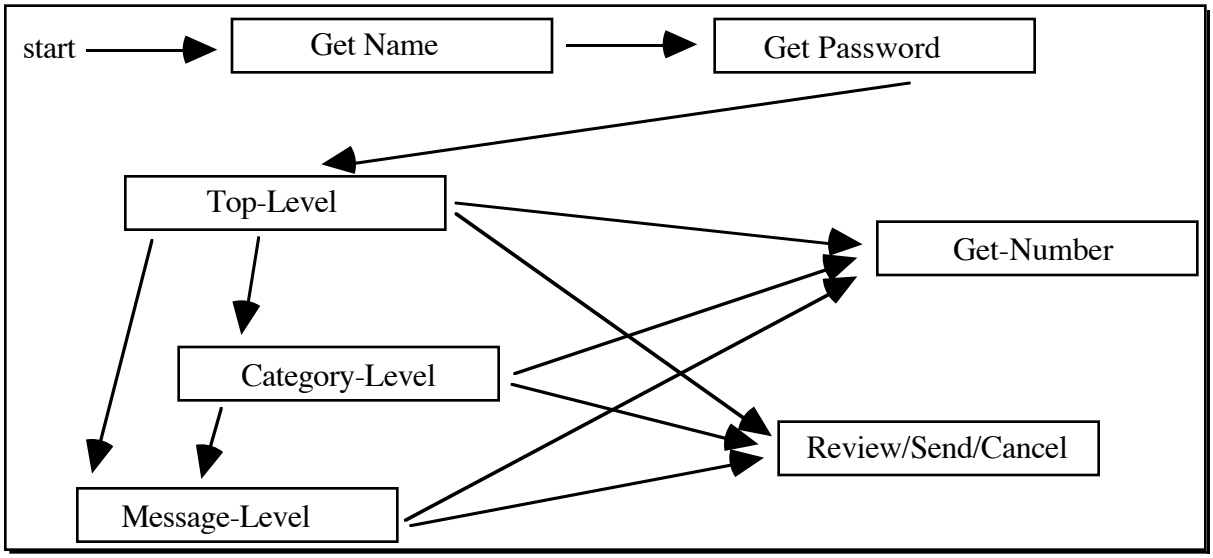

Figure 3.3: A finite-state diagram of MailCall interaction.

[5] In a sense, MailCall suggests what can be achieved without an explicit discourse model; its shortcomings will suggest what a stronger model might accomplish.

[6] Anaphora is a linguistic term for the use of pronouns such as *he, she,* and *it*.

This diagram describes the implicit dialogue states but not the transitions between those states. The transition between Get-Password and Top-Level, for instance, requires the user to say the correct password. To get from Message-Level to Review/Send/Cancel, on the other hand, the user has to say "send a message" and then record an outgoing message. Several commands, such as "next message" do not put the user into a new dialogue state; they would be represented as loops back to the same state. Other commands can lead to various dialogue states depending on the context, such as "read me the summary," which summarizes the messages in the current category (see Section 5.4). This either loops back to Category-Level or goes to Message-Level depending on the number of messages in the category, since if the summary reads "you have one message from Jordan Slott" the user might certainly expect to say "read it," "what's the subject" or "give him a call"—all message-level commands.

3.2.4 Vocabulary Subsetting

Knowledge of the user's options at a given time—as represented in the states of the diagram—is essential in order to activate the appropriate regions of the recognition grammar. For instance, when MailCall is asking the user's name, it need not listen for commands like "send a reply" or "go back to my timely messages." Doing so would needlessly increase the probability of recognition errors. Most speech recognizers including Dagger, the one used by MailCall, allow portions of the vocabulary to be deactivated so that MailCall only needs to process relevant utterances. The mechanics of activating partial regions of a grammar in this development environment are covered in [Ly 1993]. The following table represents what type of commands are available to the user at a certain state in the dialogue. (Note that the wording of the second column does not represent the extent of the vocabulary but is only descriptive of the kinds of input which are allowed.)

dialogue state	**available options (activated vocabulary)**
Get-Name (starting point of interaction)	first and last names of system users ask for help
Get-Password (once the user name has been given)	000000-999999 (six-digit password) ask for help
Top-Level (after the password has been given and the top-level summary has been read. if there are no messages, then the commands dealing with messages and categories are not activated, though the ones for calling and sending messages are)	start with the first message go to a message category ask for messages from a certain sender send someone in the rolodex a message call someone in the rolodex ask for help hang up

Category-Level (whenever the user goes to a category or moves past the end of a category, or after a summary is read)	read a summary of the category read msgs from a sender in the category start with the first message go to another message category ask for messages from a certain sender send someone in the rolodex a message call someone in the rolodex ask for help hang up
Message Level (after an action has been performed on a message, or when the user has asked to read the messages from a certain person)	read the current message delete the current message mark the current message as read ask the sender of the current message ask the subject of the current message ask when the current message arrived ask who the current message was sent to ask the priority of the current message send a reply to the current message call the sender of the current message send a message to someone in the rolodex call someone in the rolodex read the summary of the current category go to another message category ask for messages from a certain sender ask for help hang up
Get-Number (once the user says to call someone but MailCall can't find the phone number in the rolodex)	three digit area code, seven digit phone number ask for help hang up
Review-Send-Cancel (after a message has been composed)	review the message before sending it send the message don't send the message after all ask for help hang up

Figure 3.4: Vocabulary subsetting for each dialogue state.

Depending on the size of the user's rolodex, the active vocabulary can be massive—especially in the Message-Level state. MailCall deals with the inevitable recognition errors using an interface algorithm called ErrorLock described in Section 6.4.

3.3 A sample MailCall session

An annotated session with MailCall is provided below. Reading a printed dialogue cannot substitute for hearing a sample session, though, so the reader is encouraged to listen to one of the sound files found at

`http://groucho.www.media.mit.edu/people/groucho/mailcall.html`

The first step in a MailCall session is to identify the user. Since it is very important that unauthorized persons not have access to messages, MailCall requires a password. Once the user's identity is confirmed, MailCall changes its ownership to the user,[7] and a background process is launched to parse the spool file and filter the messages.

Parsing and sorting the user's messages is not instantaneous, so MailCall tries to reduce the perceived "dead time" while messages are parsed. A first step is to launch the background filtering process as soon as the user confirms that MailCall understood the correct name—not waiting until the password is spoken. This may seem to raise a security concern, but since the top-level summary is not read until the correct password is given, there is little danger of fraud. MailCall says "Hello, <user's first name>!" to quickly confirm that the user was heard, takes the half-second or so to launch the background process, and then asks "Please say your six-digit password." The background process is launched between the two prompts because if MailCall launched the process before any feedback was given, the user might fear that MailCall was stuck. The time spent prompting for the password, listening for the password, and letting the user know that the password correct gives the background process more time to finish its work. And if the user has to speak the password two or more times, then there is even more time to finish parsing. MailCall apologizes for any delays in parsing.

MailCall:	Welcome to MailCall. Who's calling?
User:	This is Matt Marx.
MailCall:	Matt Marx?
User:	Yes.
MailCall:	Hello Matt. Please say your six-digit password.
User:	Six four one nine seven zero.
MailCall:	I heard "six four one five seven zero," but that is not your password. Please say it again.
User:	Six four one *nine* seven zero.
MailCall:	Hold on while I get your messages.

Figure 3.5: Identifying the user by password.

When parsing and filtering are complete, MailCall reads a summary of the messages. After MailCall gives the user a summary of new messages, the user can ask if there are messages either from people in the rolodex or from those the user has recently sent messages or tried to call (see Section 4.2.3). For instance, if the user had recently called Bill O'Farrell or sent email to Ian Fox, then those names are added to the recognizer's vocabulary.

[7] In the Unix filesystem, a program (or *process*) which wishes to access a file must have permission to read it. Since mail spool files are typically readable and writable only by the user, the process must appear to be authorized by the user to do so. (And since the spool file can be on a different machine from the one running MailCall, it does not suffice to have root permission.) Thus the process makes a call to setpid() with the user's ID number as an argument. This only works if the process is originally owned by root.

MailCall:	Wait a second while I finish sorting your messages...OK! I found five timely messages, nine personal ones, and thirty-seven others. You can start with the first message, go to a category, or ask for a message from a specific person. You can ask for help... <user interrupts MailCall>
User:	Is there anything from Mike Phillips?
MailCall:	Sorry, no messages from Mike Phillips.
User:	Did Bill O'Farrell write back?
MailCall:	Nothing from Bill O'Farrell, either.
User:	What about Ian Fox?
MailCall:	Personal messages from Ian Fox. Message 8 from Ian Fox about "can we meet tomorrow?"
User:	Read it.
MailCall:	<reads the message>

Figure 3.6: Asking for messages from a certain user.

The user can also go to a message category. On request, MailCall reads a summary of the messages in that category. The summary is a clustered list of the people who have sent the user messages. Those names are then added to the speech recognizer's vocabulary so that the user can access them directly.

User:	Go to my timely messages.
MailCall:	Timely messages. Should I read you the summary, or should I start with the first message?
User:	Read me the summary.
MailCall:	You have a few from Nicole Yankelovich and one each from Jeff Herman and Rebecca Prendergast.
User:	Read me the one from Jeff Herman.
MailCall:	Short message four from Jeff Herman about "Zexel demo"
User:	Read it.
MailCall:	"Matt, you know that demo we're supposed to do tomorrow? I forget what time it is. Can you let me know? Thanks a bunch."

Figure 3.7: Hearing a summary and read messages from a specific sender.

The message from Jeff Herman is in the timely category because the subject line included the word "Zexel," with whom the user had a demo scheduled that same day. The system inferred that this message might be important. For more details on the CLUES filtering system, see chapter 4.

The user can send a reply to an incoming message, or compose a new message to anyone in the rolodex. Once the user's reply is recorded, it can be reviewed before sending. If the sender is not on the local voice mail system, the user is prompted for an encoding format which is stored and used the next time a message is sent to that person.

User:	Send a reply.
MailCall:	Record your voice message after the tone. Pause for several seconds when finished.
User:	*Jeff, I think the demo is at 3pm. I'm pretty sure it's in the usual conference room. Did you get the final demo schedule? If you need it let me know.*
MailCall:	Got it. Review the message, send it, or cancel?
User:	Review it.
MailCall:	*Jeff, I think the demo is at 3pm. I'm pretty sure it's in the usual conference room. Did you get the final demo schedule? If you need it let me know.* Review, send, or cancel?
User:	Send.
MailCall:	Send in which format? Sun, NextMail, Mime, or U U Encode?
User:	Mime.
MailCall:	Sending message in Mime format. I'll use that format next time you reply to Jeff Herman as well.

Figure 3.8: Sending a reply to an incoming message.

Often the recognizer misunderstands the user and would do the wrong thing if it didn't check first. By asking the user if it heard correctly, there's a chance to correct it quickly and go on. For more details on error-handling, see Section 6.4.

User:	Next message.
MailCall:	Message 5 from Christie Davidson about "free food in third floor kitchen."
User:	Delete that.
MailCall:	Did you say to delete it?
User:	Yes.
MailCall:	Message deleted.
User:	Um...go to my personal messages.
MailCall:	Did you say to hang up?
User:	No, I said go to my personal messages.

Figure 3.9: Handling substitution errors.

Equally frustrating is when the system doesn't understand the user. This happens most often when the user is speaking outside the system's vocabulary. MailCall tries to guide the user towards saying something it understands with a combination of re-prompting, speaking tips, and context-sensitive help.

MailCall:	Personal messages. Read a summary, or read the first header?
User:	Why don't you read me the very first message of the group?
MailCall:	Excuse me?
User:	Read -- me -- the -- very -- first -- message -- of -- the -- group.
MailCall:	I don't understand. Please rephrase.
User:	Ummmm....read -- the -- first -- message.

MailCall:	I'm having trouble understanding you. Speak clearly, but don't overemphasize. That is, don't -- talk -- like -- this.[8]
User:	Well, I guess I just don't know what to do here.
MailCall:	Still no luck. Try saying "help."
User:	Help!
MailCall:	You just moved to your personal messages. You can ask for a summary of the messages in this category by saying "read me the summary." You can start with the first message in this category by saying "start with the first one." You can go to another message category; for instance, "go to my personal messages." If you get really confused, you can say "start over from the beginning."
User:	Start with the first one.

Figure 3.10: Handling consecutive rejection errors and giving help.

Instead of sending voice replies, the user may elect to call someone who sent a message. The number is looked up in the user's rolodex.

MailCall:	Message 6 from Sung Han Nam about "party tonight."
User:	What's it about?
MailCall:	It's about "party tonight."
User:	Give him a call.
MailCall:	Did you say to call Sun Han Nam?
User:	That's correct.
MailCall:	Should I use the home or work number?
User:	Use the home number.
MailCall:	OK, dialing Sung Nam at home. **<conversation ensues>**

Figure 3.11: Voice dialing.

The user is free to hang up at any time during the session. Since hanging up is an irreversible action, MailCall checks with the user before doing so.

User:	That's all for now.
MailCall:	Did you say to hang up?
User:	Yes.
MailCall:	Matt, thanks for calling. Talk to you later.

Figure 3.12: Terminating a MailCall session.

This chapter has given an overview of MailCall. The next three chapters focus on the highlights of the system outlined in the introduction: filtering, navigation, and error handling. Design choices are described as well as implementation.

[8] MailCall is not actually detecting hyperarticulation; it just figures that after a string of misrecognitions it might be the case that the user is speaking as one would to a dictation system, with exaggerated pauses between words.

4. Message Filtering and Prioritization

This chapter describes CLUES, a filtering and prioritization system which exploits the richness of the user's desktop information environment. By surveying the user's calendar, rolodex, to-do list, as well as the record of outgoing messages and phone calls, CLUES can mark messages as "timely" without requiring any extra effort on the part of the user. Further, since the approach is not tied to any single application, the benefits of filtering are available at the desktop, over the phone, or through whatever medium the user chooses. Greeting the user with a list of messages ordered by relevance facilitates more effective message processing.

4.1 Capturing short-term or "timely" interests

Both rule-based systems and memory-based-reasoning have been employed to mark a user's important messages, but each has its own set of drawbacks which suggest the need for a method as powerful as the former yet as effortless as the latter.

4.1.1 The cost of authoring rules

Several systems, both research projects and commercial offerings, accomplish filtering by matching user-authored rules against the header or body of an email message. Procmail [Van den Berg 1993], for instance, allows the user to compose a set of rules: if I add a rule "anything from 'bestfriend@my.alma.mater.edu' is important", it takes effect immediately. A sample set of rules is described below:

```
ORDER=veryimportant:important:timely:personal:other
^From.*geek | set veryimportant
^From.*speech-group | set important
^To.*groucho | set personal
^.* | set other
```

Figure 4.1: A set of user-authored filtering rules.

In this example, "ORDER" is a prioritized list of categories into which messages can be filtered. Each successive line lists a rule consisting of a condition (using regular expressions) and an action. For instance, if the header line of a message contains "geek" the message is set to the category "very important." If none of the first three rules match, the fourth and default rule is invoked and the message is labeled "other."

The power of rules does not come without cost—most obviously, the time and effort required to write and maintain a list of rules.[9] Since no filtering can be performed until the user writes the rules, the cost precedes the benefit. Yet the cost of writing a rule may outweigh its benefit, especially in the case of short-term interests: for instance, it may be too much of a hassle to write a rule for a one-time visit, especially if the user is expecting only one or two messages about the visit. Further, when the usefulness of the rules has expired, the user must then remove them from the list lest it grow unmanageable. Finally, in writing rules one is often duplicating information found elsewhere; in the above situation, the user's calendar probably contains an entry "Friday 2pm: Microsoft demo." Besides wasting time, duplicating information can lead to problems of synchronization; if the Microsoft demo is canceled and replaced by one with Compaq, the filtering rules must be updated accordingly.

4.1.2 The difficulty of capturing short-term interests with MBR

One approach is to dispense with rules in favor of an autonomous filtering system, as is the approach of memory-based reasoning (MBR) as implemented by [Metral 1992] and [Ly 1993]. Instead of filtering messages on the basis of explicit rules, messages are prioritized based on the user's history of past actions regarding similar messages. Hence, if a user tends to read and save messages from Lisa Stifleman but always deletes messages from William Browning, then it will "learn" to put messages from Lisa at the top of the list and the ones from William at the bottom.

Although the learning-based approach is appealing since it involves no user effort, it forfeits some advantages of rules. Most egregious is the ramp-up involved in developing a user model; thus the system may not be effective at filtering for some time. This is actually an extreme liability when dealing with short-term interests like the above example of the Microsoft demo. By the time the system has "learned" that Microsoft is important, the usefulness of knowing that fact may have passed; in fact, Microsoft may not be interesting anymore.

4.1.3 A low-effort, high-benefit approach: CLUES

Both in the case of user-authored rules and MBR, long-term interests can be captured (whether by user effort or machine learning), but short-term interests are difficult to handle since the user may not be willing to constantly maintain rules, and since MBR

[9] The arcane syntax of regular expressions, the most common representation for such rules, can also be a barrier, though some GUIs such as Eudora [Dorner 1992] ameliorate the difficulty.

may not catch on quickly enough. Yet it is important to capture short-term interests, for they may be very important despite their temporary nature. Finding an update to a demo schedule, hearing the response to a message you've sent to a conference chair asking if your submission was received, or catching a response from someone you tried to call earlier in the day can be crucial to staying in touch and on schedule. Thus the challenge is to anticipate short-term interests and prioritize incoming messages accordingly.

Fortunately, a typical user work environment is replete with information which can help pinpoint user interests. The Media Lab speech group has an information environment which, though rich, is not terribly unusual; it boasts a calendar, to-do list, rolodex, and a record of outgoing messages and phone calls—and as such serves as a fertile testbed for this approach. Not every environment will contain all of these tools, and the information from certain tools may not be accessible due to proprietary formats or encryption, but the power of this technique may encourage the use of "open" data formats.

The process for automatically gathering this information from the user's environment is called CLUES. Regular-expression rules (invisible to the user) are automatically generated and used to filter and prioritize incoming messages as well as a set of user-authored rules for longer-term interests. This, along with the integration of user-authored rules (to capture long-term interests), is described below. MBR has not been integrated but is a natural next step; one can imagine it being especially useful for ranking messages within a category by relevance.

4.2 Gathering clues from information sources

The calendar and to-do list are both sources of timely information. Each calendar entry is indexed by a date, so a "window of interest" is set within which calendar entries are considered. Since a to-do list is by its nature up-to-date, CLUES can assume that its entries are always relevant. The rolodex, though not a source of temporal data, can be beneficial if correlated with the calendar or to-do list. The record of outgoing messages and phone calls may be relevant back to a certain point in time.

4.2.1 Extracting clues from the calendar

Establishing user interests via the calendar involves selecting the relevant entries and then extracting meaningful items from those entries. Assuming that future events will be more relevant than past ones, CLUES biases the aforementioned interest window

toward the future by a factor of four. That is, it might look a month in the future and a week in the past, or a week in the future and a couple of days in the past. CLUES must remove useless data and preserve meaningful information so that it can have the greatest number of appropriate clues and a minimum of false hits.

Figure 4.2: Sample calendar entries.

One useless piece of information to be extracted is a time. Standard Unix calendar entries are unconstrained, and since the Unix calendar is universal CLUES should support it. A user might write the entry for a certain afternoon meeting in any of the following ways:

```
3:30-4:00 p.m. meet with John
3:30 - 4pm meet with John
3:30- 4P.M. meet with John
3:30pm -4 meet with John
3:30 - 4:00 P.M. meet with John
3:30 P.M. - 4:00 P.M. meet with John
3:30-4 meet with John
...and so on...
```

Left with calendar entries stripped of times, more than one important piece of information may be contained on a single line. For instance, with an entry:

```
annual planning meeting w/ John Malinowski, Deb Cohen
```

each individual named there is important, so CLUES must separate out the names. The same applies for parenthetical expressions, words or phrases offset in quotes, etc. Treating particular words and symbols as delimiters, CLUES distill the entries into individual units of meaning. The resulting clues are:

```
annual planning meeting
John Malinowski
Deb Cohen
```

CLUES produces a list of all the unique words and proper noun phrases contained in the calendar. (The same process is applied to the to-do list, although there is no need to remove times or worry about timeliness.) In order to avoid spurious matches, CLUES removes *stop words,* commonly used words with little semantic value (e.g., "the," "at"), using lists from AppleSearch and SMART [Salton 1981].

4.2.2 Correlating with the rolodex

Another source of information in the calendar is found in area codes. Business travelers often leave behind a phone or fax number where they can be reached, and the area code of that number gives clues to where they are and who is geographically close to them. When CLUES detects an area code in a relevant calendar entry, CLUES checks the rolodex for people who share that area code. (CLUES also takes advantage of a small, user-independent knowledge base which represents proximity of area codes to each other. For instance, if it finds an area code 415, the neighboring 408 is considered, too.)

The assumption here is that while traveling, messages from people close to where you are may take on special importance. For instance, if you are heading out to Interval Research in Palo Alto for the week, friends or colleagues in the area may be trying to contact you in order to set up meetings while there.[10]

When CLUES matches an area code from the calendar with one in the rolodex, it notes the person's name, email address, and the domain from which their email originated. Certainly one wants to be alerted about email coming from them directly, and one may want to pay special attention to messages originating from their site. So even if I'm going to visit Sean at Interval, it's likely that anything coming from Interval will be interesting this week.

[10] A trivial but useful spinoff of CLUES is a program who_should_i_visit which runs through the user's calendar and rolodex, picking out the people who live in and near the area codes of the fax numbers contained in the calendar. No more does the user arrive home from a trip saying, "Oh yeah, I should have visited..."

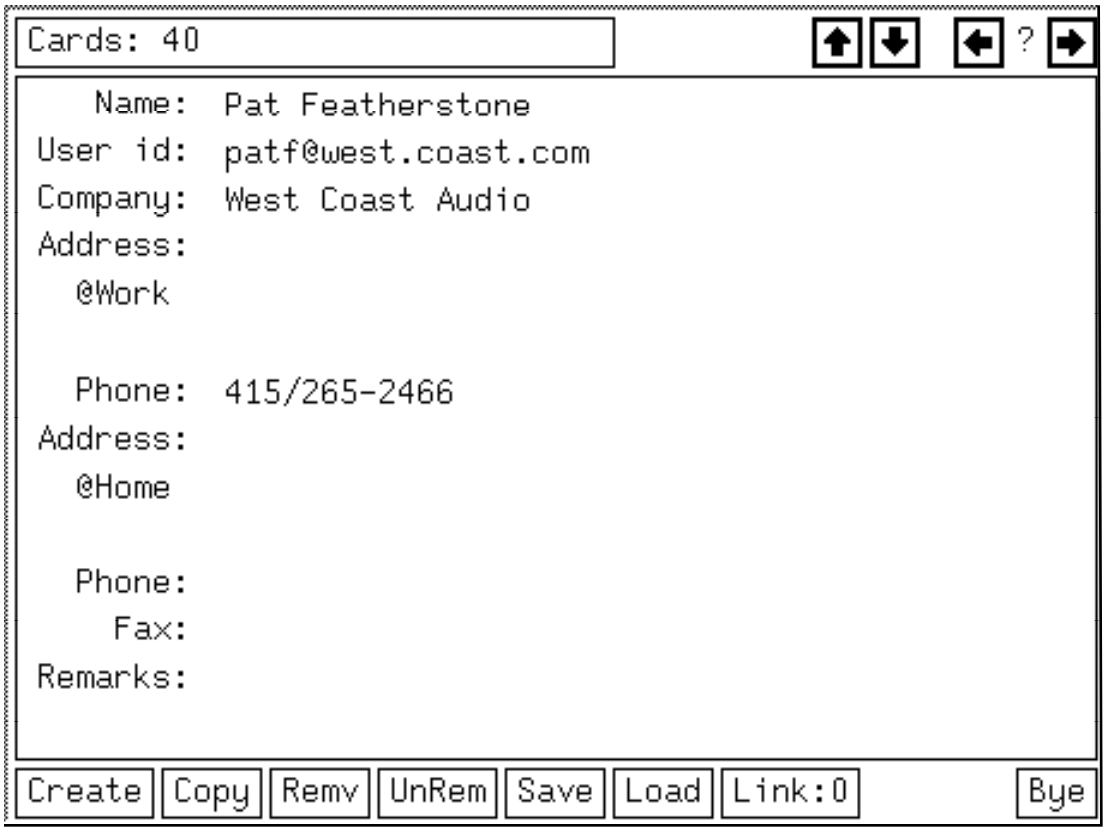

Figure 4.3: A sample rolodex card.

Not all domains are equally relevant, however. Many people send email from different machines all the time. I may have nicoley@east.sun.com in my rolodex yet receive mail from nicoley@argus.east.sun.com. Thus CLUES must isolate exactly the relevant portions of the domain. To do this, it makes an assumption about domain names and then allow exceptions: the assumption is that most Internet nodes are geocentric -- that is, that they reside in one location. This is true for many companies (interval.com is in Palo Alto) and most universities (mit.edu is in Boston). Some companies, however, are geographically distributed: Sun has offices in California and Massachusetts, and IBM has offices all over the globe.. In these cases, CLUES notes the company or organization as an exception, and pays attention to the next piece of the domain. For instance, "eng.sun.com" is the west coast and "east.sun.com" the east coast, so paying attention to more than just the main address can help filter out irrelevant messages. Further, for some hosts such as America Online and Compuserve geography is meaningless; CLUES ignores these completely, since it is false to assume that because I will be going to visit

my in-laws in Utah, who happen to be on America Online, that all messages originating at aol.com are suddenly important.

4.2.3 Examining outgoing messages and phone calls

Hearing back from someone you have been trying to contact is often important. By inspecting the record of the phone calls and outgoing messages, CLUES does a reasonable job of deciding which of the user's incoming messages constitute replies.

Of course the user may not be interested in the response to every single message sent. Thus several databases are maintained (in the form of log files), each of a particular priority. A copy of an outgoing message can be directed to one of several log files depending on its importance. The log files are specified in a configuration file, and following each file is a number representing the number of weekdays for which messages from that file are relevant. Thus the entry

```
sent mail file: /u/groucho/.filterstore/SEEKREPLY 15 days
```

means that replies to messages in /u/groucho/.filterstore/SEEKREPLY will be marked as timely for the next 15 weekdays. Similarly, the entry

```
sent mail file: /u/groucho/.filterstore/SENT 1 day
```

means that replies to messages in /u/groucho/.filterstore/SENT will be marked as timely only if they arrive within a day after the original message was sent.

The current mechanism for adding a message to one of the databases depends on the modality. If one is using a screen-based mail reader such as Elm, Mail, or Eudora, one can send a copy of the message to one of the log files specified in the configuration file.[11] And if the user is not at all interested in a reply to the message, it can simply be sent without redirecting a copy to any of the log files. Obviously, this scheme is rather cumbersome. A better approach would be to construct a mailer which allowed the user to specify an outgoing message's priority and have it record the files itself.

When sending a message from one of the several telephone-based interfaces available (Phoneshell, Chatter, MailCall), a copy of the message is kept in a specific log file. How

[11] Not wanting to modify the source code of the existing mailers, and seeking to alleviate the tedium of typing a long filename into the Cc: field of a message, a system of mailing aliases can be set up. Thus mail cc'ed to "groucho-outgoing" might be directed to /u/groucho/.filterstore/SENT, whereas "groucho-important" might place the message into /u/groucho/.filterstore/SEEKREPLY.

the user chooses to treat the messages sent over the telephone can be specified in the configuration file. For instance,

```
sent mail file: /u/groucho/.filterstore/PHONE 5 days
```

will cause responses to messages sent over the phone to be marked timely for five weekdays after their delivery.

Given these databases, CLUES attempts to decide which of the user's incoming messages are replies. Determining what is a response is an inexact science, since a reply to a message may take one of several forms or formats. It does not suffice to look for "Re:" in the subject line for several reasons. First, mailers use different schemes to mark replies; "Re:", "Re[1-n]:", and "-Reply" are common, though new mailers may not choose to follow these conventions. Second, many messages with those markers in the subject lines may not be replies to the user, especially if one subscribes to several mailing lists or often receives mail forwarded by other people. Third, someone responding may choose to initiate a new message without indicating in the subject line that it is a response. CLUES uses other heuristics which, though imperfect, should catch more actual replies than false hits.

The heuristic for finding replies is based on the types of replies one might expect to find for different messages. One may send a message to a specific person and expect that whatever that person sends back will be relevant, (this may most likely be true with people the user rarely corresponds with). One may send a message to someone with whom one corresponds frequently, in which case matching the subject line may be important to distinguish what is a reply to that specific message. One may send a message to a mailing list, in which case only the subject line can help in finding replies. Since its heuristic is to match against the sender or subject line, CLUES will catch replies in the case that an individual replies to a message, whether or not the same subject line is used, or in the case that an individual other than whom one sent the message replies but with the same subject line. (Because of the latter case CLUES ignores common subject lines like "Re: your mail".)

It is also clear that CLUES will miss some replies and have some false hits. For instance, it will always miss replies from random people with subject lines different from the ones which were sent out. False hits will also be generated, usually in the case of receiving many messages from one person when one is only interested in a specific subject. Here, all the messages from that sender will be marked as timely, which may be undesirable.

Despite the heuristic nature of the matching, this technique can prove very powerful by freeing the user of the pain of writing a rule for each important outgoing message. For instance, if one sends a message to a conference committee asking about the status of a recent submission, one can tag the outgoing message and then have the response be prioritized when it arrives.

```
Date: Thu Jan 0 1995 13:45:00
From: multimedia96@important.conference.org
Subject: yes, your submission has been received
```

Similarly, the responses to an inquiry sent to a mailing list can be marked as timely, even if the user cannot anticipate who might be replying.

```
Date: Thu Jan 0 1995 13:45:00
From: nobody@you.know.net
Subject: Re: anyone know a good mechanic in Cambridge?

Date: Thu Jan 0 1995 13:45:00
From: some-guy@on.the.list.net
Subject: Re: anyone know a good mechanic in Cambridge?
```

The same strategy allows responses to phone calls the user has made to be prioritized, whether the response arrives as email or voice mail. When a user makes a phone call using the desktop Phonetool utility [Schmandt 1987], Phoneshell, or MailCall, an entry is added to a log file containing the date, the name of the person called, the email address (if available), and the phone number called. A sample entry:

```
Date: Thu Jan 9 1995 13:45:00
Name: Chris Andrews
Email: chrisa@fish.net
Phone: (617) 253-9837
```

CLUES scouts through the file, picking out the names, email addresses, and phone numbers from the entries within the last X days specified by the user. Since the speech group voice mail system sends email notification of incoming voice mail, and since CLUES uses that email message as a pointer to the associated voice mail message, CLUES prioritizes voice mail by prioritizing the associated email notification. A sample notification message looks like this:

```
From root@media.mit.edu Thu Jan 9 1995 13:45:00
To: groucho@media
Subject: Voice message from 253-9837
From: Operator <root@media.mit.edu>

You received a 40 second voice message from 3-9837.
On a SPARCstation you can listen to your message with “cat
/sound/spool/soundfiles/groucho/vmail.9003525 > /dev/audio” or use
the graphical or telephone interface.
```

Thus CLUES is able to mark voice or text replies as timely, strengthening the claim of unified messaging. Since caller ID is available only within the MIT PBX, the number is not available for those calling from outside MIT. Still, the concept is in practice here, and with more complete caller ID one could extend this functionality to a greater percentage of the user's voice messages. [12]

4.3 Generating rules from clues

Once CLUES has gathered clue words and phrases from databases it creates rules to be used in the actual filtering process. Since it is not intended that the user modify the rules, readability is not a concern.

4.3.1 Names

Many of the clues gathered from the user's databases will be names. CLUES assumes (with some risk) that a name "Matt Marx" will be safe from false hits since it is unlikely to show up at random or as part of another string or words. Thus the rules for matching names or proper noun phrases are very simple. If, for instance, the user is looking for messages from Brady Mickelsen, the rule will be as simple as

```
^From:.*Brady Mickelsen
```

The rule reads as follows: "starting at the beginning of the line (the carat '^'), look for the word "From" followed by a colon, and then any string of characters or spaces (wildcard '.' followed by Kleene star '*') followed by the name "Brady Mickelsen". Or, in simpler terms, "look for Brady Mickelsen anywhere on a "From:" line. If CLUES knows that the element is a name—which it can, if it was collected from the database of outgoing messages or phone calls, or if it is extracted from the rolodex—then it suffices to write a "From:" rule for the name. This rule would be useful in finding the following message from Brady:

```
From milhouse@leland.stanford.edu Thu Jan 9 15:33:01 1995
From: Brady Mickelsen <milhouse@leland.stanford.edu >
Subject: Re: coming to California?
```

4.3.2 Proper Noun Phrases

There may be cases, however, where CLUES cannot tell if a proper noun phrases is a person's name or not, particularly in the case where it is extracted from the calendar or

[12] One possible feature not implemented in the group's voice mail system is changing the name of the sender from "Operator" to a person's name based on matching the identified phone number with the user's rolodex.

to-do list. A proper noun phrase from either of those sources could easily be a company or event name which might appear in the “Subject” line, as well. In this case CLUES generates a pair of rules to cover both the case where the entry happens to be the name of a person who might be sending email, and the one where the entry is a more general term for an event or something which might be the subject of the message. Thus the clues “John Linn” and “Texas Instruments” would generate the following rules, respectively:

```
From:.*John Linn
Subject:.*John Linn

From:.*Texas Instruments
Subject:.*Texas Instruments
```

In the first set of rules, it is likely that John Linn is a person from whom the user is expecting a message. It is also possible that he might be a visitor about whom some messages would be sent around. In the second set of rules, it’s unlikely that a message will ever arrive with “Texas Instruments” as the sender, but it is entirely possible that “Texas Instruments” might appear in the subject line of a message—if, for instance, the final schedule for their visit is being sent around. (Of course CLUES can’t tell if a multi-word proper noun phrase is a name or not—even an appeal to a database of names will likely fall far short—so it generates both rules.) Such a rule might be useful in matching the following header:

```
From prender@media.mit.edu Thu Jan 9 15:33:01 1995
To: speech-group@media.mit.edu
From: Rebecca Prendergast <prender@media.mit.edu>
Subject: Final Agenda for Texas Instruments visit
```

4.3.3 Individual Words

Rules for individual words are trickier than for people’s names and multi-word phrases since they present a greater likelihood of false hits: the abbreviation “TI” (for Texas Instruments), when matched case-insensitively, could falsely identify words like “situa**ti**on”; if we’re looking for the word “workstation” we’re probably interested in its plural, “**workstation**s”, as well. The goal is to match the word, whether in a pluralized or possessive form. Thus if the calendar shows "meeting with Brian", and an hour before the meeting a message arrives saying "Brian's visit canceled" that should match. The same goes for "Important--Motorola visit update"; CLUES should not miss the match because of the dashes. A regular expression which handles all of the above cases is the following:

```
^Subject.*[^a-zA-Z0-9]+Motorola((([^a-zA-Z0-9]|es|s|'?s)+.*)|([^a-
zA-Z0-9]|es|s|'?s)*$)).*
```

This will match "Motorola" as long as it is either at the beginning or end of the line, surrounded by whitespace or non-alphabetic characters, or in its plural or possessive form. In addition to finding the word in amongst punctuation, this regular expression prevents the word from being "found" as a substring of another word. Thus

```
^Subject.*[^a-zA-Z0-9]+TI((([^a-zA-Z0-9]|es|s|'?s)+.*)|([^a-zA-Z0-
9]|es|s|'?s)*$)).*
```

will match this header:

```
From prender@media.mit.edu Thu Jan 9 15:33:01 1995
To: speech-group@media.mit.edu
From: Rebecca Prendergast <prender@media.mit.edu>
Subject: TI visit tomorrow 3/15 3:30pm
```

but not this one:

```
From dnb@media.mit.edu Thu Jan 9 15:33:01 1995
To: all@media.mit.edu
Subject: networking situation
```

whereas a simple rule "Subject:.*TI" would match falsely on the word "situation." Now, it is possible that a single-word clue could be part of an email address in which one is interested. For example, the clue "Supersoft" might indicate that messages originating from the domain "supersoft.com" are interesting. Thus CLUES generates a "From" line rule, using similar though not identical syntax. In the TI example,

```
^From.*[^a-zA-Z0-9]+TI([^a-zA-Z0-9]+)
```

would be generated and would catch all messages originating from Texas Instruments:

```
From hemphill@csc.ti.com Thu Jan 9 15:33:01 1995
From: Charles Hemphill <hemphill@csc.ti.com >
Subject: what we'd like to see in Monday's demo
```

4.3.4 Email addresses

Matching messages with email addresses requires the generation of only very simple rules. The structure of an email header necessitates only checking two lines, the first "From" line which is of the structure

```
From [address] [day] [month] [date] [hour] [minute] [second]
[year]
```

(this is the first line of every email header and is used to indicate the beginning of a new message in a Unix spool file) and the second "From" line, which can be of the form

```
From: [sender@host]
From: [firstname] [lastname] <[sender@host]>
From: [sender@host] ([firstname] [lastname])
```

```
From: [sender@host] “[firstname] [lastname]”
From: [sender@host] (“[firstname] [lastname]”)
```

or not appear at all. (Of course, new mailers may invent new formats for “From” lines.) Although it might appear that looking in the second “From” line is overkill given the existence of the first one, in actuality the [sender@host] element of the first “From” line is sometimes nothing more than a generic “daemon@mailserver”, so CLUES must look at both “From” lines to be sure. In any case, if the address is to be found, it will be as simple as looking for its appearance as a unit in one of these header lines.

In looking for an email address (or, in the case of rolodex-generated rules, a domain name), CLUES can use a very simple rule to search for the appropriate item. In fact, it can collapse the rules for the first “From” line and the second into a more general regular expression, thus accelerating the search since each From line needs to be tested against one rule instead of two. For example, if CLUES wanted to find the messages from the email address “friend@high.school.edu”, the rule syntax would be

```
^From( |:.*)friend@high.school.edu
```

Thus the first “From” line, where the address appears exactly one space after the word “From”, is covered, as is the second, where “From: ” begins the line, and the address may lie in a variety of places. It is worth noting again here that the email addresses collected from the various databases are “pruned” for generality. Thus the above rule would probably read

```
^From( |:.*)friend@.*school.edu
```

in order to have a better chance of matching messages sent from another host at “school.edu”. (And if “high” was a significant part of the domain, as with companies which are geographically distributed, it would be preceded by the wild card ‘.*’ to match mail from any hosts within that subnet.)

4.3.5 Subject lines

Matching relevant subject lines is similar to matching email addresses, since CLUES is looking for a specific sequence of words in the “Subject” header line. The only precaution is to allow for the common methods for mailers to indicate that a message is a reply, which usually involves adding one or more occurrences of “Re:” to the front of the line or of “ -Reply” to its end. Thus for the subject line “how about dinner after the AT&T demo?”, the rule is simply

```
^Subject: (Re: )*how about dinner after the AT&T demo?( -Reply)*\$
```

Note that the Kleene star after the two parenthesized expressions allow multiple instances of each, as can result in a long series of replies with someone whose mailer tacks on extra elements. This approach avoids mismatches when a very short email header shows up as part of another: e.g., an outgoing message titled "thanks" is matched with an incoming message titled "thanks to everyone for six great years at the Media Lab!"

4.3.6 Phone Numbers

The clues generated from the log file of phone calls the user has placed contain phone numbers which can be useful in determining a message's relevance. As described above, the speech group voice mail system provides email notification of incoming voice messages, including the name of the caller on the subject line. Thus when CLUES finds a phone number, it generates a rule to match the subject line of an email notification of voice mail from that same caller.

```
^Subject: Voice message from 225-1535
```

The slight complication is that since caller ID is only available within the MIT PBX, it is a waste of time to generate rules for numbers outside of MIT. Rules are only generated for numbers within the 617 area code which start with 25 or 22. Should caller ID expand to cover more than just the MIT PBX, expanding the rules to cover all phone numbers is as easy as removing a few lines of code.

4.4 Setup and configurability

The clue gathering and rule generation is accomplished through a Perl script which calls other Perl scripts to scan the available databases for clues and then generates the filtering rules. There are about 20 individual scripts brought into collaboration by the master script, CLUES. In order to keep the rules current, the user can run CLUES as a cron job[13]. Updating every hour is recommended in order to catch changes to the databases.[14]

[13] Cron is a Unix process which executes at regular intervals—for instance, once each day or once an hour.
[14] A callback mechanism to let the user know when the databases had changed [Ly 1993a] would improve CLUES' efficiency and accuracy.

4.4.1 Minimizing computational requirements

Since CLUES may be run at short intervals, it is important to minimize the drain on system resources. CLUES accomplishes this with a system of time-stamping and backup files. When the script is run, it scans through the calendar, to-do list, rolodex, outgoing email log(s), and phone log in sequence. CLUES does not regenerate the clues for a particular database if it has not been modified since the clues were last generated. Then once the clues are generated, CLUES checks to see if they are different at all from the original set (since a change to the database may not necessarily produce a change in the relevant clues). If not, then the rules are not generated for that database. When all databases have been considered, the rules for the several databases are concatenated into a single file and then compared against the set of rules from the previous session. Only if there are differences is the old set replaced with the new one. And since the rule set is only changed during the one copy command rather than being erased at the beginning of the session, the chance of not finding the list is insignificant. One exception: if the configuration file has been changed since the last set of rules was generated, then CLUES redoes each set of rules for each database. (Of course if the end result is the same, CLUES doesn't copy over the new set of rules.)

4.4.2 Customizing the user's CLUES

A configuration file allows one to specify the components of one's work environment which should be inspected by the CLUES script. One may choose whether or not to use the to-do list, whether or not to correlate the calendar with the rolodex, where to save the record of outgoing voice messages and phone calls, and which databases to use in gathering clues about with whom one has recently corresponded. Since the richness of the information environment may vary from system to system, such configurability is crucial. In addition, CLUES performs error-checking so that if it can't find a rolodex file it prints out an error message and continues with its work rather than giving up. Thus without modifying the configuration file, any Unix user should be able to use the CLUES script without modification, assuming that Perl [Wall & Schwartz 1992] is installed.

4.5 Integrating with static rules

The actual matching of messages against filtering rules takes place inside the interactive system. The user's mail spool file is parsed into individual messages stored in temporary files, with the header information stored in a linked list structure. Then each message is run through a series of filtering/prioritization tests which determine its priority.

As hinted at above, CLUES does not replace user-authored rules but complements them. User-authored rules can be useful for capturing general or long-term interests where the user may feel the investment of writing a special rule worthwhile. Thus this system integrates user-authored rules with those generated by CLUES. At runtime, both lists of rules are read into a linked list structure. The process then goes as follows:

```
1. run the message through the user-authored rules. if it is
marked for deletion or as something not to be presented over the
phone (as one may choose to do for mailing lists), or if it is
already placed in the highest category, return the value and do
not use the CLUES rules (thus saving time, as regular-expression
comparison can be very slow).

2. if the message did not match any of the user-authored rules, or
if the priority assigned the message by the user-authored rules is
lower than what the message would be assigned if it were to match
one of the automatic rules, then run it through the set of
automatically generated rules.

3. return the higher of the two priorities.
```

The actual matching is done by running the entire header through the user-authored rules and the "From" and "Subject" lines through the automatically generated ones (since those are the only two header lines for which rules are automatically generated). The system does a case-insensitive regular expression match on each one and returns TRUE if a match is found. If a match is not found, the system runs through all of the available rules. Since the process is slow, CLUES stops after the first rule has been found instead of exhausting all of the rules to see which ones matched.

If a match is found by the automatically generated rules, the message is placed into a category specified in the user's configuration file, which must be one of the categories defined in the list of user-authored rules described in the beginning of this section. In the current scheme, all of the automatically generated rules are equally weighted, so that it does not matter which rule matches first.

The user has some freedom in specifying which rules to use. Filtering may be turned off altogether, or the user may use handwritten rules without the automatically generated ones. Within those, one can choose whether to use the rules generated from all of the non-useless words found in the calendar or whether to stick with proper phrases alone.

4.6 Application-independence of CLUES

As stated above, CLUES' gathering of information about the user is independent of any specific application since the database it builds is simply a set of text files. The next two sections describe two applications besides MailCall which take advantage of the rules generated by CLUES.

4.6.1 Phoneshell

Phoneshell is a touch-tone system for accessing personal information over the phone described in Section 2.3.2. The filtering provided by CLUES was integrated into Phoneshell before MailCall was even constructed and had been in use for several months at the time of this writing.

The use of CLUES in Phoneshell demonstrates that the approach is application-independent. Phoneshell, however, is a telephone-based service like MailCall. A desktop application which uses CLUES is described in the next section.

4.6.2 HTMaiL

MailCall can be run in a mode such that once all of the priorities have been assigned, an HTML file is written containing a list of the message headers grouped by category. Any HTML browser can thus be used to access one's messages, and if the browser is properly equipped to handle audio files, this method of access, called HTMaiL can serve as a unified desktop message retrieval system.[15]

One can click on a header line to go to the text of the message. Since the mail spool was parsed into a series of individual messages in preparation for the filtering process, this is as simple as making each header line a hyperlink to the file corresponding with that particular email message. Whether or not a message has been read is evident from the status of the link; most HTML browsers change the color or other display attributes of a link once it has been followed. One limitation, however, is that one cannot distinguish between messages which have newly arrived and those which are old but which have not been read, as is possible with full-fledged mailers.

[15] Note that there is no special reason to use HTML or the World-Wide Web for this task. (If anything, the user probably does not wish for the list of messages to be world-readable!).The GUI provided by the browser, as well as the simplicity of creating an HTML file with the appropriate links drove this implementation.

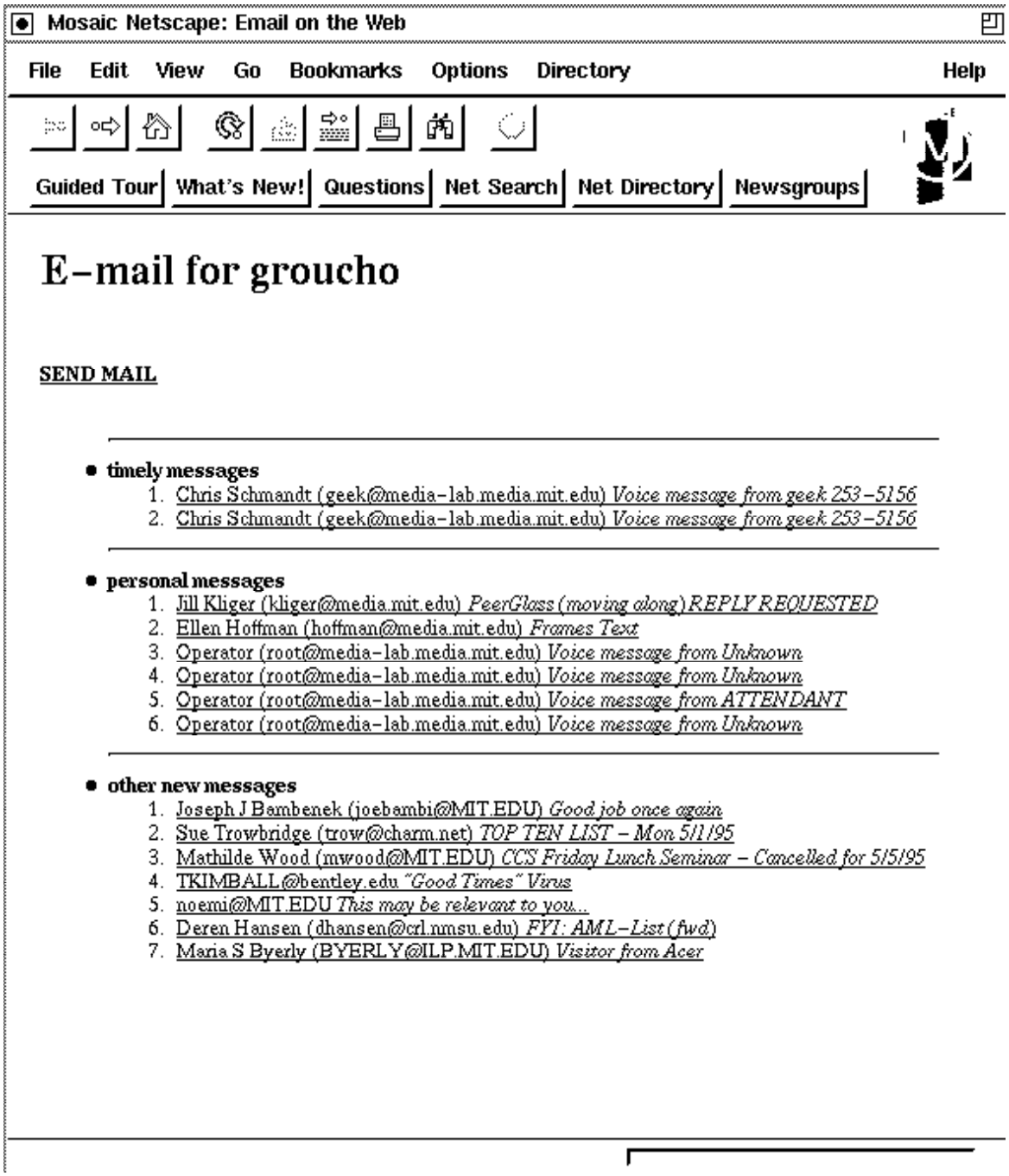

Figure 4.4: Summary-level view of HTMaiL.

Because the interpretation of raw text files may vary by browser, a separate process is run following the filtering which formats individual messages in HTML in order to homogenize and improve presentation. The script runs through the directory containing the parsed messages, formatting each message in turn. Header lines are italicized to distinguish them from the message text, with the typically more important header lines

like “From”, “Subject” and “To” highlighted in boldface. The message text is not changed except to put a hard return at the end of each line, since some browsers will interpret a file without a hard return as a single, continuous line, though the original sender may have intended paragraph breaks. At the bottom of the file is added a link “REPLY TO” which, if clicked, brings up a dialog box similar to that activated by the “SEND MAIL” button above, except that the To: field is filled in with the name of the message sender.

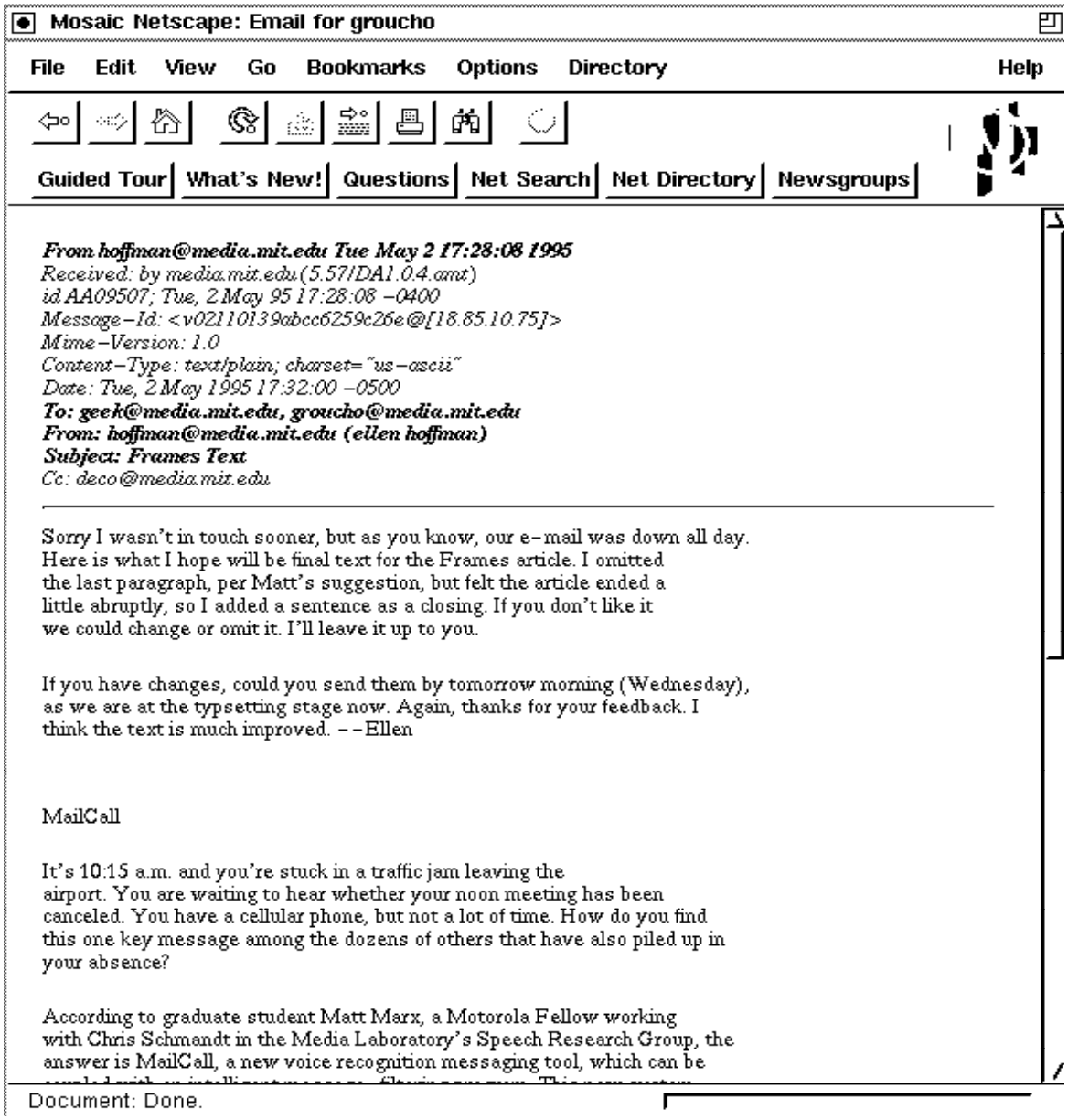

Figure 4.5: An email message formatted in HTML

As mentioned above, one receives email notification of voice mail messages. Since the name of the sound file is included in the text of the notification, the email notification serves as a pointer to the voice mail message. The HTML-ifying script, when it notices that a message is a voice mail notification, as discernible by the line

```
From: Operator <root@media-lab.media.mit.edu>
```

it changes the text of the message to "click here to play the voice message," which is written as a link to the sound file itself. Since many browsers do not recognize a sound file unless it has a suffix ".au" or ".snd", a soft link to the original message is made with the ".au" suffix appended to it.

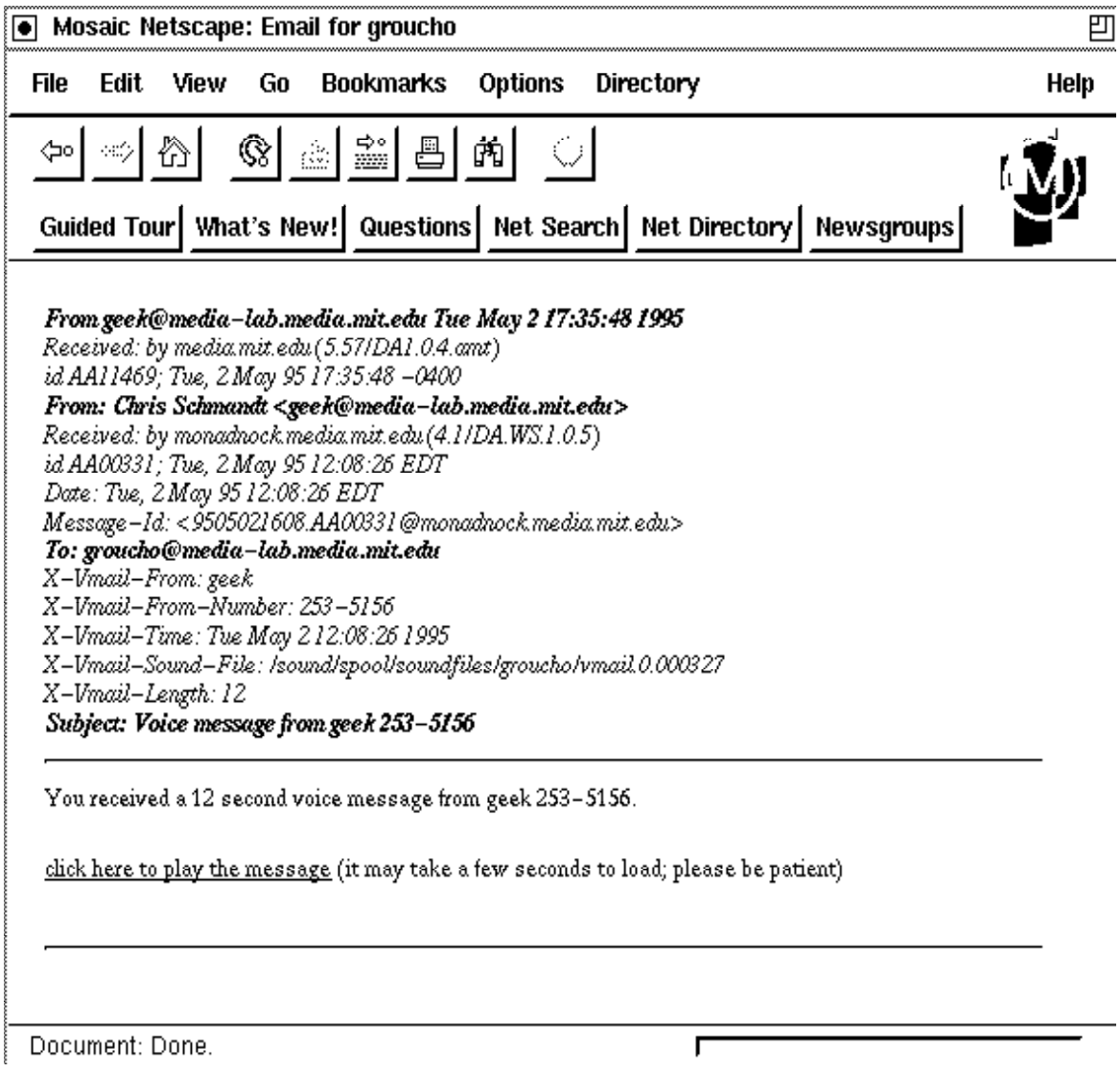

Figure 4.6: Accessing voice messages through HTMaiL.

The result is a screen-based mail reader which handles both voice and text messages. A step beyond mailers which list messages chronologically or sort only by sender or subject, HTMaiL presents the user with a structure of messages prioritized based on rules, both those written by hand and those generated from clues in the user's environment. Navigation among messages is accomplished with the familiar point-and-click style of the World-Wide Web; both voice and text messages can be presented. The mailing function allows the user to respond to messages—including the current message if desired—or initiate new messages. Thus in several ways, HTMaiL offers multimedia presentation capabilities superior to current screen-based mail readers.

HTMaiL is not, however, a full-fledged mailer. Messages cannot be deleted, nor can they be saved into specific mailboxes. In addition, the Netscape mail-sending tool does not support common conveniences like alias lists, or even Cc:/Bcc: fields. Also, updating the display requires a complete re-parsing and re-filtering of the spool file, which can be slow; incremental updates are not supported. Given that HTMaiL was originally intended as a demo system for visibly demonstrating the effects of filtering, the results are in fact quite pleasing.

4.7 Two users' experiences

The integration of CLUES into Phoneshell and the construction of HTMaiL months prior to the development of MailCall offered an opportunity to assess the long-term benefit of the filtering offered by CLUES. Two users, the thesis supervisor and the author, have been taking advantage of the filtering offered by CLUES for the last five months. Despite their different usage patterns, they have both found it to be beneficial. Since CLUES was designed to meet the needs of the designers, it is hard to say how well the approach will generalize. These positive results, however, indicate that at least a smaller goal has been achieved.

Chris Schmandt receives between 80 and 100 messages per day, most of which are addressed directly to him. Since he travels frequently, he is often away from the desk and accesses his messages by calling Phoneshell. He had been using filtering for some time, though simply separating messages addressed to him from those which were not was not of much use. He added several categories to his filtering profile with a host of rules, several of which were of temporary use (one rule was marked "just for this weekend's trip to CA").

He reports that the *timely* category generated as a result of CLUES typically contains several messages each session, and that those messages with few exceptions are more relevant than those contained in the more general "personal category." Before using CLUES, he was constantly adding rules to his filtering profile, but now the automatic rule generation of CLUES assumes much of that burden. In general, he finds the tracking of messages he has sent out most useful, since it often catches replies and helps to keep email conversations going.

His favorite anecdote about CLUES' utility helped him schedule an important sponsor visit while he was on a trip to Death Valley. A message from someone in the Media Lab sponsor relations department arrived, asking if a visit with Lotus could be set up for later that week. Since that particular sender was not in his filtering profile, it would have slipped through normally. But since Chris had been at Lotus the previous week and had the appointment "2pm Lotus" on his calendar, CLUES inferred that Lotus would be an important clue to Chris' interests. It matched "Lotus" against the subject of the message and prioritized it. Thus it was among the first messages Chris heard when dialing from a pay phone in Death Valley, so he sent a response and scheduled the meeting. When the Lotus visitors arrived later that week, Chris credited CLUES for making it happen.

Although he does not use the desktop HTMaiL mail reader, he often calls up Phoneshell to read his mail while sitting in front of the computer to get the benefit of its filtering!

The author, another frequent Phoneshell user, enjoys the benefits of CLUES and has in fact adopted HTMaiL as his mailer of choice. Receiving up to 200 messages a day as a result of being subscribed to several Internet mailing lists, he appreciates having his messages at the desktop sorted into categories and enjoys playing voice mail with the same tool he uses to read email.

His favorite anecdote involved having a message about a visit from Motorola put at the top of the list since it was on his calendar, even though the message would normally have been placed at the bottom of the list since it was not sent to him personally but to a mailing alias he didn't even know about. Normally he doesn't read messages in the "other" category until returning home, so if the Motorola message had gone unprioritized, he might not have read it in time to RSVP for the free dinner!

The fact that two users found CLUES useful does not mean that it will necessarily be useful to a general population, but the fact that it can satisfy their differing usage needs indicates that it has solved the local problem which was to be addressed.

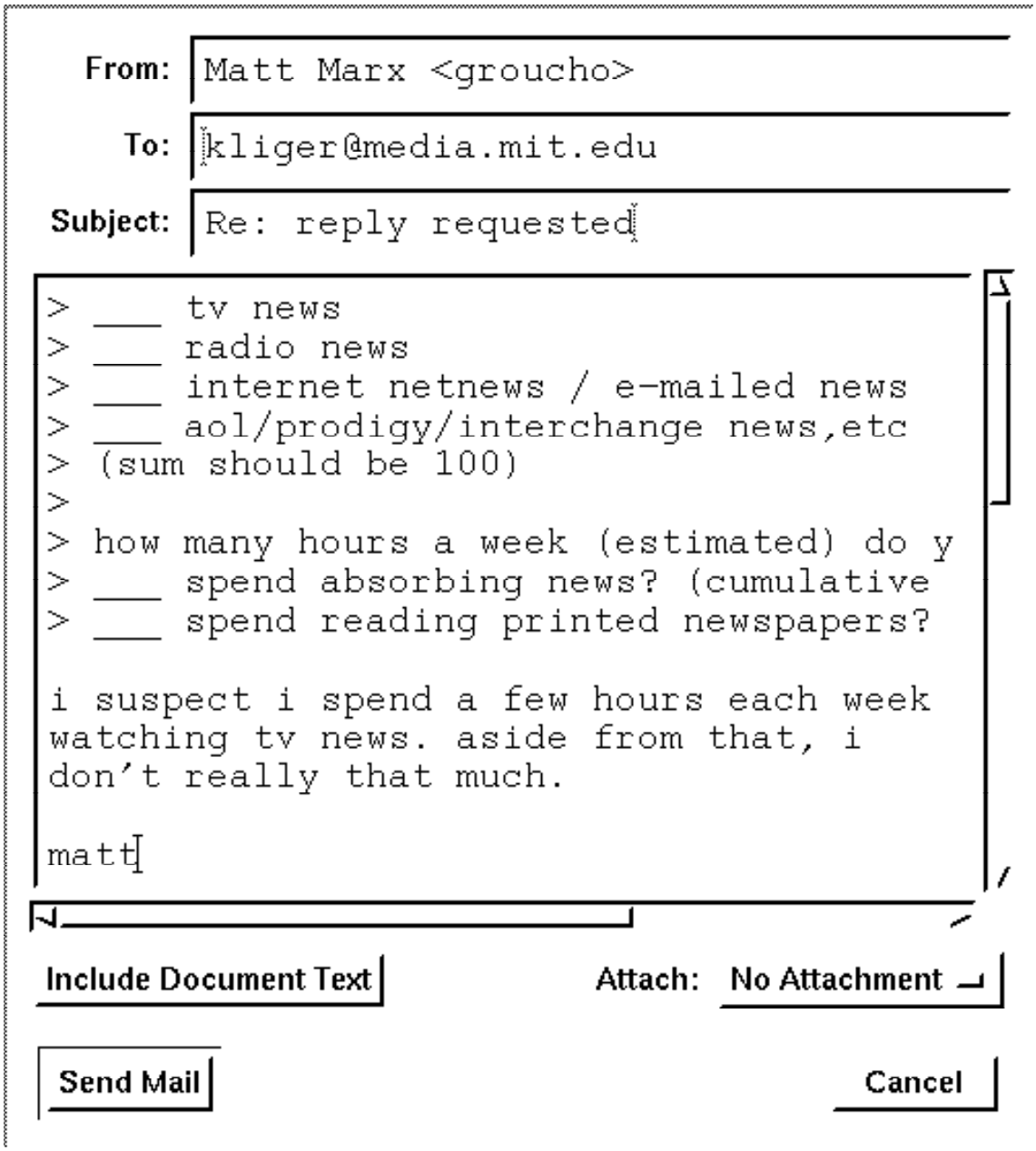

Figure 4.7: Sending a reply using a Netscape "mailto:" tag.

4.8 Limitations

As useful as CLUES is, it has several limitations which offer avenues for further work. This section describes those limitations and suggests initial approaches to alleviating them.

For one, CLUES sometimes overgenerates rules. If, for instance, one has an IBM demo scheduled, CLUES will look for mail with IBM in the Subject: or From: line. So if someone from IBM sends a message to a mailing list, that message will be mistakenly prioritized as important. The user ought to be able to give some feedback about mistaken inferences, and CLUES should incorporate this feedback into future rule generation. In short, the filtering mechanism should be adaptive.

As a step towards adaptive personalization, CLUES should explain itself better. MailCall cannot tell the user why a certain message was placed in the timely category though that may be very important, especially if a seemingly meaningless message is assigned a high category. CLUES should be able to pass on a record of which message met which rule, and how that rule was derived. This may be as simple as adding to a rule a two item list "keywords" and "source", separated by tabs. Then the filter could parse the keywords and source into separate fields and add them to the message structure, so that when the user asks why a certain message was important, the system can explain itself to some degree. One can imagine an interaction like the following:

MailCall:	Message 5 is from Deb Cohen about Motorola Fellows visit.
User:	Why was that prioritized?
MailCall:	It was prioritized because the word "Motorola" matched an entry in your calendar.

Figure 4.8: Explaining why a message was important.

Further, CLUES relies somewhat upon a knowledge base to accomplish its work—for instance, it relies upon a record of domain hosts to decide which ones are relevant for matching against the rolodex as well as a listing of which area codes are close to each other (see Section 4.2.2). Now, these databases are user-independent, so that the user does not even need to know they exist. At the same time, this can be a liability if the user wants to update or change the system's behavior. For instance, if the user learns that the domain **world.net** is a Compuserve-like service which spans the globe, it may be desirable to add it to the list which includes **aol.com, delphi.com,** etc. These databases are currently accessible though modifiable only with a text editor; further, the syntax of those databases is rather arcane.

CLUES is only an information gatherer and rule generator; an application wishing to use its rules must actually accomplish the mail parsing itself. Although several libraries are available to assist in this matter, being able to connect to a server and receive updates would be preferable. [Ly 1993] has supplied such a server, so it may be worth the time to integrate CLUES' rules into its functionality, making sure that client applications can be notified of a message's priority.

Finally, CLUES is not very exact. It uses large time windows and rather general notions of 'timeliness'. It doesn't know that an appointment ten minutes from now is probably more relevant than one next week, or that a message from someone you've been emailing and phoning repeatedly for the last four days might be more important than one which happened to match an item in your to-do list. Whereas CLUES uses a binary notion of importance (i.e., either it's timely or not), weighting the rules by temporal proximity, frequency, or some other measure would be helpful in generating a relevance ranking rather than just a single category of "timely" messages.

5. Audio-only navigation and presentation

Reading and replying to many messages can be especially time-consuming in an audio-only environment. Whereas a screen-based mailer offers a visual summary of the messages and easy selection by point-and-click, the aural landscape is not so easily traversed. This chapter discusses strategies for facilitating a more effective trek through the messaging information space.

5.1 Limitations of sequential navigation

Most telephone-based systems present messages sequentially—that is, after hearing a message one may either go to the previous message or the next one. Chatter uses this approach:

Chatter:	Hi. Who's this?
User:	Matt Marx.
Chatter:	Hi Matt! You have some messages. Hear them?
User:	Yes.
Chatter:	Message 1 is from Bruce McHenry about "tomorrow's demo canceled."
User:	Next.
Chatter:	Message 2 is from Minoru Kobayashi about "I can't make it."
User:	Delete it.
Chatter:	Message deleted.
User:	Next.
Chatter:	Message 3 is from Linda Peterson about "thesis drafts due TODAY."
User:	Next message.
Chatter:	Message 4 is from Chris Graves about "meeting tomorrow."
User:	Next message.
...a few minutes later...	
Chatter:	Message 23 is from Deb Cohen about "did Chris say anything about tomorrow's meeting?"
User:	Go back to the message from Chris Graves.
Chatter:	I didn't understand.
User:	Hmm. Previous message. Previous message. Previous message....

Figure 5.1: Navigating messages sequentially in Chatter.

Sequential navigation may suffice when there are just a few messages, but when that number grows into dozens or hundreds, it can be ineffective. In the previous section, a filtering strategy was presented for placing the user's important messages near the top of

the list.[16] Although filtering can ease the burden of digging through dozens of chronologically-listed messages, it would be unrealistic to presume that the structured list of messages always represents the sequence in which the user wishes to hear them. One can imagine wanting to call up, ask "Did Jennifer Lee write back?" and hang up immediately if not.

This is not to say that sequential navigation is useless. Indeed, when not rushed one may be perfectly content to go through messages one by one. This may also be an attractive option for someone concentrating carefully on another task (like driving, for example), since the cognitive load of sequential navigation is lower than goal-directed processing of random access [Zellweger 1988]. But since many users will be calling long-distance or from a cellular phone, often with very limited time, the value of random access cannot be overstated.

5.2 Category-level random access

One danger of navigating in a large, linear information space is getting lost. Categorization can help keep the user from getting lost among many messages by providing points of reference.

The SpeechActs filtering system, which generates many categories through grouping messages by sender and subject in addition to the user's filtering profile, makes additional use of its category structure by allowing sequential navigation not only of messages but of categories. This coarser granularity of navigation accelerates the process of skipping over uninteresting messages which are somehow related, as in the following example.

SpeechActs:	You have seventeen new messages and fifty-eight old messages. Your first two messages are from Lorin Bellow.
User:	Read them.
SpeechActs:	Message 1 from Steve Lee about "no rush on this one."
User:	Skip it.
SpeechActs:	Message 2 from Steve Lee about "this one's important."
User:	Read it.
SpeechActs:	"Hi Matt, there's something I really need you to do....." <message continues>
User:	Next.
SpeechActs:	Your next seven messages are about "can't we get a decent copy machine around here?????"

[16] It must be mentioned that Chatter uses memory-based reasoning in order to create a relevance ranking for the messages so that they are presented in order of importance.

User:	Skip those.
SpeechActs:	Your next message is from MIT.
User:	Let me hear that.
SpeechActs:	Message 10 from TechInfo at M I T dot E D U about "reprints."
User:	Read it.
SpeechActs:	"This is to let you know that your reprints of..."

Figure 5.2: Groupwise sequential navigation in SpeechActs.

Navigating sequentially by category can be especially useful for skipping several uninteresting messages on the same topic. This is preferable to hearing the first message on that subject and having to say "skip it" or "delete it" seven times to get through the block of related messages. Still, the user is not read an overview of the categories and cannot navigate randomly between them, so it may be difficult to get back to a set of messages later in the session.

Phoneshell allows random access among categories, though one must do so using touch tones. The user can press ***C** and then choose from the available categories as listed in a menu. Categories are listed in order of decreasing importance.

User:	<presses ***C**>
Phoneshell:	Choose a category. For very important messages, press 1. For timely messages, press 2. For personal messages, press 3. For other new messages, press 3.
User:	<presses **2**>
Phoneshell:	Timely messages. Message 3 is from Alex Shin about "hello."

Figure 5.3: Random access to categories using DTMF tones.

The user is thus able to jump to a specific category, but doing so requires a two-step process and listening through a menu of categories. This is similar to the location-based navigation described in the next section, since the user must remember the order of the categories or wait for the menu of choices to be read.

Although it does not deal with messaging per se, VoiceNotes offers random access among categories. In VoiceNotes, the user may create categories for individual notes by recording the name of the category. Thus the user may navigate among categories by speaking the name, as in the following example:

User:	To-do list.
VoiceNotes:	Moving into "To-do list."
User:	Play.
VoiceNotes:	*Pick up papers from Robin before 4pm.*
User:	Next.
VoiceNotes:	*Buy a loaf of bread on the way home.*

User:	Appointments.
VoiceNotes:	Moving into "Appointments."
User:	Play.
VoiceNotes:	*Meeting with Jan Ruggles at 2pm.*
User:	Next.
VoiceNotes:	*Call Chris Sommerfeldt before 3.*
User:	Ideas.
VoiceNotes:	Moving into "Ideas."
User:	Play.
VoiceNotes:	*Try the ISDN line in the sound room.*

Figure 5.4: Category-based random access in VoiceNotes.

Since a speaker-dependent recognizer is used, the recording of the category name is added to the recognizer's vocabulary. Supporting navigation among categories with MailCall is similar, although since a speaker-independent recognizer is used, the vocabulary of the system must be manipulated differently. The categories for a specific user are described in a filtering profile which is loaded at runtime, so they must be added to the recognizer's vocabulary at runtime. How this is done is described below in Section 5.6.

5.3 Message-level random access

Segmenting a long list of messages into categories can make navigation more manageable, but as the number of messages grows, the categories themselves will grow large. One can introduce more categories or even construct hierarchies of categories, but these are only short-term fixes; the problem at hand is providing random access among the individual items in a list, whether that be the entire set of messages or a just a sub-subcategory. By random access is meant offering the ability to navigate to destinations other than the previous message and the next one.

5.3.1 Location-based navigation

The simplest method of random access to implement is *location-based* navigation, where the user can jump to a message by virtue of its placement relative to other items in the list. Sequential navigation is actually a form of location-based navigation based on relative location: "next" or "previous." It is called *relative* location-based navigation since the user moves to another message based on its proximity to the current one. SpeechActs offers an embellishment upon relative location-based navigation where the user can skip ahead or behind by more than one message at a time; "go ahead five messages," for example, was supported.

SpeechActs also supported *absolute* location-based navigation, where the user can move to a message located at random, not by its proximity to the current message. Since each message is numbered, the user can easily skip to a desired message by saying (for example) "go to message seventeen." VoiceNotes offered an impoverished version of absolute location-based navigation: the user could jump to the first, last, or middle note in the list. The user is not, however, able to skip to a particular message by saying, for instance, "play the second message."[17]

One limitation of the location-based approach is that the user must remember the location of a particular message in order to find it. This may not be difficult with a small number of messages, though when the number is large the user is not likely to remember more than very generally where in the list the message is located. Revisiting the Chatter example, the user wanting to return to the message from Chris Graves would need to remember the number of the message in order to return to it. Indeed, subjects in the SpeechActs usability study were observed writing down the message headers and numbers so that they could return to them later [Yankelovich et. al. 1995]. While this may be possible when one has pencil and paper handy it is not while driving.

5.3.2 Content-based navigation

The extra burden imposed upon short-term memory by remembering the location of a particular message can be alleviated by *content-based navigation*, where the user can navigate directly by addressing some feature of the message itself—the sender, for instance. Saying "read the message from Lisa Stifelman" is easier than remembering that it's message twenty-three.[18] Further, field studies [Yankelovich 1994] suggest that users find content-based navigation quite natural and that they may be surprised if they can't ask about things by name.

As with location-based navigation, both absolute and relative methods of content-based navigation exist. Phoneshell allows the user to perform *relative* content-based navigation by looking for the next message from the same sender or on the same subject. Doing so places the user in scan mode. This kind of content-based navigation is called relative because the command issued by the user depends on the current state. When one reads a message from Jill Kliger and then jumps into scan mode, the command issued is to find

[17] It is likely that the limitations of the recognizer used would not have allowed this.

[18] The term *content-based navigation* should not be taken to imply that any sort of language understanding and semantic processing is being done—that is, one cannot expect a query like "did anybody in the Lab say anything about the election?" to process the text of the messages and return a result. The phrase *content-based* is used because the user is addressing the messages according to their inherent features or attributes (like who sent it or what the subject line says), not by external tags or labels (e.g., message #7 in a list of 10).

the next message *from the same sender* and not *the next message from Jill Kliger*. (Indeed, issuing the second command with DTMF tones would be rather difficult.) Although the distinction may seem moot in a touch-tone system, the two commands would mean very different things in a speech-driven system. The command "read a message from the same sender" would take the user to the next message from the current sender, whereas "read the next one from Jill Kliger" would jump to the next one from Jill Kliger regardless of the current sender. Thus resolving the command is dependent on resolving the command based on the current context.

Absolute content-based navigation, on the other hand, involves finding some inherent feature of an item itself, not its relation to other items. An example of this is found in MailCall, where the user can ask if there is ask if there are messages from certain individuals—system users, people in the user's rolodex, and people whom the CLUES filtering system thinks are currently relevant (see Section 4.2.3).

MailCall:	You have five important messages and six others.
User:	Is there anything from Jill Kliger?
MailCall:	Sorry, no messages from Jill Kliger.
User:	Did Charles Hemphill write back?
MailCall:	Nothing from Charles Hemphill, either.
User:	What about Lisa Stifelman?
MailCall:	Nothing from Lisa Stifelman.

Figure 5.5: Asking for messages from a specific sender.

Finding out whether there is a message from Jill Kliger requires no knowledge of the current context or the relation between different items. Absolute content-based navigation is of great worth when the user wants to zero in on one message in a large set. Asking for messages from a certain user is a perfect example; the user need not have explored the information space at all before issuing such a query. Another scenario is one where a summary of some items has been presented and the user wishes to select from among them. In MailCall, this applies both at the category level and the message level. For instance, once the user's password has been verified, the system reads a top-level summary of the messages and the user can choose to go to a category. Once in a category, one can hear summary of those messages and pick one to read.

MailCall:	You have two timely messages, seven personal ones, and fifty-three others.
User:	Go to my personal messages.
MailCall:	Personal messages. Read a summary, or start with the first one?
User:	Read me the summary.
MailCall:	You have a few from Raja Rajasekharan, a couple from Charles Hemphill, and one each from John Linn and Richard Wiggins.
User:	Read me the messages from Raja Rajasekharan.

Figure 5.6: Summarization and random access in MailCall.

Absolute content-based navigation requires knowledge about the content of the messages, and that this content is relayed to both the system and the user. How the language processing tools of the system—namely the speech recognizer and the parser—are informed of the information space is described in Section 5.6. How the user is informed is described in the following section.

5.4 Summarizing messages within a category

Absolute navigation, whether by location or by content, requires that the user know the shape of the information space and the system's vocabulary—either by manually browsing the information space or by listening to a summary. Summarization is not simple due to the conflicting goals of exposition and brevity: if the summary is not informative, then the user may not bother to ask for it; if it is not brief, the user may opt to explore the information space manually instead.

5.4.1 Scanning headers

Assuming random access, it is important to have an effective summarization of a list of messages. The simplest approach is to read the message headers one after the other without pausing or stopping for user input. SpeechActs does exactly this so that the user will know both which messages are in the category and what their numbers are.

SpeechActs:	Your next twelve messages are from people in the speech group.
User:	Scan the headers.
SpeechActs:	Message 1 from Gina-Anne Levow about "Swiftus problems."
	Message 2 from Gina-Anne Levow about "new prompts."
	Message 3 from Gina-Anne Levow about "is Hark slow?"
	Message 4 from Gina-Anne Levow about "can't compile."
	Message 5 from Gina-Anne Levow about "Weather app ready."
	Message 6 from Gina-Anne Levow about "help with Perl."
	Message 7 from Gina-Anne Levow about "weather works now."
	Message 8 from Stuart Adams about "can't compile."
	Message 9 from Stuart Adams about "is Hark slow?."
	Message 10 from Paul Martin about "Swiftus problems."
	Message 12 from Nicole Yankelovich about a response to "weather works now."
	Message 11 from Eric Baatz about "can't compile."
	Message 12 from Andrew Kehler about "Swiftus problems."
User:	Go to message 10.
SpeechActs:	Message 10 from Paul Martin about "anyone for hiking?"

Figure 5.7: Scanning headers in SpeechActs.

An obvious shortcoming of this technique is the need to remember the number of each message. If the system supported barge-in, SpeechActs might be able to use voice-direct

manipulation [Stifelman 1992, Muller & Daniel 1990] to say "that one" right after the header of the interesting message was read.

5.4.2 Clustering by sender

An alternative method of summarization, the one explored in MailCall, is clustering messages by sender. The goal is to provide an effective summary of the messages from which the user can then navigate using absolute content-based navigation. Concise wording of a summary is very important, for if the summary seems to take too much time, the user may abandon it.

First of all, the messages are grouped by sender. The assumption is that they will have been sorted by the filtering system and grouped from the greatest number of messages per sender to the smallest. Having done this, the summary would sound like:

```
You have    seven messages from Gina-Anne Levow,
            two messages from Stuart Adams,
            one message from Paul Martin,
            one message from Nicole Yankelovich,
            one message from Eric Baatz, and
            one message from Andrew Kehler.
```

One notices that the summary still sounds somewhat mechanical due to the repetition of words like "message" and "one." Conversationally speaking, these words are spoken early in the sentence and need not be repeated. The linguistic phenomenon of *ellipsis* refers to the omission of words which have been contextually established and thus can easily be inferred; MailCall uses ellipsis to prevent its summaries from seeming long-winded. The result is the following:

```
You have    seven messages from Gina-Anne Levow,
            two            from Stuart Adams, and
            one each       from Nicole Yankelovich,
                                Eric Baatz, and
                                Andrew Kehler.
```

In the example, words which have been added to substitute for those which were removed are italicized. (Note that only two words were added where 12 were removed.) Ellipsis can be very effective in helping a speech application not only to take less time, but to *seem* more efficient—i.e., there is a difference between actual speed and perceived speed. Repetitive feedback may feel even more long-winded than it is, just as an interesting lecturer can captivate an audience for well over an hour whereas a boring speaker can lose the same group within a few minutes. Ellipsis helps MailCall to avoid the repetitive feel which often characterizes the feedback of computational systems.

MailCall does not to announce the exact number of messages from a sender but rather a "fuzzy quantification" of how many messages there are; similarly, when it announces the length of a message (e.g. "incredibly long message...," "short message...") it gives a fuzzy quantification rather than a character or line count. This table represents the conversions for the number of messages:

# of messages from sender	representative description
1	"one"
2	"a couple"
3	"a few"
4-7	"several"
8-15	"a bunch"
>15	"an incredible number"

Figure 5.8: "Fuzzy" equivalents for message totals.

Finally, if there are more than a certain number of senders in the list, only a certain number of them are summarized. Research on short-term memory suggests that someone cannot hold more than seven items in short-term memory [Zimbardo 1985], and that more than four options in a menu system can decrease user performance [Engelbeck & Roberts 1989]. Interactionally, therefore, a point of diminishing returns may be reached if too many senders are named. The end effect is the following (using the same messages as in the SpeechActs example):

User:	Read the summary.
MailCall:	You have a bunch of messages from Gina-Anne Levow, a couple from Stuart Adams, and one each from Paul Martin, Nicole Yankelovich, Eric Baatz, Andrew Kehler and several more.
User:	Read me the one from Eric Baatz.
MailCall:	Message 11 from Eric Baatz about "can't compile."

Figure 5.9: A clustered summary of a category.

Here, the summary is shorter and more conversational, and the user is able to zero in on the message from Eric Baatz without having to remember its number or location in the list. By choosing its words carefully, making use of ellipsis, and limiting the length of a summary, MailCall informs the user of the currently available options.

5.5 Supporting multiple navigation strategies

As suggested above, the user may browse messages differently depending on the situation; hence, it may be advantageous to support multiple methods of navigating the message space. This includes a combination of sequential and content-based strategies.

Location-based navigation has not been implemented since the novel issue here is the use of content-based navigation, but it will be added if users regret its absence.

5.5.1 Sequential navigation

The unhurried user may elect to step through messages one by one similarly to the Chatter example above. The only difference is that MailCall reads an initial summary of the categories and then announce when a new category is entered (as does Phoneshell).

The user enters sequential mode by saying "start with the first message" after being read a summary. At that point, MailCall assumes the user wants to consider the messages one by one and will not ask if the user wants to hear a summary upon entering a new category but just read the first header.

MailCall:	You have two timely messages, five personal ones, and sixteen others.
User:	Start with the first one.
MailCall:	Timely messages. Message 1 is from Rebecca Prendergast about "tomorrow's demo canceled."
User:	Next.
MailCall:	Message 2 is from Jeff Herman about "I can't make it."
User:	Next.
MailCall:	Personal messages. Message 3 is from Santina Tonelli about "thesis drafts due TODAY."
User:	Next message.
MailCall:	Message 4 is from Chris Graves about "meeting tomorrow."

Figure 5.10: Sequential navigation in MailCall.

The categories still serve to establish the user's location in the information space but do not slow down the interaction significantly. Usually MailCall would ask "read a summary, or hear the first header?" when reaching a new category, but since the user entered a sequential-navigation mode by saying "start with the first one" earlier the first message is presented. The user is free to ask for a summary of the category at any time and go into content-based navigation mode.

5.5.2 Content-based navigation

The user can exploit content-based navigation in one of two ways: by asking if there are messages from someone in the rolodex, etc., or by requesting the messages from a sender mentioned in a category summary. If the latter is the case, MailCall knows the current category; otherwise it is announced as in the following example:

MailCall:	You have six timely message, twenty-four personal ones, and ninety-six others.
User:	Is there anything from Jamie Goldstein?
MailCall:	Timely messages from Jamie Goldstein....

Figure 5.11: Content-based querying.

Once MailCall has announced the category it is moving into (if necessary, as above), it moves to the first message from that particular sender and read the first header. Then the user can move among the messages from that sender sequentially. When the messages from that user are exhausted, MailCall simply moves to the next message in the list. The user can ask to hear the summary again at any time.

User:	Read the summary.
MailCall:	You have a bunch of messages from Gina-Anne Levow, a couple from Stuart Adams, and one each from Paul Martin, Nicole Yankelovich, Eric Baatz and Andrew Kehler.
User:	Read me the one from Eric Baatz.
MailCall:	Message 11 from Eric Baatz about "can't compile."
User:	Next message.
MailCall:	Message 12 from Andrew Kehler about "Swiftus problems."
User:	Read me the summary again.
MailCall:	You have a bunch of messages from Gina-Anne Levow, a couple from Stuart Adams, and one each from Paul Martin, Nicole Yankelovich, Eric Baatz and Andrew Kehler.
User:	Read me the one from Nicole Yankelovich.
MailCall:	Message 9 from Nicole Yankelovich about "weather works now."

Figure 5.12: Hearing a summary twice.

5.6 Updating the vocabulary on the fly

Facilitating the absolute content-based navigation described above requires performing some gymnastics with the speech recognizer. Most speech recognizers require the user to compile the vocabulary before starting the recognizer, a complicated process which can take several minutes. Dagger [Hemphill 1993], the speech recognizer used by MailCall, supports dynamic updating of the recognizer's vocabulary at runtime.

5.6.1 Updating vs. Subsetting

Updating the vocabulary is different from the vocabulary subsetting which many speech recognizers offer (and which is exploited elsewhere in MailCall, see Section 3.2.4); whereas subsetting allows the system to "gray out" useless regions of the grammar, Dagger allows the grammar to actually be modified. An illustration of the difference is the process of maintaining a user's list of personal contacts. With subsetting, the

developer would have to write a grammar which included *everyone's* rolodex and then "gray out" the ones which were useless. The following represents a grammar where all lists except for Heather's are grayed out:

```
Names-Matt ---> Gary Marx | Eric Ly | Jeff Herman | John Linn.
Names-Tom ---> Frank Martein | Josh Goldenhar | David Blank.
Names-Heather ---> Sherri Pendelton | Heather Foley.
Names-Jimmy ---> Jill Kliger | Sung Nam | Chris Kimball.
Names-Ted ---> Andy Brown | David Hoki | Paulina West.
```

Figure 5.13: "Graying out" portions of a recognition grammar for improved performance.

Although this accomplishes roughly the same thing, it is very inefficient. For one, it needlessly increases the size of the grammar to be compiled; excessively large grammars may fail to compile if not enough memory is available. More significantly, it requires constant re-compilation of the grammar whenever there is a change to any of the databases represented in the grammar—for example, when Heather adds "Anna Jorgensen" to her rolodex. Even if one had the computational means to devote to recompiling the grammars, one cannot hope to keep up with the constant flow of incoming messages—if the user receives a message just seconds before calling to check, one cannot hope to recompile the grammar quickly enough.

5.6.2 Meeting user expectations

Two types of vocabulary updating are performed in MailCall. The first type includes one-time updates such as the user's rolodex or list of categories; these take place as soon as the user is identified. The second type includes the information about the messages, such as the sender's name; this may be updated several times as the user moves among categories.

As with any recognition vocabulary, care must be taken to insure that the user knows what constitutes an allowable utterance; adding a thousand words to the vocabulary will not help if the user has no idea of what is available. Thus in each instance of vocabulary updating, attention is paid to user expectations.

With static information like the user's filtering categories or rolodex names, it is assumed that the user knows what is available. Thus it is safe to add the entire list. When it comes to messages, however, the user does not know which messages are going to arrive or who is going to send them; thus MailCall adds to the vocabulary only information about the messages which the user knows have arrived. Details are described below.

5.6.3 Updating one-time, user-specific vocabulary

Three items are updated as soon as the user is identified. First off, MailCall updates the list of the user's filtering categories as read from the user's filtering profile. Although it is not likely that the filtering categories will change very often, simply loading them at runtime for the relevant user avoids the unnecessary compilation and graying out as in the example above. Once the filtering categories are updated, they persist for the entire session. Since the system mentions the categories in the top-level summary, the user is reminded which categories are legal to ask about.[19]

Next, the names of people in the user's rolodex are added to the system's vocabulary. Names are read from the rolodex and added to the grammar[20] so that the user can call them, send them messages, or ask if there they have sent any messages. Since a rolodex can have well over a hundred names, the savings from not compiling the rolodexes for all system users into the grammar are great. Since the user cannot add names to the rolodex during a session, the rolodex names are only updated once. Here, the assumption is that the user knows the contents of the rolodex and will know who to ask about.[21]

The third update is the list of people which CLUES has deemed "timely"—that is, people whom the user has recently sent messages or called. (See Section 4.2.3) This list is read in and added to the grammar so that the user can issue queries such as "Did Bill O'Farrell write back?" when the user has been trying to get in touch with Bill recently. As with the above items, it suffices to update this list once at the beginning of the session. User expectations are a bit trickier here, however, since this list is dynamic. Someone who was on the list of "timely" people a few days ago may have dropped off, or the user may have forgotten to mark an outgoing message as being very important. The chance for dissonance between user expectation and system performance is greater. One can imagine the user asking if Tim Riker wrote back, and having the system confuse that name for Rich Miner since Tim Riker was no longer in the list of "timely" people.

5.6.4 Updating and re-updating information about messages

There are several approaches to updating the vocabulary. The simplest, as is used for the rolodex, is to update the recognizer's vocabulary with the names of everyone who

[19] Even if a certain category does not have any messages, it is still added to the vocabulary. Should the user request to go to that category, MailCall will report that the category is empty.

[20] See section 7.3.3 for details on preparing a grammar for vocabulary updates.

[21] One thorn in MailCall's side is that a name in the rolodex may be in a different form than the name uttered by the user, or the one that appears in a message header. Thus if the rolodex contains "Robert Sproull", the user may ask about "Bob Sproull," and the message header may read "Robert F. Sproull." Some general mechanism for inferring variations on a single name would assist this work.

have sent messages. The trouble with this approach is twofold: one, the perplexity may be extremely high if all the names are loaded simultaneously; two, if the user has not heard who the messages are from, having them all in the vocabulary is not of much use. One can guess which names might have been added to the vocabulary, but that increases the probability of misrecognition.

Another approach is to update the vocabulary by category. Here, one can either choose an incremental approach or a partitioned approach. In the incremental approach, one appends information about the messages to the vocabulary as the user moves from category to category, so that once all categories have been visited the vocabulary includes information about all those who have sent messages. This approach alleviates the problem of the user not knowing the contents of the vocabulary (though it relies on short-term memory to recall what all of that information was). It does not, however, solve the perplexity problem.[22] Experience has suggested that a large number of messages may produce an unwieldy grammar, so a partitioned approach was chosen instead.

In the partitioned approach, only the vocabulary for the current category is active at a given time. the user moves to a message category and asks for the summary of that category to be read, the vocabulary is updated with information about those messages. For this approach to work, the user must not assume that anything which has been summarized, whether in the present category or in another one, is fair game for recognition. Whether this assumption is sound is a issue for user testing. Once the user has read a summary of the messages in a particular category, however, a later return to the category will cause the vocabulary for that category to be updated automatically, without re-reading the summary. Thus the user can do the following:

MailCall:	You have two timely messages and five personal ones.
User:	Go to my personal messages.
MailCall:	Personal messages. Read a summary, or read the first header?
User:	Read a summary.
MailCall:	You have several messages from Ben Chigier and one from Jamie Goldstein.
User:	Go back to my timely messages.
MailCall:	Timely messages. Read a summary, or read the first header?
User:	Read me the summary.
MailCall:	You have two messages from Eric Davies.
User:	Go back to my personal messages.

[22] A third issue rears its head: if the user asks for a message from a sender for whom messages exist in two categories, which one does MailCall present? A reasonable solution implemented in MailCall is to check for messages from the requested sender in the current category first, and, failing that ,check the entire list. This is not an issue, however, when the partitioned approach is chosen, since only the names of the senders in that category are in the vocabulary.

MailCall:	Back to your personal messages. Read a summary, or read the first header?
User:	Read me the one from Ben Chigier.

Figure 5.14: Returning to a message from a certain sender.

Although updating the vocabulary is quite quick, it does introduce a slight delay which can be exacerbated by the time necessary to compile the names for the update and to generate the summary.[23] So if the user asks for a summary of a category for which the vocabulary was just updated, only the summary is generated.

5.7 Limitations

Content-based navigation via dynamic grammar updating, in concert with sequential navigation, offers the user flexibility in dealing with large lists of messages. Querying the system for messages from certain users can be particularly useful if one has a specific goal and little time to waste. As MailCall is an initial attempt to exploit Dagger's dynamic grammar capabilities, it leaves open several avenues for design and development. At the same time, its liabilities become readily apparent.

5.7.1 Specifying names

One such liability is that MailCall does not offer much flexibility in the way the user refers to someone. If, for instance, the name Steven C. Wheelwright is listed in my rolodex, and I want to find out if he sent me any messages, I might want to ask "is there anything from Steve Wheelwright?" or even "is there anything from Steve?" MailCall currently cannot handle either of these; it insists that a name be spoken as it is listed in the rolodex, message header, etc. Although this constraint helps to cut down on ambiguity, it also forces the user to speak awkwardly at times: e.g., "read me the messages from Professor Robert C. Berwick." Two extensions would be useful: one, which for a given name produces a list of acceptable "alternatives"—leaving out a middle name or initial, for instance, or substituting an abbreviated form of someone's first name. The other extension would be to support use of the first name only. This would be convenient at login time as well as while browsing messages. The following scenario seems entirely reasonable:

MailCall:	Welcome to MailCall. Who's this?
User:	It's Matt again.
MailCall:	Matt Marx?

[23] The delay is nearly negligible, however—usually less than a second.

User:	Yes.
MailCall:	Hello Matt. Please say your six-digit password.
User:	Five three one zero zero four.
MailCall:	Hold on while I get your messages....Okay! You have three important messages.
User:	Go to my important messages.
MailCall:	Important messages. You have two from Chris Schmandt and one from Jordan Slott.
User:	Read the one from Jordan.
MailCall:	Message 3 from Jordan Slott about "Truetalk interruption".....

Figure 5.15: Supporting reference to people by first name only.

Now, this only works if there is only one system user with the first name "Matt"—otherwise there needs to be some disambiguation dialogue in order to establish which Matt is calling. SpeechActs has such a name disambiguation dialogue which, for example, once it decides that "Bob" means Bob Sproull and not Bob Kuhns, will resolve further references to Bob as Bob Sproull until the user mentions "Bob Kuhns," in which case it switches the referent. Such a scheme would greatly benefit MailCall by making exchanges more natural; if the last name is not necessary to disambiguate the person, then Grice's maxim of quantity dictates that one need only use the first name. Finally, with a large number of names, the chance for confusion is quite great. A disambiguation algorithm based on spelling [Marx & Schmandt 1994a] will be incorporated into MailCall if this proves to be a limitation of MailCall; experience with Chatter suggests that this will be the case.

5.7.2 Other content on which to base random access

Another avenue for exploration is the use of information other than the message sender. The user may appreciate a summary along subject lines as well as senders. This can help especially when one subscribes to mailing lists where many people comment on a single topic, and one wants to delete a series of messages or skip over them.

User:	Go to my other messages.
MailCall:	Other messages. You have one from Lena Davis and two about "looking for a good mechanic in Cambridge."
User:	Delete the ones about "looking for a good mechanic in Cambridge.

Figure 5.16: Random access to subject lines.

In addition, users may appreciate the ability to ask for messages about events on their calendar they same way they ask if there are messages from people in their rolodex. One can imagine a scenario like this:

MailCall:	You have six timely messages and three others.
User:	Is there anything about Nokia?
MailCall:	Timely messages about Nokia. Message 3 from Chris Gant about "Final Schedule: Nokia 3/11/95."
User:	Read it.

Figure 5.17: Asking for messages related to events in the user's calendar.

Using calendar information should not be difficult; the words themselves exist in the rules generated by CLUES, and the same rules could be used to search the headers of messages at runtime to find matches. Adding entire subject lines to the vocabulary is a bit trickier due to their unpredictable syntax: numbers, in particular, are difficult to model. In trying to make speakable links from World Wide Web pages for a speech-aware HTML browser, numbers and acronyms were found to be very difficult to model. [Hemphill & Thrift 1995]. The difficulty is exacerbated by the fact that Swiftus does not handle numbers easily; to handle the occasional numbers which pop up in email addresses, MailCall converts them into their stringified equivalents—thus "brown5@harvard.edu" becomes "brown five at harvard dot e d u." Further work with Swiftus may alleviate this difficulty, and perhaps some collaboration with text-to-speech developers (who constantly wrestle with issues of rendering numerical combinations in audio) would help.

6. Avoiding conversational breakdown

In conversational interaction, special care is required in order to sustain the confidence and interest of the user [Hayes & Reddy 1983]. Given the invisibility of a speech-only environment, a number of factors can contribute to conversational breakdown. Unlike a GUI, no visible menu of available options is present, so the user may become lost more easily. Further, unlike the visually persistent feedback of an alert box, which remains on screen for perusal and re-examination until the user is comfortable enough to go on, auditory feedback is temporary; hence, not understanding feedback the first time can be disorienting. Finally, speech recognition errors are a thorny problem. Two causes of recognition errors are predictable phenomena like out-of-vocabulary utterances and unpredictable phenomena like background noise. The approach to minimizing the impact of recognition errors, decreasing the number of out-of-vocabulary utterances and handling those recognition errors which occur despite one's best efforts, is described below. The error-handling system is an interface algorithm called ErrorLock.

6.1 Learning the system's vocabulary

The first focus in minimizing the impact of recognition errors is to decrease the number of out-of-vocabulary utterances (OOVs). Although expert users have had the chance to learn the vocabulary and will thus utter fewer OOVs, beginners suffer severely for lack of knowledge. The invisibility of application functionality is particularly daunting for the first-time user. There exist at least three methods for helping the user to learn the system's vocabulary: *directive* prompts, *time-out* (or *delayed*) prompts, and offering *context-sensitive help*. The pros and cons of these approaches, as well as their implementation in MailCall, are described below.

6.1.1 Directive prompts

Explaining one's options at every point in the discourse is the simplest approach and can help keep beginners from getting lost. Most interactive voice response (IVR) systems using DTMF touch-tones list the user's choices at every stage of the interaction; the prompts are *directive* in the sense that the user is guided toward issuing a valid command [Kamm 1994].

For your checking account balance, press one. For your savings account balance, press two. For interest rates and service charges, press three.

Figure 6.1: IVR-style listing of options.

The same technique can be employed for prompts in a speech recognition system in attempt to keep users within the system's vocabulary. By presenting choices one by one, MailCall takes advantage of the *parroting effect,* the tendency to phrase a response using the same language in which the request was presented. Thus the option "You can go to a message category." may be more likely to evoke the desired response "Go to my personal messages." than "What's in my personal messages today?" which is not in MailCall's vocabulary.

Of course this technique is not foolproof, for the user may say something other than what is suggested; unlike the telephone keypad, where input is restricted to twelve keys, the user is still free to utter anything which can be produced by the vocal tract. For this reason it is imperative to give a more complete list of options in a help sequence.

Even when directive prompts are being given, the principle of tapered presentation [Yankelovich et. al. 1995] applies. MailCall can avoid repetitiveness and perceived slowdown by giving successively shortened versions of the prompts. Tailoring prompts to the user may help reduce recognition errors and misunderstandings [Hirschman & Pao 1993]. For instance, after the user has recorded a voice message, the system asks "Do you want to review the message, send it, or cancel?" If the user then reviews the message, the next prompt is shortened to "Review, send, or cancel?"

Even if prompts are tapered, the user may decide what to say before the prompt is finished. In this case, the user may be tempted to "barge-in" and give the response before the prompt is finished. If barge-in is not supported, speaking too early can lead to confusion and recognition errors, since nothing is processed by the speech recognizer while the synthesizer is still finishing its work.

The system supports barge-in when run in ISDN mode, since ISDN does not mix the voice channels of the two speakers. Thus the user is free to interrupt a lengthy prompt (or the reading of a lengthy email message) at will. When using analog telephony—which will almost always be the case when calling from outside the office—barge-in is not supported. In order to coax the user into not speaking too soon, MailCall can play a brief prompt tone to signal that the synthesizer is finished speaking; it is hoped that the user will become accustomed to waiting for the prompt tone before speaking.

Directive prompts can be tiring for expert users, especially with analog telephony where barge-in is not supported. Thus the user can choose to eliminate the wordy prompts if desired; a command-line option specifies the use of directive prompts. When run in expert mode, the system gives no prompts except where it is necessary to inform the user of exceptions to normal circumstances. An example of this is when the user tries to respond to a voice message, the origin of which is unknown; the system has to tell the user "sorry, you can't respond to that message" lest an attempt is made to record a voice message. Here is the same interaction with and without verbose prompts:

User:	send a reply.
MailCall:	Record your voice message at the tone. Pause for several seconds when finished. <tone>
User:	*blah blah blah blah blah blah.* <pause>
MailCall:	Got it. Review the message, send it, or cancel? <tone>
User:	Send the message.
MailCall:	Message sent. What now? <tone>

Figure 6.2: Sending a message with verbose prompts.

By contrast, the same interaction without verbose prompts:

User:	send a reply.
MailCall:	Record your message. <tone>
User:	*blah blah blah blah blah blah.* <pause>
MailCall:	Got it. <tone>
User:	Send it.
MailCall:	Message sent. <tone>

Figure 6.3 Sending a message with terse prompts.

The latter interaction is obviously more efficient than the former, where the user must wait through lengthy prompts before speaking. Note that feedback is not completely absent, however—only the directive prompts. MailCall still gives implicit feedback regarding the user's last action: "record your message" lets the user know how the latest command was interpreted; "done" informs the user that recording is complete; and "message sent" provides confirmation, as well.

6.1.2 Time-out prompts

Another approach is to give directive prompts only if the user fails to issue a command within a few seconds. This is accomplished by setting a timeout callback in the event loop so that once *n* seconds have passed, MailCall is notified. If nothing has been recognized by that point, MailCall assumes that the user hasn't said anything and then issues another, more helpful, prompt.

The assumption behind giving timeout prompts is that the user is confused, does not know what to say, and is waiting for guidance from the system. If this is the case, then timeout prompts may be keep the user from feeling lost. If, however, the user is simply trying to decide what to do next, the timeout prompt may be unwelcome. Predicting the user's intention is an inexact science, so the utility of this technique must be evaluated by some usability testing.

The current approach to deciding when to give timeout prompts is not entirely satisfactory since the user may begin to speak near the end of the timeout window. If the user is still speaking when the timer runs out, or if recognizer is not finished processing the utterance, then system does not know the user is speaking and will mistakenly interrupt with a prompt. A better solution which takes into account whether the user has started speaking is to arrange for the audio server to send regular energy events to the system and count time by measuring the number of events multiplied by the interval between events. A value above a certain threshold indicates that the user is speaking, in which case the counter is reset. Technical difficulties have so far prevented this solution from being implemented in MailCall.

6.2 Context-sensitive help

Regardless of whether directive prompts are used, it is imperative that a system offer help at the user's request. A new user may not have grasped from the directive prompt exactly what needs to be said. An expert may temporarily "blank out" on the wording of a certain command. And for any number of reasons one may become confused as to what state the conversation is and what the system is prepared to accept as a command: perhaps the synthesizer is difficult to understand; perhaps background noise provides a temporary distraction; perhaps a sneeze or cough triggers a series of misrecognitions. For whatever reason, it is important that the user be able to reestablish context, explicitly understand the system's vocabulary, and if all hope is lost, find a better way out than just hanging up the phone.

6.2.1 Step 1: re-establish context

The first step in providing meaningful help is tell the user what just happened by revisiting the previous utterance. The system tells the user what it heard and what action was taken, which provides more complete feedback than would usually be given. This can be particularly helpful when a substitution error results in the system doing

something other than what the user intended. In the following example, the help facility provides a more complete explanation of what happened following a substitution error.

User:	Who was that from? **<MailCall hears "who was that sent to?">**
MailCall:	It was sent to groucho, geek, and, lisa with copies to frank at m i t dot e d u and Bruce Papazian.
User(confused):	Help!
MailCall:	I just read you the recipients of the current message.

Figure 6.4: Reestablishing context through help.

It is important that the system phrase its explanation in terms of its behavior, not in terms of the user's intention—to claim "you just told me to..." is to make the mistaken assumption that the recognizer has necessarily understood the user's intent; this may engender frustration when the system is in error. That is, the system should explain what it has done (i.e., "I just...") rather than what it thinks the user said (i.e., "You just told me to..."), for in the case of a substitution error the user may become argumentative.

6.2.2 Step 2: list the current options

The second step is to provide the user with a list of legal commands in the current discourse state. This involves both explaining the general nature of the options and also providing specific examples of what the user may say.[24]

In determining what the allowable commands are, MailCall need only look at the active regions of the grammar (see Section 3.2.4); this reveals which commands are currently valid.(One exception: even though the "HELP" region of the grammar is active at all times, MailCall doesn't bother telling the user about this command, presumably because the user has at least been successful at issuing *that* one!) In the following example, the user asks for help after moving to a message category. The currently active regions of the grammar are

```
NAVIGATE-FIRST ("read the first header", etc.)
NAVIGATE-CATEGORY ("go to my personal messages", etc.)
PRESENT-CATEGORY ("read the summary", etc.)
HELP ("help", "what can I say now", etc.)
NAVIGATE-START-OVER
```

MailCall:	Timely messages.
User:	What can I say now?

[24] Experimentation has been done with using audio clips of actual spoken commands rather than the synthesizer. In addition to being more intelligible, hearing a live voice speak the command continuously may serve as a set of speaking tips, weaning some users of the tendency to place exaggerated pauses between words.

MailCall:	We just moved to your personal messages category. To start with the first one, say 'read me the first header'. To hear a summary of the messages in this category by sender, say 'read me the summary'. You can also go to another message category; for example, to go to your personal messages, say 'go to my personal messages'. If things get confused, you can say 'start over from the beginning' to return to the top-level summary of your messages and restore any deleted messages.
User:	read me the summary.

Figure 6.5: Explaining the user's current options.

As described above, the help system first describes the resulting action of the user's last utterance (more exactly, the system's interpretation of the user's last utterance) and then provide possible commands. Several points about the wording of the list of commands can be made. First, the help system offers both a description of the command as well as a recording of the command itself; this is done because the brief command may not be suitable to explain its functionality. In addition, if there are many options, one should list them in order of local relevance to the current state: for instance, if the user has just read a message header, it may be more useful to present the options dealing with single messages before those dealing with moving to different categories.

Since giving help can take quite a long time, especially if there are a lot of options at a certain point, it is important that the user be able to interrupt the stream of help messages. When MailCall is run in barge-in mode, the user can interrupt using voice. When touch-tones are listened for, the user can not only interrupt the help sequence but also skip back and forth among the different help options. Using touch-tones, the user can skip back and forth between help messages just like inside an email message.

6.2.3 Step 3: offer a way out

The third step is to provide the user with a way out. The worst possible case is for the user to hang the phone prematurely, since a resort to such a measure indicates serious user frustration and a decrease in one's confidence in the system. Thus a single command, "start over from the beginning", returns the user to the initial state of the system: the user is at message number one, in no particular category, with the status of all messages marked new and unread. The command is available at all times and is verified since it causes everything to be reset. Although the consequence of this action may be rather severe, it nonetheless offers the user an alternative to hanging up and calling again, which is costly both in terms of time and money.

6.3 Providing coherent, comprehensible feedback

Particularly when using synthetic speech, comprehensibility is a serious concern [Pisoni 1985]. If, for instance, one cannot understand the header of an email message, how is one to decide whether or not to read it? Some user-configurable parameters are available as well as automatic adjustments to insure that feedback is understood.

6.3.1 Varying the speed of output

Just as novices may need longer prompts than experts, they may need them to be spoken more slowly since they are unfamiliar. The converse applies for experts, who may find it ideal to have prompts spoken quickly. Similar to adapting to the prompts, the user will become accustomed to the synthesizer over time [Pisoni 1985]. These phenomena suggest the need for a variable speech output rate.

Given that the user has control of the speech output rate, how should this control be exercised? Some expert users may wish to specify the rate in words-per-minute, which would call for a rather large vocabulary to handle all supported speeds, possibly resulting in more recognition errors. Speech synthesizers have individual upper and lower bounds on output rate, so an effective mechanism would have to adjust the vocabulary for the rates supported by each synthesizer, or provide boundary-checking mechanisms to inform users of the available speeds. Although this is not unreasonable, it is suspected that since only expert users would be likely to take advantage of this feature, that they would not be likely to change the rate that often, and so a large vocabulary of word-per-minute rates may in fact be counterproductive. Less experienced users may be more likely to adjust the speaking rate and less likely to seek the user of a discrete control. Thus a relative control is provided, which allows for commands like

User:	Talk faster.
MailCall:	Did you say to increase the speech output rate?
User:	Yes.
MailCall:	Increasing speech output rate.

Figure 6.6: Increasing the speech output rate.

The increment selected is 40 words per minute, a number which may change with experimentation and user feedback. Others suggest the need for a non-linear control, which might be helpful as well [Stifelman et. al. 1993]. If the user tries to increase the speed beyond the capabilities of the synthesizer, the system announces that it is already at the highest speed.

Although the user's desired speech output rate is stored within the discourse manager, communicating that to the text-to-speech engine requires some additional maneuvering since synthesizers use different escape sequences to change the rate of output, and in fact use different representations of speech output rate. For instance, DECTalk uses the

```
[:rawpm]
```

command to change the speaking rate, where *wpm* is the rate in words-per-minute, whereas TrueTalk uses the

```
\!Rrel
```

command, where *rel* is the speed inversely proportional to the default speed (190 wpm). Thus *rel* = 1.0 is 190wpm, *rel* = 1.5 is 95wpm, and *rel* = 0.5 is 285wpm. Thus converting from words per minute to the TrueTalk representation involves the calculation

```
truetalk_rate = 2 - (<words-per-minute> / 190)
```

Wishing to shield the discourse manager from these arcane commands and conversions, the mechanism for setting the output rate is incorporated within the text-to-speech server. The application sends a *tts_setrate* command with the words-per-minute count as an argument, and the server does the conversion (if necessary) and sends the appropriate command.

6.3.2 Adjusting speed for given vs. new information

As discussed above, one may want to increase the speech output rate as one becomes more accustomed to the system prompts. Yet there is much information provided by the system to which the user has not become accustomed: for instance, the name of someone who has sent a message. Indeed, a common complaint of Chatter and SpeechActs users was that although system prompts were (or became) easy to understand, message headers and text were often cryptic. Now, part of this can be attributed to the unpredictable nature of header and body text, but there is certainly a difference between presenting familiar (or *given,* in linguistic terminology) information and *new* information.[25]

If both given and new information is presented at the same output rate, and one has difficulty understanding new information, one may be forced to decrease the overall output rate. Then, although new information may be more understandable, given

[25] Although people may not necessarily slow the rate of their speech for new information—placing emphasis on the word may be more common—the difficulties users have had understanding the synthesis suggest that slowing down the output rate may be more helpful.

information will again become tiring. It would be preferable to have given information presented at a faster rate and new information at a slower rate, and to have this decrease in speed be relative to the global output rate set by the user (and possibly modified during the session).

A function *slowify* is provided for temporarily slowing the output speech rate for a piece of text in a longer string without modifying the global output rate; it inserts the synthesizer-specific commands as necessary, slowing down a certain amount from the global output rate.[26] The function is called for new information such as the headers of messages, with the following effect (expanded text indicates a slower speaking rate, not that the words are being spelled out):

User:	read me the first header.
MailCall:	Incredibly long message 4 is from C h r i s S c h m a n d t about t h i s a f t e r n o o n ' s m e e t i n g.

Figure 6.7: Slowing down while presenting new information.

6.4 Effectively handling recognition errors with ErrorLock

Despite MailCall's best efforts to inform users of the system's vocabulary, and due to unpredictable factors like line and background noise, recognition errors will occur. How a conversational system handles errors can mean the difference between success and disaster. Any messaging system which deletes messages, hangs up, or throws the user into some strange state for no apparent reason will quickly lose user confidence; likewise, if feedback is repetitive and uninformative, the user may perceive the system as hostile and unhelpful.

ErrorLock is an application-independent interface algorithm for managing speech recognition errors. Three goals in the design of a facility for dealing with recognition errors are centralization, homogeneity and extensibility: that is, 1) a single mechanism rather than code copied in several sections of the application, 2) similar behavior regardless of the current place in discourse, and 3) appropriate feedback regardless of the nature and content of the error.

[26] One can imagine chopping up a sentence into several smaller pieces, and adjusting the global output rate between them, but delays in text-to-speech conversion render such an approach ineffective.

6.4.1 Progressive assistance for rejection errors

A *rejection error* refers to the recognizer's inability to decode an utterance. This may take two forms: either the speech recognizer processes an utterance and returns no result (like someone asking "What did you say?") or when it returns a result but is not very confident (like someone asking "I'm not sure, but I think you said 'goodbye'. Is that right?"). Some recognizers are more likely than others to return a result even when unsure of its accuracy, while others are very strict and often return no result instead of returning one with a low confidence.[27] Dagger, the speech recognizer used by MailCall, tends to return a result most of the time, though often with low confidence. In the interest of avoiding too many misrecognitions, ErrorLock throws out results below a certain threshold.[28]

A typical approach to handling rejection errors is to give some indication that the utterance was not processed, for example by saying "I didn't understand you." The danger here with such an approach is that a string of rejection errors can lead to the "brick wall" effect, where the system uncooperatively repeats the same error message over and over.

User:	Read me the text of the next message.
MailCall:	I didn't understand.
User:	Read me the text of the next message.
MailCall:	I didn't understand.
User:	I said READ ME THE NEXT MESSAGE!
MailCall:	I didn't understand.
User:	Gee whiz, I just don't know what to say.
MailCall:	I didn't understand.

Figure 6.8: The "brick wall" effect of rejection errors.

The user in this situation may quickly become frustrated with the apparently unhelpful conversant. A better strategy is to offer increasingly more help, finally dropping the user into the help system. This technique, called *progressive assistance*, assumes that a long string of recognition errors is a rather sure sign of out-of-vocabulary errors or extreme background noise, whereas the occasional rejection error might be caused by passing phenomena such as speaking too soon.

The strategy is as follows: for the first rejection, ErrorLock asks a very short question to indicate that an error has occurred. The interrogative nature of the response may

[27] Many recognizers allow the developer to set a rejection threshold to raise or lower the sensitivity of the recognizer.

[28] The threshold is set differently for telephone-quality speech and microphone-quality speech, and for male vs. female models, as accuracy varies among these two axes. Male speech through a microphone seems to be the most accurately modeled whereas female speech over the telephone seems to be the most poorly modeled.

encourage the user to repeat the previous utterance, and its short length decreases the probability that the user will speak too soon. A subsequent rejection suggests that the utterance is probably an OOV, so it asks the user to rephrase the utterance in hopes that a different wording will match something in the current vocabulary. If the user gets four rejections in a row, the system drops directly into help.

To further reduce the monotony of error messages, ErrorLock incorporates a degree of randomness in generating feedback. Instead of having a single, standard error message for each number of consecutive rejections, one is selected at random from a group of semantically synonymous but phonetically distinct phrasings. As a result, the feedback for the first rejection error may be "What?" or "Huh?" or "Excuse me?"

By giving progressively more assistance as the user experiences consecutive rejection errors, the "brick wall" effect may be avoided. Here's the same dialogue from above, incorporating progressive assistance.

User:	Read me the text of the next message.
MailCall:	What?
User:	Read me the text of the next message.
MailCall:	I didn't understand. Please rephrase.
User:	I'd like to hear the body of the message following this one.
MailCall:	Still no luck. Say it a different way.
User:	Go and find me the next message.
MailCall:	This obviously isn't working. Here's what you can say.....<drops into help>...

Figure 6.9: Avoiding the "brick wall" effect with progressive assistance.

Not being able to specify a command due to background noise may be very frustrating, for the interference may keep the confidence score just below the threshold. To combat this problem, ErrorLock records the user's successive attempts, so that if the user says "read the message" three times but never gets above the threshold, then the system can infer that the desired utterance is barely out of reach, ask "did you say to read the message?" and then go ahead when the "yes" is given.

6.4.2 Identifying potential substitution errors

Substitution errors are more difficult to detect than rejection errors. Although the user can tell whether or not the recognizer has correctly interpreted the utterance, the system has no way of knowing this. Thus the approach is different from that with rejection errors: whereas there the system knew immediately that an error had occurred, here the user's help is necessary in order to determine whether a mistake has occurred.

The simplest technique is to verify the utterance by reading the user the text returned by the recognizer and asking if it is correct.This technique will prevent the system from immediately performing an action the user did not request due to a substitution error, which is a first step toward effectively managing substitution errors. Quite clearly, however, verifying every utterance may soon become tiring if the recognizer was correct, as in the following example:

MailCall:	Personal messages. Read a summary, or read the first header?
User:	Read me the summary.
MailCall:	Did you say "Read me the summary?"
User	Yes.
MailCall:	You have several messages from Rich Schwartz, a few from John Makoul, and one from Bruce Papazian.
User:	Read me the ones from John Makoul.
MailCall:	Did you say "Read me the ones from John Makoul?"
User:	Yes.
MailCall:	Long message 3 from John Makoul about "see you next Thursday."
User:	When did that arrive?
MailCall:	Did you say "When did that arrive?"
User:	Yes.
MailCall:	It arrived about three hours ago.
MailCall:	Did you say "Read it?"
User:	Yes.

Figure 6.10: Asking too many verification questions.

Such an interaction is tiresome because the conversation is constantly punctuated by verification questions. Chatter opted not to ask verification questions at all, presumably for fear of slowing down the conversation too severely, but this resulted in numerous substitution errors; for instance, it would often hang up for no apparent reason based on a misrecognition. The Pegasus system did not verify utterances, either, although it offered a pair of "undo" commands to insure that no action was irreversible [Zue et. al. 1994]. Undo commands can help, but verification is necessary to some extent, since it will save the user in situations like this:

MailCall:	Long message 3 from John Makoul about "see you next Thursday."
User:	Let me hear it.
MailCall:	Did you say to hang up?
User:	No!

Figure 6.11: Verifying a command to hang up.

Like SpeechActs and VoiceNotes, MailCall asks verification questions only when it believes that executing the action without first asking the user for verification may have very negative consequences, as in the above example.

The notion is that some utterances have a higher cost than others. Those which *present* information, as in reading a stock quote or the body of an email message, have a low cost since they do not do anything damaging. Re-reading an email header, for instance, may take a matter of seconds; should that happen by mistake, the user need not wait very long for control to return. And if barge-in is supported, then reading even the longest email message can be curtailed. Those which *navigate*, as in going to a different section of messages, also have a low cost since the user can always return to the previous state.[29] If the system accidentally goes to one's personal messages, one can easily issue the command "go back to my timely messages" at the next prompt. In the interest of not slowing down the dialogue unnecessarily, ErrorLock does not ask a verification question for an utterance of low cost. For higher-cost utterances, however, it will ask a verification question. Higher-cost utterances include those which are *irreversible* (e.g., "hang up") or which *destroy data* (e.g., "delete the message").

An addition to the SpeechActs delineation of utterances is another class which are not irreversible and do not destroy data but throw the user into a subdialog. For instance, when the user says "send a reply", the system records a voice message and then asks whether it should be sent. Now, the user can exit this subdialog by saying "cancel,", but being thrown into the subdialog by mistake will slow down the interaction. Verifying every time the user says "send a reply" may prove irritating, however, so it tries to verify selectively. The criterion here is the confidence score returned by the recognizer. For example, if the recognizer is 90% confident, the command is executed; if it is only 50% confident, however, ErrorLock checks before proceeding. Although the particular recognizer in use does not generate reliable confidence scores, they prove helpful more often than not.

ErrorLock relies heavily on the parser to streamline the verification mechanism. Fortunately, the Unified Grammar coupled with the Swiftus parser allows it to extract large amounts of data from the parse and in fact describe much of the application's behavior. For instance, the following is the parser entry for "delete the message":

```
{manipulate-message/delete := "delete the message";
   action := 'delete;
   object := 'message;
   cost := 'high;
}
```

[29] Though this can be tedious if the user was near the end of "personal messages" and is thrown into "other messages" by mistake. Going back involves returning to the personal messages and going one by one until returning to the original message. An "undo" command might be very helpful in this case.

When the recognizer returns the phrase "delete the message", the parser returns the three feature/value pairs defined in the grammar entry. This is combined with the text returned by the recognizer and the confidence score to fill the fields of an *utterance structure*:

```
struct utterance {
   text: "delete the message"
   confidence: 1.5
   parse: "ACTION:DELETE OBJECT:MESSAGE COST:HIGH"
}
```

Figure 6.12: an utterance structure.

The "cost:high" pair in the parse reveals that this is an utterance which needs to be verified before proceeding. Similarly, if the pair had been "cost:variable" ErrorLock would check the confidence score of the recognizer and proceed if the score were below a certain threshold.

6.4.3 The language of verification

Once ErrorLock has decided to verify an utterance, it must then choose the wording of the verification question. One danger of centralizing the verification system is over-homogenizing the language of the verification, which can result in a mechanical feel. Thanks again to the Swiftus parser, the developer can specify the verification prompt in the grammar (thus freeing the verification mechanism of the need to manage such information). The verification prompt for "hang up" is deduced from its grammar entry:

```
{logout := "hang up";
   action := 'logout;
   cost := 'high;
   verify-prompt := 'did-you-say-to-hang-up;
}
```

Figure 6.13: the grammar entry for the "hang up" command.

The parse includes the feature/value pair "verify-prompt:did-you-say-to-hang-up", so ErrorLock changes the dashes into spaces, tacks on a question mark, capitalizes the first letter and speaks the prompt. It can be useful to specify a single verification prompt for a class of utterances, so that the synonymous phrases "hang up", "good-bye", "that's all for now", and "so long" are all verified similarly. By specifying both the cost of an utterance and its verification prompt in the grammar, the verification mechanism is kept compact yet can handle any sort of verification.

Now, the developer should not be forced to specify a verification prompt, so ErrorLock provide some default prompts in case "verify-prompt" is not found. ErrorLock

prepends the text returned by the recognizer with "Did you say" and append to it a question mark. Thus if the command is "delete the message" and no verify-prompt is given, the verification prompt will be given as "Did you say 'delete the message'?"

The word choice of a verification prompt is crucial. Many systems ask the question "do you really want to....?" whereas ErrorLock favors "did you say to...?" The reason behind this choice is the conversational implicature [Grice 1975] of each. The former implies that the system has understood correctly and that the user may have misspoken, whereas in reality it is the accuracy of the recognizer which most often needs to be checked. Using the former can lead to exchanges like the following:

User:	Go to my timely messages.
MailCall:	Timely messages. Read a summary, or read the first header?
User:	Read the first header.
MailCall:	Message 1 from Joey Francis about meeting agenda.
User:	Next message.
MailCall:	Message 2 from Yezdi Lashkari about free food.
User:	Next message. **<this utterance is mistaken for "hang up">**
MailCall:	Do you really want to hang up?
User:	.....uh.....I didn't say that....why does it think.....?

Figure 6.14: Asking the wrong verification question.

The user is confused because the question the system asked presupposed that "hang up" had been spoken, when in fact something entirely different was said. One may wonder why on earth the system thinks one would want to hang up! The same session with the verification question "did you say to hang up?" may not confuse the user as much, since it is clear from the wording of the question that the system thinks the user said "hang up" in some form or another.

6.4.4 Facilitating one-step correction

When a verification question is posed, the system is not simply looking for a 'yes' or 'no'. Usability studies show that although people are quite content to say 'yes' when they agree, disagreement more often brings a full-blown correction than a simple 'no' [Yankelovich et. al. 1995].

MailCall:	Did you say to delete the message?
User:	Yes.
MailCall:	Message deleted.
User:	That's all for now.
MailCall:	Did you say to start over from the beginning?
User:	No, I said that's all for now!

Figure 6.15: Just saying "yes" but not saying just "no."

This phenomenon may occur because the notion of a dialog box does not work in auditory space the same way it does on a visual screen, where the user can see the choices as well as the explicit boundaries. In a speech-only domain, neither the boundaries nor the options are persistently visible, so there are fewer constraints on the user to respond only with 'no'.

The wager here is that attempting to model the correction the user makes instead of saying "no" will be more fruitful than putting up a brick wall until the user complies. On the other hand, doing so makes for a larger vocabulary and increases the chance of recognition errors. But with this strategy, the user is often able to correct system mistakes in one step, as in the latter example above, rather than in two steps.

User:	Read it.
MailCall:	Did you say to delete the message?
User:	No.
MailCall:	My mistake. What did you say?
User:	*Read* the message.

Figure 6.16: Correcting a mistake in two steps.

When making a correction, a user often adds a corrective prefix like "no, I said..." to emphasize that a correction is being made [Kitai et. al. 1991] (though the user often will express the same through increased intonation alone [Stifelman 1993]). In the interest of modeling such corrections, ErrorLock changes the active regions of the grammar to include those prefixes as well as the usual utterances. This is accomplished by inspecting all of the active regions and replacing them with slightly modified regions which contain the original utterances both with and without the corrective prefixes. Thus if the system were listening for

```
Read the message.
```

it would now be listening for

```
Read the message.
No, read the message.
I said read the message.
No, I said read the message.
```

The ability to do this requires that special regions be defined in the grammar to include the corrective prefixes.

6.4.5 Taking the blame for recognition difficulties

It is important that a spoken language system be accountable for its mistakes, especially since misrecognitions are so common. If the system attempts to blame the user for its

own difficulty, the user may be offended. If it seems indifferent, user confidence may disappear. Accomplishing this includes apologizing for recognition errors. Effective apologies must explicitly take responsibility for the problem, should be commensurate with the difficulty the user is experiencing, and ought not be canned or repetitive.

ErrorLock chooses its words carefully when generating an apology. It does not use ambiguous words like "sorry" and "oops" which can be used to attribute blame to either the system (the sense "sorry, I goofed') or the user (the sense "sorry, you can't do that."). Instead it explicitly takes the blame by saying "my mistake" or "I goofed."

Whenever the recognizer makes a mistake, ErrorLock selects an apology based on how many consecutive errors have been made. If only a couple of errors have been made, a short apology like "my mistake" will likely be given. If, however, several errors have occurred in a row, a more lengthy apology like "I really got confused that time; sorry about that" may be more appropriate.

In the interest of not sounding repetitive, ErrorLock introduces some randomness into its selection of an apology. Several different wordings are available at various levels of apology, and one is selected at random. Varying the output may help to avoid the mechanical feeling which can result from always saying the same thing.

6.4.6 Dealing with consecutive substitution errors

As the large vocabulary of the regions with corrective prefixes may suggest, it is possible, even likely, that the user will go through more than one round of correcting the recognizer. Here the problem is similar to that of multiple rejection errors, so ErrorLock provides progressive assistance in the same way, giving longer apologies until a point where suggests asking for help.

User:	Read the message.
MailCall:	Did you say to delete the message?
User:	No, I said read the message.
MailCall:	My mistake. Did you say to send a reply?
User:	*No, READ the message!.*
MailCall:	I'm sorry; I goofed. Did you say to hang up?
User:	NO!!!!!!
MailCall:	I'm really not getting this. What did you say?

Figure 6.17: Progressive assistance for multiple substitution errors.

One mistake unique to some speech recognizers is the tendency to make the same mistake multiple times. While trying to identify one's self, for instance, one may experience difficulty in getting the system to know who one is (note that the user name is verified because getting the wrong name could be very costly):

MailCall:	Welcome. Who's calling?
User:	Matt Marx.
MailCall:	Jordan Slott?
User:	No, Matt Marx.
MailCall:	My mistake. Was it Jordan Slott?
User:	*of course not!!! I just said that!!!*

Figure 6.18: Making the same mistake twice in a row.

In order to avoid sounding oblivious to previous errors, ErrorLock keeps track of the user's past attempts at issuing a command so that if the speech recognizer repeats a misrecognition, the program can at least acknowledge the fact and sound a bit less banal. By keeping track of the past attempts, the above sounds like:

MailCall:	Welcome. Who's calling?
User:	Matt Marx.
MailCall:	Jordan Slott?
User:	No, Matt Marx.
MailCall:	Well, I heard Jordan Slott, but you said that before. Who is it?

Figure 6.19: Overcoming repetitiveness by tracking past attempts.

If the speech recognizer returned not just a single utterance but a list of its top choices (called an *N-Best* return), ErrorLock could use its history to eliminate those names which the user had already rejected. So if the recognizer returned, say, its three top choices, it could cross off the one which has already been rejected and use the next one. With an N-best return the interaction might be a bit more successful:

MailCall:	Welcome. Who's calling?
User:	Matt Marx. **<MailCall hears 1) "Jordan Slott" 2) "Matt Marx">**
MailCall:	Jordan Slott?
User:	No, Matt Marx. **<it hears 1)"Jordan Slott" 2) "Matt Marx", but can cross out Jordan Slott>**
MailCall:	Matt Marx?
User:	Yes.

Figure 6.20: Exploiting N-Best in handling multiple substitution errors.

In summary, using a history of attempts with multiple substitution errors can help ErrorLock to react more intelligently and win the user's trust.

6.5 Touch-tones

Since MailCall supports touch-tone control as well as voice commands, it is important that ErrorLock deal with touch tone input as well. This is actually very simple; since DTMF detection is so robust, rejection and substitution errors are not really an issue. A more complicated issue is how to detect and screen out the misrecognitions which result when the speech recognizer mistakes a touch-tone for speech. Since in the asynchronous architecture (see Section 7.2) it is impossible to know whether the recognition server or DTMF server will return an event first, there is no way to reliably throw out the misrecognition. If the DTMF server returned its event before the recognition server, then the following event from the recognition server could automatically be thrown out. But this is not the case, so throwing out the next recognition event after a DTMF tone may cause MailCall to miss the next instruction from the user. The current approach is to do nothing but just deal with the false recognitions inside ErrorLock; a preferable arrangement would be to have the recognition server distinguish between DTMF tones and speech so that only a tone event or a speech event would be sent as appropriate.

MailCall translates DTMF tones into spoken commands and sends them as if they had been recognized, noting however that the command resulted from a touch tone. When ErrorLock notes that it is a touch-tone command, it bypasses the usual verification mechanisms and simply executes the command.

6.6 Summary

ErrorLock helps to avoid the "brick wall" effect of rejection errors and the "runaway effect" of substitution errors, and helps MailCall to be generate appropriate feedback when an error or multiple errors occur. ErrorLock is implemented in MailCall as a set of procedures, but they are independent enough of the application to be compiled into a separate library and used for another application. Thus ErrorLock is best thought of as an *interface algorithm*, a methodology for dealing with recognition errors. The diagram on the next page represents the flow of information and action within ErrorLock.

It should be clear from the diagram that ErrorLock is independent of the task at hand; indeed, it is this task-independence which allows ErrorLock to be integrated into a variety of applications. The feedback generation can be customized in the grammar specification, as described in Section 6.4.3. In summary, ErrorLock relieves the developer of the need to consider how to handle recognition errors in the many different contexts in which they appear.

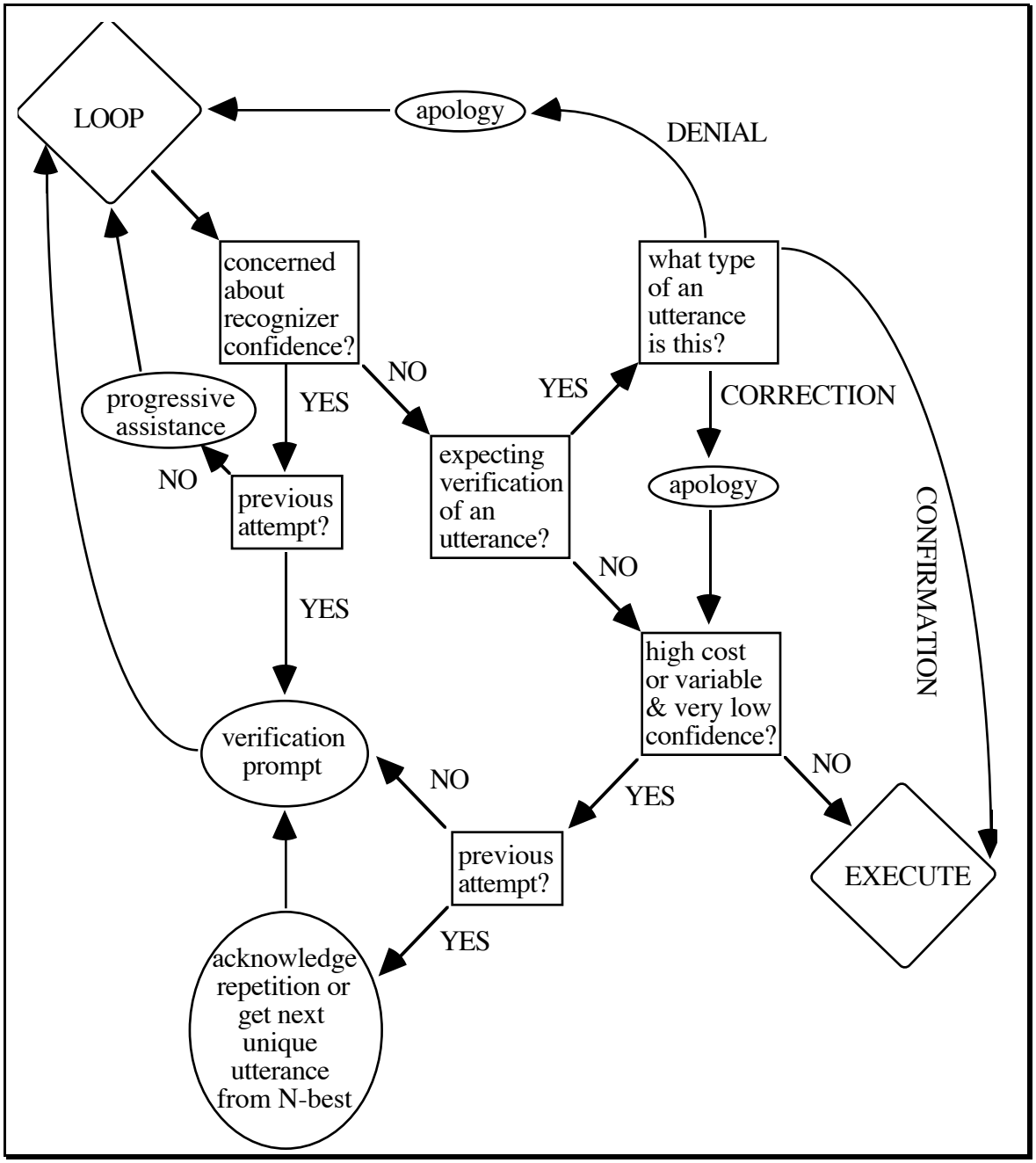

Figure 6.21: ErrorLock, an interface algorithm for handling recognition errors.

7. System Architecture

MailCall is a complex, distributed application. More than a dozen separate processes are active during a single session, not to mention the work being done in the background to generate filtering rules.

7.1 Hardware, operating systems and programming languages

MailCall was developed on a multiprocessor Sun Microsystems Sparcstation 20 with 96 megabytes of memory; this seems to be the minimum power required to run MailCall at full speed. With less memory or processing power, delays in mail filtering, speech recognition and parsing can be anticipated.

Since MailCall was developed during the introduction of the Solaris operating system in the speech group, and since the speech group has many machines which will likely continue to run SunOS 4.1.x, the choice was made to develop MailCall simultaneously for Solaris and SunOS 4.1.x. The only differences between MailCall for Solaris and for 4.1.x is that the XTL call control library for ISDN is available only on Solaris, so in order to use MailCall on a 4.1.x machine, a computerfone must be installed.

Built upon a significant legacy of tools for speech and audio management developed under SunOS 4.1.x, working with Solaris required the porting of many libraries and servers which will be used in future projects for a variety of researchers. Most changes were in the Makefiles, though some code modifications were made. In all cases, however, a single set of files, compilable under both operating systems, is kept rather than having one set of files for 4.1.x and one for Solaris.

As MailCall consists of several separate programs, different languages are used as appropriate. The central MailCall executable, which parses messages, generates feedback, and manages interprocess communication, is written in ANSI C. As described in chapter 4, the clue-gathering and building of regular-expression rules is done in Perl. The servers vary between C and C++.

7.2 Asynchronicity

MailCall is an asynchronous, event-driven system. After connecting to various servers (described in the next section), it sits in an event loop, waiting to be notified of an incoming call. When the call is identified, it speaks a greeting and then sits in an event loop, waiting for commands in the form of speech or touch-tones. Once the appropriate action is taken, MailCall returns to the event loop. Should the user hang up prematurely during a session, an event is sent alerting of the disconnect, and MailCall exits. The asynchronicity of MailCall is made possible by a series of callback mechanisms [Arons 1992] and servers, which shall be described in the next section.

7.3 Server Architecture

In an attempt to make MailCall independent of machine architecture or the specific speech tools it utilizes, a set of servers mediate its interaction with low-level devices. Most of these servers were legacy applications of the speech group, though some of them were modified or created especially for MailCall.

7.3.1 Existing speech group servers used in MailCall

A phone server [Schmandt & Casner 1989] detects incoming calls as well as when the user hangs up. An audio server [Schmandt & Slott 1995] mediates the capture and presentation of audio data, whether through the audio device or through an ISDN port. A user utterance is passed from the audio server to a recognition server [Ly et. al. 1993], which employs the user's choice of recognition engines to process the audio. (Modifications made to the recognition server to support dynamic grammars are discussed below.) A copy of the audio is also sent to a DTMF server, which sends MailCall an event when it detects a DTMF tone. MailCall then processes the incoming text via the Swiftus parser [Martin & Kehler 1994] and generates the response. If the action only involves playing a sound file, it is sent directly to the audio server. If, however, voice synthesis is involved, the text is sent to a text-to-speech server which passes the text on to the user's choice of text-to-speech engine, which returns the relevant audio to the text-to-speech server, which then sends it to the audio server to be played. The flow of information among the servers is represented in the diagram on the following page.

The recognition server existed prior to the development of MailCall but was modified in order to take advantage of Dagger's dynamic grammar updating capabilities. The text-

to-speech server was constructed during the development of MailCall to provide an abstraction above multiple text-to-speech engines as well as to enable asynchronous synthesis (for barge-in). The next few sections detail the modifications to the recognition server and the construction of the text-to-speech server.

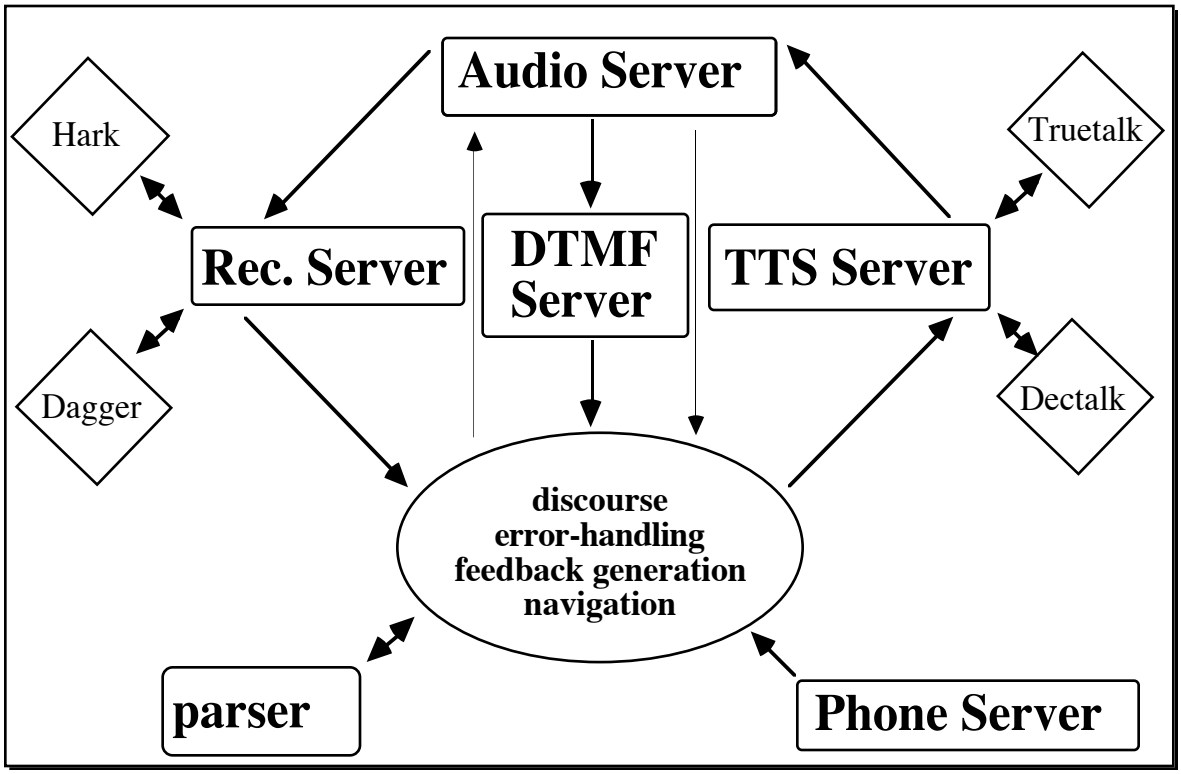

Figure 7.1: MailCall's servers and their interaction.

7.3.2 Enabling dynamic grammars within the recognition server

Leveraging Dagger's new capabilities of dynamic grammar updating required some modifications to the recognition server, hereafter called the *r_server* [Ly et. al. 1993]. The goal was to make the process of adding a new grammar as painless as possible while offering shortcuts for those willing to do extra work for the sake of speed.

Dagger updates a grammar by taking a grammar represented in a list structure and comparing it against the grammars currently in memory. If the grammar is not already defined when the recognizer started up, the new grammar is added; if the grammar is defined, then it is replaced with the new one.

The r_server offers the developer two methods for updating grammars on the fly: one, simple and flexible but slow; the other, quick but complex and specialized. The former involves sending the r_server a text file containing a grammar to be updated. The r_server then reads in the file, converts it to a linked-list grammar structure using Dagger's routines, and sends it to Dagger for updating. This is the simplest method of updating a grammar since the grammar can be written in the usual BNF format as the developer is accustomed to doing. It is also the most flexible, since the grammar read in from a file can be arbitrarily complex. The drawback, however, is that the time taken to read a grammar from a file and convert it is substantial, and may result in a slowdown of a second or more. But this may suffice in situations where a slight slowdown can be masked with an audio watch cursor or a lengthy prompt, as when one is switching between applications.

Such a delay is unacceptable during a dialogue, however. If, for instance, one is read a list of the people who sent messages, one does not want to wait several seconds before being able to respond "read the one from Chris Graves". It is for this reason that a second, more streamlined method of updating a grammar is supported. In this mechanism, the user sends a pre-formatted string to the r_server as well as a single nonterminal which one wishes to replace. The r_server then rearranges the string in the linked-list grammar format and processes it in one-tenth of a second or so, a serious savings in time.

The drawback is twofold: one, that only a single nonterminal can be updated at a time. (This is not a problem for MailCall, since it only needs to update the grammar with the appropriate names; the structure of the rest of the vocabulary is already expressed.) The second problem is rendering the expansion of a nonterminal to be sent over a socket to the r_server. The format is actually rather simple, though it adds to the developer's list of conventions to learn. Consider the expansion of a grammar nonterminal EMAIL_SENDER in the following example. If using the file method, the developer could write the following BNF grammar to a file and then have the r_server read it in.

```
start(EMAIL-SENDER).
EMAIL-SENDER --> Jack Fogarty.
EMAIL-SENDER --> Mike Kellison.
EMAIL-SENDER --> Lela Cokeson.
```

If using the nonterminal method, however, the client program sends the nonterminal EMAIL-SENDER as the first argument and its expansion as a second argument. The expansion is represented as a single string with spaces delimiting words and a special character (one not legal in a grammar specification) to delimit each expansion. In this case, the expansion would be

```
Lisa Stifelman ~Jordan Slott ~Barry Arons ~
```

If the grammar is specified in this form, the update of a nonterminal can be very fast, sometimes less than a tenth of a second, which is more than satisfactory for MailCall's purposes.

7.3.3 Preparing a grammar for on-the-fly expansion

At runtime MailCall adds several lists of names to the vocabulary (see Section 5.6). Since the language of navigation is largely the same (i.e., "go to my _____ messages", "read me the ones from ____"), most of the vocabulary can be precompiled with a few slots to fill at runtime. Thus the grammar for the recognizer has rules like the following:

```
NavigateCategory ---> go to my CategoryName messages.
CategoryName ---> "".
NavigateSender ---> read me the ones from SenderName.
SenderName ---> "".
```

where the nonterminals CategoryName and SenderName can be expanded into the user's category names. It turns out that the Unified Grammar compiler [Martin & Kehler 1994], in the process of generating a recognizer-specific grammar from a more general specification, removes any non-expanded nonterminals. Thus the developer has to 'trick' the parser into thinking that the nonterminal has been expanded. This is done by adding an empty lexicon entry which is then compiled into the recognizer's grammar.

```
(empty
 (sem . CategoryName)
)
```

Thus the resulting specification for the recognizer has the nonterminal CategoryName expanded as

```
CategoryName ---> empty.
```

which means that the utterance "go to the empty messages" is valid. This is remedied by replacing the definition of CategoryName with the actual names before allowing the user to exercise this command. The same applies to the nonterminal for the names of people who have sent the user messages.

Replacing that nonterminal involves rewriting the lexicon with the appropriate name; instead of having an empty entry the following entries now exist:

```
(timely
 (sem. CategoryName)
 (category-index . 0)
)
```

```
(personal
 (sem. CategoryName)
 (category-index . 1)
)

(other
 (sem . CategoryName)
 (category-index . 2)
)
```

MailCall then reloads the lexicon into the parser and send the new expansion to the speech recognizer (described in Section 5.6). At this point the client program activates those regions of the grammar so that the user can perform absolute content-based navigation.

7.3.4 Creating a text-to-speech server

As with speech recognizers, multiple speech synthesizers exist with varying areas of expertise. Whereas recognizers tend to be evaluated quantitatively (word-error rate, feature set), synthesizers often have their appeal in more qualitative respects. A DECtalk veteran may have adapted quite well to its conventions and therefore find it the easiest to listen to. A pronunciation purist may be enchanted by Orator. Others may find Centigram to have a very pleasant voice. Linguists may be impressed by the intonational control possible with Truetalk. Whereas people tend to flock to the most accurate recognizer available, individuals may express strong preferences for one synthesizer or the other. The text-to-speech server described in this section allows for run-time selection of a synthesizer to match the user's taste.

Failing to abstract above the application programming interface for a given synthesizer leads to undesirable dependencies in the code or a series of switch statements. The current text-to-speech server, hereafter referred to as the *tts_server*, may seem bureaucratic since it only supports two synthesis engines, but as that number grows its benefit will become readily apparent.

The server's main function is to act as an intermediary between a client program and a synthesis engine. Since the server may support multiple clients from different hosts wanting different engines, it must keep track of each connection, its desired engine, and to which audio server it wants output data to be delivered.

The issue is complicated slightly because the output of the hardware DECTalk is not stored in a sound file but directly passed through a cable to a speaker. Thus the tts_server cannot reroute the audio to another host if the client wishes to use DECTalk. For TrueTalk and other software-based engines, however, the client may specify that the audio be routed to any host. Thus the tts_server can act as an intermediary for a client on any other machine.

To use tts_server, a client opens a connection and specifies which synthesizer it wants, sets the output rate, and specifies the destination host machine. The tts_server then sits waiting for the client program to send it text to be spoken. The client can also recognize asynchronous callbacks [Arons 1992], so text can be spoken synchronously or asynchronously. If synchronous, the tts_server speaks the requested text and then returns from the function; if asynchronous, the tts_server returns immediately and then speaks the text, returning a callback if desired. When text is spoken asynchronously and callback are requested, the client can interrupt the speech output with a call to *tts_cancel_message*.

7.4 Third-party software

MailCall uses a variety of third-party software for recognition, synthesis, call control, and parsing. Each is described briefly.

7.4.1 Speech recognition

By using the recognition server, MailCall can choose between speech recognizers. Although the r_server supports both speaker-dependent and speaker-independent recognizers, both discrete-word and continuous speech, MailCall requires speaker-independent, continuous speech recognition. Thus the options are Dagger [Hemphill 1993a], from Texas Instruments, and Hark [Hark 1993] from BBN[30]. Both are software-only recognizers which support grammars in Backus-Naur Form. Dagger is more fully integrated into the r_server, including the support of dynamic vocabulary updating, but when Hark is integrated it may serve as an alternative.

[30] Hark is actually not quite integrated, but its lack of dynamic updating capabilities at the time of MailCall's development made Dagger the recognizer of choice anyway.

7.4.2 Speech synthesis

The text-to-speech server interfaces to two separate engines, DECtalk [Digital 1987] from Digital Equipment Corporation and TrueTalk [Entropic 1995] the AT&T Bell Labs Synthesizer marketed by Entropic Inc. DECTalk is a hardware box whereas TrueTalk is software; they can be used interchangeably since the text-to-speech engine provides a layer of abstraction.

It is worth noting that another synthesizer is used, not for speech output but to generate pronunciations for the recognizer. Adding a name to the recognizer's vocabulary is nontrivial due to the various nationalities from which names spring; using English text-to-speech rules is unsatisfactory [Vitale 1991]. The Orator synthesizer from Bellcore [Spiegel 1985], which attempts to divine the nationality of a name, does an outstanding job with name pronunciation. Since it can output the pronunciation as text, MailCall uses it to generate the pronunciations for names in the recognition grammar to be compiled.[31] Using Orator required minor modifications to the script for compiling grammars for Dagger.

7.4.3 Call control

Two options are available in the realm of call control. For analog telephony, a Computerfone box provides an RS-232 interface to an analog line; simple commands allow MailCall to answer the call. If using Solaris, however, one can take advantage of ISDN-based call control the XTL [Sun 1994] system from Sun. XTL enables the use of ISDN phones and provides information about when the caller has hung up so that MailCall can terminate immediately. Since ISDN does not mix the two audio channels, barge-in is possible with an end-to-end ISDN connection.

7.4.4 Parsing

The Sun Microsystems Laboratories parser, Swiftus, and the Unified Grammar specification is a boon for speech developers. It allows the developer to specify a single grammar from which BNF grammars are generated for both Dagger and Hark, so that an application can switch between them and insure that the same grammar is active. (The grammar for MailCall is described in Appendix A.) At the same time, it builds a Lisp parser which returns feature-value pairs for a given text string.

[31] Unfortunately the process is too slow to be used for the dynamic grammar updating, but by keeping a record of those names we can batch them offline in hopes of getting them right the next time around.

7.5 Run-time options

Earlier sections have hinted somewhat at MailCall's configurability: verbose prompts may be turned on or off, time-out prompts may be used or not, etc. There are additional options provided for choice of speech tools, flexibility in a distributed computing environment, and debugging.

As the recognition server and text-to-speech server provide a layer of abstraction above recognition and synthesis engines, the desired engine can be set at runtime. The -s option sets the synthesizer, and the -r option sets the recognizer. The -a option sets the host for the audio server, should one want to use the audio server on another host.

Since the recognition server and text-to-speech server can be run on any machine, MailCall needs to know which server to connect to. The -rhost and -thost options specify the host machines for the recognition and text-to-speech servers, respectively.

Clearly, the number of processes required to run recognition and synthesis can be overwhelming, especially when trying to debug a small problem. To ease the development process, MailCall can be run in four different modes. In *text-only* mode the user types a command on the keyboard, and that text is treated like a callback from the recognition server. Instead of speaking the output, MailCall prints it to the screen.[32] Where a sound file would be played, a message is printed to the same effect. Thus one can simulate the recording and sending of voice replies to check the mechanism without actually recording audio. MailCall can also be run with recognition or synthesis but not both, should the developer want to check how a prompt sounds, see if a certain error is being caught properly, etc. *Text-input* mode allows the developer to type commands as if they had been recognized by the r_server, which is useful if one wants to check the pronunciation of prompts by the tts_server. *Text-output* mode is useful for testing the recognizer, since there is no delay in waiting for the tts_server to output the audio. All three of these debugging modes proved useful in the development of MailCall, especially text mode, which made it possible to test new dialogue management schemes from home using a dialup connection.

In addition, one can toggle other aspects of the interaction on and off at runtime. Prompt tones (to signal when the synthesizer is finished speaking) can be turned on or off, barge-in can be turned on or off, and MailCall can be instructed to listen for touch-tones or

[32] MailCall's text mode proved invaluable for debugging over a dialup connection. Since MailCall runs 24 hours a day, bugs are sometimes discovered at odd hours and require swift attention.

not. If conflicting options are issued (for instance, listening for touch-tones when using keyboard input, the user is notified.

		input modality	
		speech	**text**
output modality	**speech**	normal operation	debugging text-to-speech
	text	debugging recognition	debugging discourse

Figure 7.2: run-time options for debugging.

As users develop preferences for certain system options, these could be stored in their user profiles and loaded as soon as the user is identified. This may be particularly useful in the case of speech output rate and a preference for touch tones.

8. Conclusions

This document has described MailCall, a conversational messaging system which supports random access among messages, CLUES, its underlying filtering system, and ErrorLock, an interface algorithm for managing recognition errors. This final chapter will discuss the contributions of this project and suggest future work.

8.1 Distinguishing MailCall from previous work

MailCall is unique both in its preparation and presentation of messages. CLUES enables it to present timely messages (based on short-term interests), filling a gap left by rule-based systems and memory-based reasoning filtering. Since CLUES is independent of any application program, its output can be accessed by any number of messaging tools—whether at the desktop or on the road. And since it handles voice and text messages in a unified manner, the user can expect a similar experience regardless of location or media.

MailCall's random access to messages brings the nomad a step closer to the kind of browsing and navigation one can perform at the desktop. Exercising the dynamic grammar capability of the recognizer, it opens up new possibilities for dynamic access using speech.

MailCall attempts to deal with recognition errors in a standardized way. Drawing on the power of the parser, it provides ErrorLock, a framework for handling both rejection and substitution errors and thereby suggests how future systems might be designed in order to facilitate rapid prototyping of speech applications.

8.2 Insights and Issues

Designing and implementing MailCall produced insights regarding conversational messaging systems and speech recognition systems in general.

8.2.1 Filtering

It is not enough for a filtering system to model the user's long-term interests; although that is important, there are too many temporary yet crucial interests for a user to keep track of. A filtering system can be more effective by taking advantage of "free information"—that which already exists and is readily available—instead of forcing the user to develop complicated user profiles. Indeed, such a system should have no setup effort involved (a liability of rule-based systems) and should produce results immediately (a liability of memory-based systems).

Wherever possible, all forms of messaging should be prioritized and presented similarly. A single in-box, accessible at the desktop and over the phone in the same format and with the same filtering, provides consistency and helps to develop user confidence.

HTML browsers such as Netscape and NCSA Mosaic proved to be excellent prototyping tools for developing a GUI mailer. HTML extensions such as "mailto:" suggest them as communication tools, not just browsers of static information.

Filtering like CLUES is only possible if the user's personal databases are available; proprietary data formats or encryption can foil such a system. Open data formats or pass keys for authorized applications would help to insure that user modeling programs have access to the data they need.

8.2.2 Speech as a control channel

Minimizing and managing recognition errors is central to any speech application. Much of the battle can be won up-front by providing extra guidance for beginners and help on-demand for all users. Help must be structured carefully, though, since some help sequences can be extremely long.

Regardless of recognition performance, the feedback given can do much to establish the system as a cooperative entity. Recording what the user has tried to say in the context of an error can keep the system from making distinctly non-human errors: for instance, (mis)hearing the same thing twice in a row yet not expressing surprise at having heard it a second time.

The combination of DTMF tones and speech is especially powerful; DTMF tones are generally faster and more robustly detected than speech, though speech offers a more

flexible command set and can be easier to learn. The two complement each other, and having both can give the user a feeling of flexibility.

8.2.3 Speech as feedback

Principles of linguistics, especially ellipsis, apply in the word choice for both input and output in a speech interface. One must be extremely sensitive to the conversational implicatures of an utterance, recognizing that what is intended is not necessarily what is understood. The designer must think in terms of user expectations rather than system reaction. Achieving appropriate wording can be tricky but is worth it.

Many speech interfaces (particularly IVR systems) are perceived as mechanical because their feedback never varies.[33] Keeping track of where the users have been and what they've said may inspire confidence and minimize frustration with "mechanical" interaction. The use of ellipsis can tighten sentences without sacrificing intelligibility.

As is almost always observed during the development of a speech system, interruptibility is vital. Since commands which cause the presentation of information are usually not verified, the user may get stuck listening to a long message or help sequence. Barge-in allows the user to stop the output and issue another command. DTMF tones allow the user to navigate within a long message in order to skip included text or jump to the next help option.

Perceived speed can be quite different from actual speed. Ellipsis helps MailCall from sounding more repetitive than it actually is. In addition, masking silence with courtesy prompts (as is done while waiting for mail parsing to finish) can also be effective.

8.3 Next Steps

The navigational schemes offered by MailCall should be more complete, offering location-based navigation as well as sequential and content-based. Content-based navigation itself should be expanded to include subject lines and items in the user's calendar. Using multiple attributes to reference messages may push MailCall to more of a query-based interface [Stifelman 1995] than a navigation-based one.

[33] Some would argue it important that a system strive to un-anthropomorphize itself—that mechanical feedback will help the user remember not to drop into spontaneous conversation with the system.

ErrorLock, the error-handling mechanism should be transplanted into another application to validate its claims of generality. Serious user studies are needed to evaluate how effective the verification/correction schemes are. If it turns out that users do not use corrective phrases "no, I said..." very often but simply re-issue commands, then corrective phrases could be taken out and make for a smaller grammar.

Generating feedback is complicated and can become messy in a complex application like MailCall. A tool for generating feedback such as described in [Stifelman 1995] would be very useful, especially if it could be integrated into a larger discourse model.

More than anything, MailCall the speech interface needs some serious usability data gleaned from weeks and months of use. Phoneshell, which has been in operation for a few years, has proved its worth by becoming rather indispensable to members of speech group. Whether MailCall could enjoy that kind of popularity remains to be seen; users may be put off by high error rates, no matter how good the error-handling is. Since MailCall offers touch-tone equivalents for several functions, it will be interesting to see how often they are used. Another interesting finding will be the percentage of users that take advantage of MailCall's summarization and navigation capabilities using the dynamic grammar loading, and what percentage is content to step through messages one by one.

The larger question to be answered, though, is this: what is speech recognition (in its current state) useful for? MailCall itself pushes the edge of Dagger's capabilities, and even its task is rather constrained. Speech seems to be useful for formulating queries and issuing multifaceted commands which would be difficult to represent with touch-tones (aside from using multiple keypresses and hierarchies), but is speech recognition up to the very tasks which make it worthwhile? These questions can be answered only through long-term user studies and by bringing real products to market.

Appendix A: Recognition Grammar (for Dagger)

The following describes the recognition grammar for MailCall. It is written in Backus-Naur Form (BNF) [Aho et. al. 1988] with some notation unique to Dagger. For an complete interpretation of this grammar specification, see [Hemphill 1993b]. This grammar was generated automatically from a Unified Grammar specification; similar grammars formatted for the BBN Hark recognizer and the SRI Decipher recognizer can be similarly generated. The boldface word **empty** indicates that the nonterminal will later be expanded to include lists of names or other information (see Section 5.6).

```
start(TOP-LEVEL).
export(TOP-LEVEL).
TOP-LEVEL --->   MAILCALLSENTENCE  .
export(MAILCALLSENTENCE).
MAILCALLSENTENCE --->   LOGINSENTENCE  | NAVIGATESENTENCE  |
PRESENTSENTENCE  | STATUSSENTENCE  | REPLYSENTENCE  | HELPSENTENCE  |
ACKSENTENCE  | LOGOUTSENTENCE  .
export(LOGINSENTENCE).
LOGINSENTENCE --->   NAME  | C-NAME  | PASSWORD  .
export(NAME).
NAME --->  [ this is | it ( is | _s)] SEM_USER [ again] .
export(C-NAME).
C-NAME --->  [ CORRECTIVE ] NAME  .
export(PASSWORD).
PASSWORD --->  [ ( it | is | _s) | my password is] DIGIT  DIGIT  DIGIT
DIGIT  DIGIT  DIGIT  .
export(DIGIT).
DIGIT --->   CAT_NUMBER_EQ_CARDINAL_T_LT_NUMVAL_10  .
export(CAT_NUMBER_EQ_CARDINAL_T_LT_NUMVAL_10).
CAT_NUMBER_EQ_CARDINAL_T_LT_NUMVAL_10 --->   nine | eight | seven | six
| five | four | three | two | one | zero | oh .
export(HELPSENTENCE).
HELPSENTENCE --->   HELP  | C-HELP  .
export(C-HELP).
C-HELP --->  [ CORRECTIVE ] HELP  .
export(HELP).
HELP --->   help[ me] .
export(ACKSENTENCE).
ACKSENTENCE --->   AFFIRM  | DENY  .
export(AFFIRM).
AFFIRM --->   SEM_YES  |[ that ( is | _s)] ( correct | right) .
export(SEM_YES).
SEM_YES --->   sure | yeah | yea | yes .
export(DENY).
DENY --->   SEM_NO  .
```

```
export(SEM_NO).
SEM_NO --->   nope | no .
export(LOGOUTSENTENCE).
LOGOUTSENTENCE --->   LOGOUT  | C-LOGOUT  .
export(C-LOGOUT).
C-LOGOUT --->  [ CORRECTIVE ] LOGOUT  .
export(LOGOUT).
LOGOUT --->   SEM_LOGOUT  | that _s all for now | hang up | quit .
export(SEM_LOGOUT).
SEM_LOGOUT --->   (so long) | goodbye | (bye bye) | bye .
export(CORRECTIVE).
CORRECTIVE --->   no | i said | no i said .
export(NAVIGATESENTENCE).
NAVIGATESENTENCE --->   NAVIGATE  .
export(NAVIGATE).
NAVIGATE --->   NAV-START-OVER  | C-NAV-START-OVER  | NAV-FIRST  | C-
NAV-FIRST  | NAV-SEQUENTIAL  | C-NAV-SEQUENTIAL  | NAV-CATEGORY  | C-
NAV-CATEGORY  | NAV-CLUSTER  | C-NAV-CLUSTER  | NAV-QUERY  | C-NAV-QUERY
.
export(C-NAV-START-OVER).
C-NAV-START-OVER --->  [ CORRECTIVE ] NAV-START-OVER  .
export(NAV-START-OVER).
NAV-START-OVER --->  [ erase everything and] start over from the
beginning .
export(C-NAV-FIRST).
C-NAV-FIRST --->  [ CORRECTIVE ] NAV-FIRST  .
export(NAV-FIRST).
NAV-FIRST --->   ( start with | read[ me]) the first ( message | one |
header) .
export(C-NAV-SEQUENTIAL).
C-NAV-SEQUENTIAL --->  [ CORRECTIVE ] NAV-SEQUENTIAL  .
export(NAV-SEQUENTIAL).
NAV-SEQUENTIAL --->   SEM_MOVEMENT_SET-
INTERSECT_DIRECTION_CONS_FORWARD_BACK [ message] .
export(SEM_MOVEMENT_SET-INTERSECT_DIRECTION_CONS_FORWARD_BACK).
SEM_MOVEMENT_SET-INTERSECT_DIRECTION_CONS_FORWARD_BACK --->   previous |
next .
export(C-NAV-CATEGORY).
C-NAV-CATEGORY --->  [ CORRECTIVE ] NAV-CATEGORY  .
export(NAV-CATEGORY).
NAV-CATEGORY --->  [ go[ back] to[ my | the]] SEM_FILTER-CATEGORY
messages .
export(SEM_FILTER-CATEGORY).
SEM_FILTER-CATEGORY --->   other | personal | timely .
export(C-NAV-CLUSTER).
C-NAV-CLUSTER --->  [ CORRECTIVE ] NAV-CLUSTER  .
export(NAV-CLUSTER).
NAV-CLUSTER --->   ( read[ me] | go to) the ( one | ones | message |
messages) from SEM_EMAIL-LINE  .
export(SEM_EMAIL-LINE).
SEM_EMAIL-LINE --->   empty .
export(C-NAV-QUERY).
C-NAV-QUERY --->  [ CORRECTIVE ] NAV-QUERY  .
export(NAV-QUERY).
NAV-QUERY --->   ( is there ( anything | one a ( response | reply |
message)) from | ( what | how) about) SEM_PAST-SENDER  | did SEM_PAST-
SENDER [ ever] ( say anything | ( call | write)[ back]) .
```

```
export(SEM_PAST-SENDER).
SEM_PAST-SENDER --->    empty .
export(PRESENTSENTENCE).
PRESENTSENTENCE --->   PRESENT  .
export(PRESENT).
PRESENT --->   PRES-MESSAGE  | C-PRES-MESSAGE  | PRES-SUMMARY  | C-PRES-
SUMMARY  .
export(C-PRES-MESSAGE).
C-PRES-MESSAGE --->  [ CORRECTIVE ] PRES-MESSAGE  .
export(PRES-MESSAGE).
PRES-MESSAGE --->   let me hear it | read it[ to me] | read the header[
again] | who ( ( is | _s) it from | sent ( it |[ me] that)) | what ( is
| _s) ( it about | the subject) | when did ( that | it) arrive | who (
was | is | _s) ( it | that) addressed to .
export(C-PRES-SUMMARY).
C-PRES-SUMMARY --->  [ CORRECTIVE ] PRES-SUMMARY  .
export(PRES-SUMMARY).
PRES-SUMMARY --->   read[ me] ( a | the) summary[ again] |[ tell me] (
what | which) messages are ( here | in this category) .
export(STATUSSENTENCE).
STATUSSENTENCE --->   MAN-SYSTEM  | C-MAN-SYSTEM  | MAN-MESSAGE  | C-
MAN-MESSAGE  .
export(C-MAN-SYSTEM).
C-MAN-SYSTEM --->  [ CORRECTIVE ] MAN-SYSTEM  .
export(MAN-SYSTEM).
MAN-SYSTEM --->   talk faster | increase the speaking rate | talk slower
| decrease the speaking rate .
export(C-MAN-MESSAGE).
C-MAN-MESSAGE --->  [ CORRECTIVE ] MAN-MESSAGE  .
export(MAN-MESSAGE).
MAN-MESSAGE --->   delete ( it | that | the message) .
export(REPLYSENTENCE).
REPLYSENTENCE --->   REPLY  .
export(REPLY).
REPLY --->   REPLY-SEND  | C-REPLY-SEND  | REPLY-CONFIRM  | C-REPLY-
CONFIRM  | REPLY-FORMAT  | REPLY-CALL  | C-REPLY-CALL  | REPLY-LOCATION
| C-REPLY-LOCATION  | REPLY-AREA-CODE  | C-REPLY-AREA-CODE  | REPLY-
NUMBER  | C-REPLY-NUMBER  .
export(C-REPLY-SEND).
C-REPLY-SEND --->  [ CORRECTIVE ] REPLY-SEND  .
export(REPLY-SEND).
REPLY-SEND --->  [ send[ PRONOUN ] a][ voice | text] reply .
export(C-REPLY-CONFIRM).
C-REPLY-CONFIRM --->  [ CORRECTIVE ] REPLY-CONFIRM  .
export(REPLY-CONFIRM).
REPLY-CONFIRM --->   CONFIRM-CMD [ it |[ the] message] .
export(CONFIRM-CMD).
CONFIRM-CMD --->   send | review | cancel .
export(REPLY-FORMAT).
REPLY-FORMAT --->   SEM_VOICE-FORMAT  .
export(SEM_VOICE-FORMAT).
SEM_VOICE-FORMAT --->   macintosh | (u u encode) | sun | nextmail | mime
.
export(C-REPLY-CALL).
C-REPLY-CALL --->  [ CORRECTIVE ] REPLY-CALL  .
export(REPLY-CALL).
REPLY-CALL --->   call ( SEM_USER  | PRONOUN ) .
```

```
export(SEM_USER).
SEM_USER --->   (sumit basu) | (matt marx) | (lisa stifelman) | (jordan
slott) | (jeff herman) | (deb roy) | (chris schmandt) | (bruce mchenry)
.
export(C-REPLY-LOCATION).
C-REPLY-LOCATION --->  [ CORRECTIVE ] REPLY-LOCATION  .
export(REPLY-LOCATION).
REPLY-LOCATION --->  [[ at | use][ the]] ( work | home)[ number] .
export(C-REPLY-AREA-CODE).
C-REPLY-AREA-CODE --->  [ CORRECTIVE ] REPLY-AREA-CODE  .
export(REPLY-AREA-CODE).
REPLY-AREA-CODE --->   DIGIT  DIGIT  DIGIT  .
export(C-REPLY-NUMBER).
C-REPLY-NUMBER --->  [ CORRECTIVE ] REPLY-NUMBER  .
export(REPLY-NUMBER).
REPLY-NUMBER --->  [ it | is | _s] DIGIT  DIGIT  DIGIT  DIGIT  DIGIT
DIGIT  DIGIT  .
export(PRONOUN).
PRONOUN --->   him | her .
```

Appendix B: Biographies of the Committee

Chris Schmandt is the director of the Speech Research group at the MIT Media Laboratory, where he has been building conversational computer systems for over a decade. He holds the B.S. and S.M. degrees from MIT and was a research associate in the Architecture Machine Group, where he developed (among other projects) Put That There. He has served on several program committees and is co-chair of CSCW '96.

Pattie Maes is an Assistant Professor of Media Arts and Sciences at MIT and leads the Autonomous Agents group at the MIT Media Laboratory. She holds a Ph.D. in Computer Science from the University of Brussels and was a Visiting Professor at the MIT Artificial Intelligence Laboratory before coming to the Media Lab. She has published widely on a variety of topics including adaptive interfaces, computer graphics animation, and artificial life.

Nicole Yankelovich is Co-Principal Investigator of the Speech Applications project at Sun Microsystems Laboratories in Chelmsford, MA. In addition to project management responsibilities, she specializes in designing speech user interfaces. Prior to joining Sun in 1991, Nicole worked as Project Coordinator at the Brown University Institute for Research in Information and Scholarship (IRIS). During her 9 years at IRIS, she focused on the design of integrated hypertext applications. Nicole has published a variety of papers on hypertext, user interface design, and speech applications, and she has served on the organizing and program committees of conferences such as Hypertext, CHI, UIST, and CSCW.

References

[Aho et. al. 1988] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1988.

[Arons 1991] B. Arons: "Hyperspeech. Navigating in Speech-Only Hypermedia." *Proceedings of Hypertext '91*, pp. 133–146. ACM, 1991.

[Arons 1992] B. Arons. "Tools for Building Asynchronous Servers to Support Speech and Audio Applications." *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1992.

[Bolt 1980] R. Bolt. "'Put-That-There': Voice and Gesture at the Graphics Interface." *Computer Graphics*, 14(3):262-270, 1980.

[Chalfonte et. al. 1991] B. Chalfonte, R. Fish, and R. Kraut. "Expressive Richness: A Comparison of Speech and Text as Media for Revision." *Proceedings of CHI '91*, ACM Press, May 1991. pp. 21–26.

[Coulthard 1977] M. Coulthard. *Introduction to Discourse Analysis*. Longman, 1977.

[Danis et. al. 1994] C. Danis, L. Comerford, E. Janke, K. Davies, J. DeVries, and A. Bertrand. "Storywriter: A Speech-Oriented Editor." *CHI '94 Conference Companion*, Boston April 24–28, 1994.

[Davis 1990] J. Davis. "Let Your Fingers Do the Spelling: Implicit Disambiguation of Words Spelled With the Telephone Keypad" *Proceedings of the American Voice Input/Output Society*, 1990.

[Digital 1987] DECTALK DTC101 Programmer's Reference Manual. Digital Equipment Corporation, 1987.

[Dorner 1988] S. Dorner. "Eudora: Bringing the P.O. Where You Live." Qualcomm, Inc. Copyright 1988–1992 University of Illinois Board of Trustees.

[Entropic 1995] TRUETALK Text-to-Speech Software Developer's Toolkit Version R1. Entropic Research Laboratory, Inc., 1995.

[Engelbeck & Roberts 1989] G. Engelbeck and T. Roberts. "The Effects of Several; Voice-Menu Characteristics on Menu-Selection Performance."*U.S. West Advanced Technologies Technical Report*, 1989.

[Grice 1975] H. Grice. "Logic and Conversation," *Syntax and Semantics: Speech Acts*, Cole & Morgan, editors, Volume 3, Academic Press, 1975.

[Goldberg et. al. 1992] D. Goldberg, D. Nicols, B. Oki, and D. Terry. "Using Collaborative Filtering to Weave an Information Tapestry." *Communciations of the ACM*, 35(12): pp 61–70, 1992.

[Grosz & Sidner 1986] B. Grosz and C. Sidner. "Attention, Intentions, and the Structure of Discourse." *Computational Linguistics*, 12(3):175–203, 1986.

[Hark 1993] HARK Prototyper User's Guide. BBN Systems and Technologies: A Division of Bolt Beranek and Newman Inc., 1993.

[Hayes & Reddy 1983] P. Hayes and D. Reddy: "Steps toward graceful interaction in spoken and written man-machine communcation." *Interanational Journal of Man-Machine Studies*, 19:231–284, 1983.

[Hemphill 1993a] C. Hemphill. "DAGGER: a parser for Directed Acyclic Graphs of Grammars Enhancing Recognition." Texas Instruments Inc., 1993.

[Hemphill 1993b] C. Hemphill. "FLANGE: Formal LANguage Grammars for Everyone." Texas Instruments Inc., 1993.

[Hemphill 1993c] C. Hemphill. "EPHOD: an Electronic PHOnetic Dictionary and tool set." Texas Instruments Inc., 1993.

[Hemphill & Thrift 1995] C. Hemphill & P. Thrift. "Surfing the Web by Voice." Submitted to *ACM Multimedia '*95, San Francisco, CA, Nov. 5–9, 1995.

[Herman 1995] J. Herman. "NewsTalk: A Speech Interface to a Personalized Information Agent." MIT Master's Thesis, Program in Media Arts and Sciences, June 1995.

[Hirschman & Pao 1993] L. Hirschman and C. Pao. "The Cost of Errors in a Spoken Language System." Presented at the 3rd European Conference on Speech Communication and Technology, September 21–23, 1993, Berlin, Germany.

[Kamm 1994] C. Kamm. "User Interfaces for Voice Applications," *Voice Communication Between Humans and Machines*, National Academy Press, Washington, DC, 1994.

[Kitai et. al. 1991] M. Kitai, A. Imamura, and Y. Suzuki. "Voice Activated Interaction System Based on HMM-based Speaker–Independent Word Spotting," *Proceedings of the Voice I/O Systems Applications Conference*, Atlanta, GA, September 1991.

[Lashkari et. al. 1994] Y. Lakarshi, M. Metral, and P. Maes. "Collaborative Interface Agents." *Proceedings of AAAI '94*.

[Ly 1993] E. Ly. "Chatter: A Conversational Telephone Agent" MIT Master's Thesis, 1993.

[Ly et. al. 1993] E. Ly, C. Schmandt, and B. Arons. "Speech Recognition Architectures for Multimedia Envronments." *Proceedings of the American Voice Input/Output Society*, San Jose, 1993

[Ly & Schmandt 1994] E. Ly and C. Schmandt. "Chatter: A Conversational Learning Speech Interface" *Proceedings of AAAI Spring Symposium on Intelligent Multi-Media Multi-Modal Systems*, March 1994.

[Malone et. al. 1987] T. Malone, et. al. "Semi-Structured Messages are Surprisingly Useful for Computer-Supported Coordination." *ACM Transactions on Office Information Systems*, 5(2): 115–131, 1987.

[Malone et. al. 1989] T. Malone, R. Grannat, K-Y. Lai, R. Rao, and D. Rosenblitt. "The Information Lens: An Intelligent System for Information Sharing and Coordination." *Technological Support for Work Group Collaboration*, M. Olson (ed), 1989.

[Martin & Kehler 1994] P. Martin and A. Kehler. "SpeechActs: A Testbed for Continuous Speech Applications," *AAAI-94 Workshop on the Integration of Natural Language and Speech Processing*, 12th National Conference on AI, Seattle, WA, July 31–August 1, 1994.

[Marx & Schmandt 1994a] M. Marx and C. Schmandt. "Reliable Spelling Despite Unreliable Letter Recognition." *Proceedings of the 1994 Conference*. San Jose, CA: American Voice I/O Society, September 1994.

[Marx & Schmandt 1994b] M. Marx and C. Schmandt. "Putting People First: Specifying Names in Speech Interfaces." *Proceedings of the ACM Symposium on User Interface Software and Technology*. Marina del Rey, California, November 2–4, 1994.

[Metral 1992] M. Metral. "A Generic Learning Interface Agent." S.B. thesis, Department of Electrical Engineering and Computer Science, MIT, 1992.

[Muller & Daniel 1990] M. Muller and J. Daniel. "Toward a definition of voice documents." *Proceedings of COIS*, 1990.

[Orwant 1991] J. Orwant. "Doppelgänger goes to School: Machine Learning for User Modeling." M.S. Thesis, Program in Media Arts and Sciences, MIT, 1991.

[Pisoni et. al. 1985] D. Pisoni, H. Nusbaum, and B. Greene. "Perception of Synthetic Speech Generated By Rule." *Proceedings of the IEEE*, 73(11), pp. 1665–1676, 1985.

[Postman 1992] N. Postman, *Technopoly: The Surrender of Culture to Technology*. Alfred A. Knopf, New York, 1992.

[Rau & Skiena 1994] S. Rau and S. Skiena. "Dialing for Documents: An Experiment in Information Theory." *Proceedings of the ACM Symposium on User Interface Software and Technology*. Marina del Rey, California, November 2–4, 1994.

[Salton 1981] G. Salton. "The SMART environment for retrieval system evaluation–advantages and problem areas." In *Information retrieval experiment*, K. Sparck Jones, editor. London, Butterworth, 1981.

[Schmandt 1984] C. Schmandt. "Speech Synthesis Gives Voiced Access to an Electronic Mail System." *Speech Technology*, Aug/Sept 1984, pp. 66–68.

[Schmandt 1985] C. Schmandt. "A Conversational Telephone Messaging System." *IEEE Transactions on Consumer Electronics*, August 1985.

[Schmandt 1986] C. Schmandt. "A Robust Parser and Dialog Generator for a Conversational Office System." *Proceedings of the 1986 Conference*, pp. 355–365. San Jose, CA: American Voice I/O Society, September 1986.

[Schmandt 1987] C. Schmandt. "Conversational Telecommunications Environments." *Second International Conference on Human-Computer Interaction*, 1987.

[Schmandt & Casner 1989] C. Schmandt and S. Casner. "Phonetool: Integrating Telephones and Workstations." *IEEE Communications Society*, November 27-30, 1989.

[Schmandt 1993] C. Schmandt. "Phoneshell: the Telephone as Computer Terminal" *Proceedings of ACM Multimedia Conference*, August 1993.

[Schmandt 1994a] C. Schmandt. "Multimedia Nomadic Services on Today's Hardware." *IEEE Network*, September/October 1994.

[Schmandt 1994b] C. Schmandt. *Voice Communication with Computers: Conversational Systems*, Van Nostrand Reinhold, New York, 1994.

[Schmandt & Slott 1995] C. Schmandt and J. Slott. "An Asynchronous Audio Server." MIT Media Lab Technical Report, 1995.

[Spiegel 1985] M. Spiegel. "Pronouncing Surnames Automatically", *Proceedings of the 1985 Conference*. San Jose, CA: American Voice I/O Society, September 1985.

[Stanfill & Waltz 1986] C. Stanfill and D. Waltz. "Toward Memory-Based Reasoning." *Communications of the ACM*, 29:12 1986.

[Strong 1993] B. Strong. "Casper: Speech Interface for the Macintosh." *Proceedings of EuroSpeech*, Berlin, Germany, 1993.

[Stifelman 1991] L. Stifelman. "Not Just Another Voice Mail System." *Proceedings of the 1991 Conference*. San Jose, CA: American Voice I/O Society, September 1991.

[Stifelman 1992] L. Stifelman. "VoiceNotes: An Application for a Voice-Controlled Hand-Held Computer." Master's Thesis, Massachusetts Institute of Technology, 1992.

[Stifelman et. al. 1993] L. Stifelman, B. Arons, C. Schmandt, and E. Hulteen. "VoiceNotes: A Speech Interface for a Hand-Held Voice Notetaker, *ACM INTERCHI '93 Conference Proceedings*, Amsterdam, The Netherlands, April 24–29, 1993.

[Stifelman 1993] L. Stifelman. "User Repairs of Speech Recognition Errors: An Intonational Analysis." Manuscript. 1993.

[Stifelman 1995] L. Stifelman. "A Tool to Support Speech and Non-Speech Audio Feedback Generation in Audio Interfaces." MIT Media Laboratory Technical Report, October 1994.

[Sun 1994] SUNLINK ISDN 1.0 Software Reference Manual. Sun Microsystems Inc., 1994.

[Van den Berg 1993] S. R. Van den Berg. Procmail program. Available from ftp.informatik.rwthaachen.de (137.226.112.172), 1993.

[Vitale 1991] Vitale, T. "An Algorithm for High Accuracy Name Pronunciation by Parametric Speech Synthesizer", *Journal of Computational Linguistics*, 17(1), pp. 257–276, 1991.

[Wall & Schwartz 1992] L. Wall and R. Schwartz *Programming Perl*. O'Reilly & Associates, 1992.

[Wildfire 1995] Wildfire Communications, Inc. Lexington, MA, 1995.

[Winograd & Flores 1987] T. Winograd and F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*. New York, Addison-Wesley, 1987.

[Wong 1991] C. Wong. "Personal Communications." MIT Master's Thesis, Program in Media Arts and Sciences, 1991.

[Yankelovich 1994] N. Yankelovich. "Talking vs. Taking: Speech Access to Remote Computers." ACM *CHI '94 Conference Companion*, Boston, MA, April 24-28, 1994.

[Yankelovich & Baatz 1994] N. Yankelovich and E. Baatz. “SpeechActs: A Framework for Building Speech Applications.” *Proceedings of the American Voice Input/Output Society*, San Jose, 1994.

[Yankelovich et. al. 1995] N. Yankelovich, G. Levow, and M. Marx. “Designing SpeechActs: Issues in Speech Interfaces.” *Proceedings of CHI ‘95*, Denver, CO, May 8–11, 1995.

[Zellweger 1988] P. Zellweger. “Scripted documents: A hyermedia path mechanism.” *Proceedings of Hypertext ‘89*, pp. 1–14. ACM, 1989.

[Zimbardo 1985] P. Zimbardo, *Psychology and Life*. Scott, Foresman and Company, Glenview, Illinois, 1995, pp. 308–309.

[Zue et. al. 1994] V. Zue, S. Seneff, J. Polifroni, M. Phillips, C. Pao, D. Goddeau, J. Glass, and E. Brill. "Pegasus: A spoken Language Interface for On-Line Travel Planning.” Presented at the ARPA Human Language Technology Workshop, Merrill Lynch Conference Center, Princeton NJ, March 6–11 1994.