Clique: Perceptually Based, Task Oriented Auditory Display for GUI Applications

Peter Parente

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2008

Approved by:

Gary Bishop, Advisor

Prasun Dewan, Co-principal Reader

Gary Marchionini, Reader

Russell Taylor II, Reader

Stephen Weiss, Reader

Abstract

Peter Parente: Clique: Perceptually Based, Task Oriented Auditory Display for GUI Applications.
(Under the direction of Gary Bishop.)

Screen reading is the prevalent approach for presenting graphical desktop applications in audio. The primary function of a screen reader is to describe what the user encounters when interacting with a graphical user interface (GUI). This straightforward method allows people with visual impairments to hear exactly what is on the screen, but with significant usability problems in a multitasking environment. Screen reader users must infer the state of on-going tasks spanning multiple graphical windows from a single, serial stream of speech.

In this dissertation, I explore a new approach to enabling auditory display of GUI programs. With this method, the display describes concurrent application tasks using a small set of simultaneous speech and sound streams. The user listens to and interacts solely with this display, never with the underlying graphical interfaces. Scripts support this level of adaption by mapping GUI components to task definitions. Evaluation of this approach shows improvements in user efficiency, satisfaction, and understanding with little development effort.

To develop this method, I studied the literature on existing auditory displays, working user behavior, and theories of human auditory perception and processing. I then conducted a user study to observe problems encountered and techniques employed by users interacting with an ideal auditory display: another human being. Based on my findings, I designed and implemented a prototype auditory display, called Clique, along with scripts adapting seven GUI applications. I concluded my work by conducting a variety of evaluations on Clique. The results of these studies show the following benefits of Clique over the state of the art for users with visual impairments (1-5) and mobile sighted users (6):

1. Faster, accurate access to speech utterances through concurrent speech streams.

2. Better awareness of peripheral information via concurrent speech and sound streams.

3. Increased information bandwidth through concurrent streams.

4. More efficient information seeking enabled by ubiquitous tools for browsing and searching.

5. Greater accuracy in describing unfamiliar applications learned using a consistent, task-based user interface.

6. Faster completion of email tasks in a standard GUI after exposure to those tasks in audio.

Acknowledgments

I would like to thank all of the following people and organizations for their contributions to my research. Without them, this work would not exist.

- The National Science Foundation and IBM, for funding my studies and research.
- Gary Bishop, Prasun Dewan, Gary Marchionini, Russ Taylor, and Steve Weiss for serving on my committee.
- Mary Whitton, for reviewing, commenting on, and approving my many user study proposals.
- Chris Wiesen, for helping with the statistical analysis of my study data.
- John Grose, for teaching an enlightening audiology course and taking the time to review my simplistic chapter about the human auditory system.
- All of the participants in my user studies, for investing time and energy in my research.
- Jason Morris, Leeanne Brown, and Barbara Riverdahl for openly sharing how they live and work.
- Gary Bishop, for advising me on all things both great and mundane, for sharing his endless energy and compassion for people, and for motivating me with opportunities to do good.
- My mother, father, and brother, for always supporting my endeavors, even when they took me far from home and their love.
- Jackie, my wife, for her loving commitment, even when this work drew me from her.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The development of the graphical user interface (GUI) at Xerox PARC in the 1970's and 1980's changed the face of the personal computer. Within ten years, the pixel-based desktop environment made popular by Apple's Macintosh operating system began to replace the character-based command line on many workstations. Graphical user interfaces started to appear in a number of popular applications including word processors, spreadsheets, database front-ends, and communication clients. The benefits of the graphical desktop environment, such as direct manipulation and iconic representation (Shneiderman, 1992), reduced the level of expertise required to operate a computer for many people. Office workers with little technical experience joined programmers, engineers, and scientists as typical computer users.

Today, the majority of human-computer interaction occurs through graphical user interfaces. Every major personal computing platform features at least one graphical desktop environment capable of running hundreds, possibly thousands, of readily-available GUI programs. Machines, such as televisions, cash registers, public kiosks, and even the kitchen refrigerator, once dependent on physical interfaces, have been re-engineered or augmented with point-and-click graphical interfaces. The strong prevelance of GUIs in daily tasks has turned people of all ages and disciplines into users: children, adults, home users, office workers, and computer gurus alike. Twenty years after its commercial debut, the GUI and its users are nearly ubiquitous.

1.1 The Problem

Yet while graphical user interfaces have been successful in many circumstances, the strong push for GUI development has actually disadvantaged some users and stymied useful alternatives. In the early 90s, concerns were raised about the accessibility problems resulting from the rapid replacement of command line programs with GUI equivalents (Boyd et al., 1990). Prior to the advent of the commercial GUI, people with visual impairments enjoyed effective access to computers with the help of an assistive technology called a *screen reader*. Because the visual display was simply an 80 by 24 array of characters, a screen reader could easily extract meaningful text from the video buffer, and then speak it aloud using a speech synthesizer or convert it to Braille. What users with visual impairments heard or felt was equivalent to the lines of text that a sighted user saw.

GUI applications rendered the strategy of mining the video buffer for human-readable text practically useless. A grid of pixel color values supplanted the easily accessible array of characters. Bitmapped graphics that could be moved, obscured, and resized at runtime now filled the screen, not fixed lines of text easily reported by a screen reader. Explicit support from the windowing system or applications themselves was now required to interpret the content of the visual display.

As a result of this drastic change, screen reader developers were forced to find new ways of obtaining and reporting information about applications to users (Blenkhorn and Evans, 2000). The first screen readers for desktop environments treated GUI windows as if they were simply miniature screens of text: a familiar concept for command line users (Thatcher, 1994). As GUIs grew more complex in both content and layout, this guise became difficult to maintain. Developers struggled to find other ways to represent the screen, and eventually settled on a surprising solution. Instead of creating another mapping from on-screen elements to familiar audio forms, developers decided to expose GUI elements directly in audio. That is, the primary operation of modern GUI screen readers involves naming visual widgets, describing their state, and echoing events all within a narrow point of regard, often the keyboard input focus. For instance, a typical screen reader might report the selection of a menu item with speech saying, *menu item, Save, disabled, Control-S*.

While the approach of describing exactly what appears on the screen grants non-visual access, mimicking a successful visual paradigm does not guarantee equal success in audio. In fact, the decision to provide a thin auditory facade has established numerous long-standing usability problems in GUI screen readers (Barnicle, 2000; Edwards, 1991). To this day, users with visual impairments must deal with notions such as scrolling text boxes to hear clipped text, unfolding drop-down lists hidden to save screen space, navigating the spatial layout of controls they cannot see, and using tools built primarily for sighted, mouse wielding users such as drag-and-drop. Furthermore, prime benefits of the multitasking, graphical environment such as peripheral information channels and rapid task navigation are lost in the translation to a narrow, auditory point of regard.

The momentum of GUI development has similarly influenced the design of portable computing devices. Personal digital assistants (PDAs) and cell phones are designed for use away from the office environment and often invite use while a person is mobile: walking, exercising, riding, driving, and so on. Desirable qualities of these devices include a small form factor, long battery life, and ease of interaction. Surprisingly, nearly every mobile device on the market today relies on graphical display which limits the minimum size of the device, drains the battery when active but not in view, and makes use of the device difficult or dangerous while a user is moving. Features like Web browsing and text messaging are quite common on these devices, but wholly unusable in many situations where a mobile user is likely to be.

User interfaces better suited to both screen readers and mobile devices are not hard to imagine. For instance, auditory displays designed around strengths and weaknesses of human hearing could prove fruitful in either situation. Nevertheless, developers have been reluctant to embrace such novel methods of interaction. Twenty years of building, testing, and improving graphical user interfaces represents a significant investment of resources. It is neither easy nor desirable to cast off the tools, techniques, and experience gained from this work in favor of creating wholly different, though potentially useful, audio solutions.

Two forces are in conflict here. On one hand, there is a desire to explore novel interface designs, particularly in the audio domain, to better support the growing populations of visually impaired (Vanderheiden, 1990) and mobile computer users. On the other hand, there is a devotion to the familiar graphical desktop metaphor and the years of work invested in improving its utility. A problem, therefore, is balancing these goals so that developers may create more usable, innovative audio user interfaces without sacrificing past efforts spent building mainstream GUI applications.

At present, systems that create audio adaptations of GUI programs lie close to the GUI-devotion extreme. They favor a least-effort translation from on-screen visuals to audio while paying for it with severely decreased usability. On the other hand, dedicated audio applications written from the ground-up lie at the best-usability extreme. They sacrifice nearly all of the effort invested in GUI development in favor of improving the usability of the audio interaction. A solution that lies closer to the mid-point of these two extremes is desired.

1.2 A Potential Solution

The result of this research is Clique, a software system that adapts GUI applications to an auditory display designed for a listening, working user. Clique presents the user with the tasks supported by GUI programs, not their visual representations, in a manner conducive to auditory perception and processing. The user interacts solely with the auditory display while Clique takes charge of inspecting and controlling the underlying programs via their GUIs. Representing common interactions between programs and users in a mode-independent way allows Clique to create a unified audio experience across a diverse set of applications. This ability requires no modifications to the original programs, only the construction of scripts binding task definitions to widgets in a GUI.

More specifically, Clique differs from previous attempts to adapt GUI applications to audio in the following respects.

Adapting the Model Existing audio adaptation solutions change the domain in which information is rendered, but not the form of the information. For example, a screen reader describes a check box widget as a check box with or without a check in it. Likewise, a screen reader describes the operation of switching between programs in audio in terms of minimizing, maximizing, opening, and closing windows.

Clique changes both the rendering domain and the form of the rendered information as appropriate to auditory display. For example, a graphical check box is more naturally understood as a yes or no question in audio than as a square with a check mark in it. Similarly, switching between programs is more easily accomplished in audio by referencing a task by name rather than modifying the visual state of an unseen window.

Designing for the Listener Audio adaptations have traditionally provided output via a single stream of synthesized speech. Modern systems also replace some common speech utterances with an unstructured set of everyday sounds called *auditory icons*.

These designs, however, assume that one narrow bandwidth audio channel is a sufficient replacement for the vastly wider bandwidth of the screen. These systems have yet to consider the potential benefits of many other techniques described in the research literature such as simultaneous audio streams and three-dimensional sound positioning.

Clique integrates a gamut of audio rendering techniques including multiple simultaneous synthesized voices, natural sounds called *auditory icons*, short musical motives called *earcons*, and persistent environmental sounds located in a three-dimensional sound space. Each technique serves some purpose in the auditory display including, but not limited to, representing task state, indicating information type, and summarizing complex content. Concurrent streams, in particular, provide peripheral reports about inactive tasks in a multitasking environment. While Clique uses a variety of techniques, the design remains mindful of the strengths and weaknesses of each with respect to the information to be presented and the characteristics of human audition.

Designing for the Worker Current audio adaptation systems observe and report the interaction between a user and an application. If the application is accessible, all of features available on the GUI desktop are made available to the user in audio. Little is done by the adaptation system to enhance or filter these features in any way. Inherent in this design is the assumption that the tools available to a visual user are sufficient and necessary for a listening user. The striking differences between the human visual and auditory senses suggest that this presumption is incorrect.

To name just one example, seeing is limited to a field of view with visual objects potentially obscuring one another along lines of sight. The ability of a user to position a search dialog independently of the content searched is, therefore, critical to the usability of the search feature. Hearing, on the other hand, is limited by distance with sound sources potentially masking one another according to their number and common attributes. The timing, concurrency, and similarity of audible reports made while the user is searching are the critical factors usability factors for an auditory search feature.

Instead of exposing the features of the GUI desktop environment, Clique exposes its own set of tools designed to support the unique needs of the listening user. It provides substantial aids for memory and navigation to offset the problems of transience and invisibility associated with auditory information display (Lai and Yankelovich, 2003).

Supporting Common Tasks The earliest audio adaptation systems were built for expert users with visual impairments, often computer programmers who needed to "hear the screen" in order to complete tasks like formatting code and building software with visual interfaces. Today, such experts are a small minority of the total user population, yet such systems cater disproportionately to their uncommon needs. Less attention to minute visual details and more focus on facilitating common tasks might better serve today's less technical, emailing, Web browsing, chatting user.

Clique makes common interactions like searching long lists, summarizing large bodies of text, and browsing structured documents more efficient in audio by sacrificing details about the exact appearance of information on the screen. For tasks that do not require an intimate understanding of the visual display, this approach eliminates non-essential interactions and feedback, and allows users to concentrate on information relevant to their goals instead of the appearance of that information in a visual interface.

Bridging Adaptation and Dedication At present, there is a disconnect between the approaches of adapting GUI applications to audio and creating dedicated audio applications from scratch. For instance, resources invested in making a GUI application more accessible to a screen reader do not benefit future attempts to integrate a dedicated auditory display into the same application. This inability to transfer development effort from the former approach to the latter makes moving beyond basic audio adaptation to the creation of rich, program-specific audio displays costly and unattractive.

Clique allows external audio adaptations for GUI programs to later become part of the original application. To make a GUI application accessible via Clique, a developer describes the tasks available in that application in terms of the interactions needed to complete them. The developer then associates this task description with the components to be inspected and manipulated in the GUI. However, binding tasks to GUI widgets is a convenience, not a requirement. A developer could create richer, more direct channels of communication between the model of the program and the auditory display provided by Clique thus bypassing, supplementing, or completely eliminating the GUI.

For example, a Clique script for a chat program might initially inspect and manipulate the chat application through its GUI. If the chat client developer later decides to support auditory display directly, he or she could update the original Clique script to have direct communication with the chat application model. The bindings from the auditory view to the chat application data model would change, but the tasks and auditory display experienced by the user could remain untouched.

Focusing on Output Researchers have sought an effective, error-free means of supporting speech input to computer for years. This endeavor has drawn so much attention that the term *audio user interface* now connotes talking to a computer. Perfect speech recognition is difficult problem to crack, however, and solving it does little to improve the interaction between a mobile or blind user and a device with a graphical display. The form of the output from the machine has a greater effect on usability in both situations.

For this reason, Clique focuses on improving how an application is represented in audio and what functions are available to explore and control it. The physical method used to activate those functions is not a primary concern. Any device that can support the command set can be used for input: a standard keyboard, a one-handed keyboard, a wireless keypad, or, indeed, even voice input. To continually highlight this focus on output over input, the term *auditory display* is used most often throughout this work when describing Clique.

1.3 Thesis and Approach

The following questions highlight the problems I tackled in my research. Their answers, particularly those involving user evaluation, lend support to my thesis:

> Adapting GUI applications to an auditory display that describes concurrent user tasks via multiple streams of spatialized speech and sound provides a better user experience than a single stream narration of the screen. Such an approach can improve the awareness and effectiveness of people with visual impairments working in a multitasking environment, benefit sighted users working away from a graphical desktop, and function across diverse applications using third-party scripts.

What is the state of the art in designing auditory displays and how might it be improved? I reviewed the current literature on designing auditory displays both for audio aware applications and as agents for audio unaware programs. I studied the metaphors and audio rendering techniques used to create the interfaces in both kinds of systems. I chose to further the integration of the variety of audio rendering techniques discussed in the literature, specifically concurrent audio streams, in an attempt to improve auditory display usability. I also chose to develop methods of adding auditory displays to audio unaware applications using accessibility interfaces and task-oriented scripts in order to keep development costs low.

How must the auditory display support common computing tasks? I studied the research on user information seeking, text editing, and mediation behaviors to develop a list of interactions the display must support. In addition, I performed a qualitative user study involving thirteen sighted and six blind participants. During this study, I observed their methods for completing tasks in four office productivity applications through an ideal auditory display: an expert human user. The results of the study indicate that an auditory display must provide significant memory aids, guiding prompts, content summaries, peripheral awareness, and effective searching and filtering tools.

How must the auditory display account for user audition? I studied the literature on sound detection, perception, and attentive processing to build a model of the listening user. I used this model to build recommendations for applying auditory rendering techniques such as synthesized speech, auditory icons, spatial sound, and simultaneous streams to satisfy the working user needs. The theory of auditory scene analysis (Bregman, 1990) plays a crucial part in these guidelines, particularly in suggesting that concurrent audio streams can increase the efficiency of user interaction over that of the single-talker auditory displays common today.

How does a software auditory display support the needs of the user as a worker and listener? I designed and implemented an auditory display called Clique to satisfy the usability requirements previously identified. Clique provides a system for organizing interaction based on user tasks, presents concurrent audio streams to reduce the time a user spends waiting to hear pertinent information and to improve awareness of events happening outside of the current task, uses continuous environmental sounds as persistent reminders of the user's current context, mixes speech and non-speech sounds to provide details and summaries of content, implements a "tape recorder" type function to offload the burden of memorization onto the computer, and enables fast switching between browsing and searching with a non-modal, type-ahead find function.

Can a GUI adaptation method account for user, task, and developer requirements? Yes. I devised a method for separating how tasks are implemented in a particular GUI application and how they should be presented in audio. I implemented methods for accessing common GUI patterns via an accessibility toolkit on the Windows platform and adapters allowing interaction with them through the auditory display. This model-view separation allows a script developer to access information provided by an application through any public channel, in any form, and expose it in an appropriate, unified manner through the Clique auditory display.

Does Clique benefit blind users working in a multitasking environment? Yes. I analyzed the timings of utterances in the Clique concurrent stream design as well as the timings in a single stream display (e.g., screen reader). The proofs I developed show that individual utterances play sooner and repeat sooner in a concurrent design than in a serial one. An evaluation of the ability of listeners to report target and non-target utterances played concurrently revealed that not only did utterances play sooner, but that users with visual impairments were able to accurately hear more information sooner using concurrent streams rather than a serial stream.

In addition, I performed a heuristic evaluation of the peripheral speech and sound streams in Clique. This study revealed that these ambient streams provide useful information about related and unrelated tasks in an effective manner. In contrast, the single stream narration found in screen readers is incapable of providing similar awareness without risk of needlessly interrupting the user's primary task, allowing the peripheral information to become stale, or letting the user interrupt the ambient report without even realizing it.

Does Clique benefit users with visual impairments in tasks already facilitated by screen readers? Yes, for certain tasks. I conducted a user study comparing the performance of eight participants with visual impairments working on tasks with Clique and JAWS, the most popular commercial screen reader available today. The results of the study show users hear more information sooner, locate search results faster, and provide more accurate descriptions of newly learned tasks with Clique than with JAWS. Users with years of JAWS experience fare equally well using Clique to complete office work after only a few hours of practice, even when the tasks do not require peripheral information reported only by Clique. Responses to ease of use and usefulness surveys show positive correlations between Clique features and tasks such as concurrent speech and searching, searching and learning, and learning and working.

Does Clique confer benefits on sighted users? Yes. I conducted a user study comparing the performance of thirteen sighted participants completing tasks using a standard GUI email program after a brief hiatus from a workspace conducive to visual computing. In three separate trials, I varied the level of control participants had over Clique before they returned to the graphical desktop: no access, minimal commands, and all commands. The results show a significant reduction in the time spent completing tasks in the GUI after having minimal and full control over Clique. A Clique feature allowing the user to "record" spoken email passages and later see them highlighted in the GUI was critical to the reduction in time in both conditions. Ratings on questionnaires also highlight the subjective differences between having minimal and full access in terms of usefulness, ease of use, and workload.

Is the adaptation approach feasible? Yes. I analyzed the size and complexity of four application scripts I targeted for user evaluation, plus three additional scripts implemented after the fact. The numbers indicate Clique scripts are far smaller than the code for both simple and complex GUI applications implemented from scratch. I also studied the trends in the implementation of Clique components for information display, adaptation of GUI widgets, and sequences of commands executed on GUIs. The data suggest most components are reusable within and across scripts, with a few components appearing very often. Still, a long-tail of custom components exists requiring code specific to particular scripts. The extensibility of Clique was sufficient to handle these cases. Furthermore, I noted that the design of Clique is not adverse to reverting to screen reading when scripts are not available for whole applications or specific application tasks.

1.4 Overview

The next three chapters of this dissertation define the requirements of an effective auditory display. Chapter 2 reviews existing systems, their models of user interaction and techniques for presenting information aurally. The chapter concludes with a short assessment of user and developer needs, and a choice of methods for further study. Chapter 3 introduces the behaviors of the working user that an auditory display for office computing tasks must support. This chapter includes findings from a study of user interaction with an ideal auditory display (i.e., another human). Chapter 4 provides a basic review of how humans detect, perceive, and process sound based on human physiology and the theory of auditory scene analysis. This chapter concludes with recommendations on how to support the ways users work with respect to how they hear.

Chapter 5 introduces Clique, my implementation of the recommendations developed in the prior chapters. The first half of this chapter describes the Clique user experience. Explanations of how Clique satisfies the user requirements stated earlier are the heart of this section. The second half describes the software architecture of Clique, one that allows Clique to present a unified auditory display across disparate applications.

Chapter 6, probably the most important in this work, reports the results of numerous evaluations of Clique. A mathematical model shows the potential reduction in time to hear information presented in a multi-channel display versus a single stream display. A heuristic evaluation reports on the benefits and shortcomings of the peripheral streams in Clique. A summative evaluation by people with visual impairments tests the effectiveness of the concurrent speech, semi-modal searching, and task-based structure of Clique in comparison with the features of JAWS. A second summative study tests the value of using Clique during brief departures from a machine with a graphical display. A final analysis shows the feasibility of adapting applications to work with Clique with respect to extensibility and development effort.

The final chapter suggests directions for future work based on the benefits and the shortcomings of the Clique system. Topics such as automating the adaptation process and integrating task-based and screen-based auditory displays are discussed.

Each chapter in this work builds on information presented in earlier chapters. I advise that readers looking to glean the maximum amount of information from this work read the chapters in order. Nevertheless, readers willing to accept that justification for certain statements may appear in prior chapters may read select chapters in isolation. In many cases, I have included explicit references to chapters and sections where readers may find such validation.

1.4.1 Example Audio

In this document, I have tried to give clear textual descriptions and visual depictions of inherently auditory concepts. Yet I believe nothing can do them justice as well as auditory recordings. To this end, I have posted numerous sound and speech examples at `http://www.cs.unc.edu/~parente/` to complement this document. The sounds linked from that page are listed with references to various sections, figures, tables, and pages in this static document.

Chapter 2

Audio Computing Today

The purpose of this chapter is to examine the state of the art in designing software auditory displays. The first section reviews the building blocks of audio displays: techniques of rendering information in the auditory domain. Synthesized speech, audio icons, earcons, ambient sounds, concurrent streams, and spatialization methods are discussed. The second section describes common metaphors such as narration, direct manipulation, and conversation for combining these techniques into coherent displays. The third section reviews numerous systems with built-in support for audio output as well as systems that add auditory displays to existing, audio unaware programs. The chapter culminates with an analysis of the systems studied along shared dimensions. The results are used to select methods for further research based on their potential for minimizing development cost and maximizing usability of auditory displays for business applications.

The content in this chapter reviews a wide variety of auditory displays, not just those within the immediate problem domain. The intention is to study diverse uses of audio in order to identify methods that may enable more effective non-visual access to GUI applications.

2.1 Auditory Rendering

Modern computers are quite capable of producing rich audio output. Consumer-grade desktop sound cards have the ability to simultaneously output over one-hundred high quality (96 kHz, 24-bits resolution), filtered sounds through multiple speaker channels. Integrated sound chips for mobile devices, while not as advanced, are still quite capable of mixing multiple, high quality waveform sounds for stereo output. Furthermore, nearly all sound hardware is able to synthesize Musical Instrument Digital Interface (MIDI) music from sampled instrument banks. These means have been combined in many interesting and useful ways to convey information in audio.

2.1.1 Synthesized Speech

Synthesized speech is a sampled approximation of human speech generated from a source text. There are two classic methods for synthesizing speech (Binding et al., 1990). In *concatenative synthesis*, a speech engine segments the source text into base components, hashes the components into a database of pre-recorded utterances, and concatenates the utterances to produce the desired output. The granularity of the segments, the hashing algorithm, and the database size may vary, but all concatenative systems follow this general approach. With a large database of high quality utterances, the synthesized output can be almost indistinguishable from true human speech at normal speaking rates. This naturalness is the primary benefit of concatenative synthesis.

Concatenative synthesis is not without its problems, however. At high speech rates, the slight mismatches in frequency and intensity at points of concatenation become audible. These glitches detract from the naturalness of the speech and have the potential to harm its intelligibility. Qualities of the voice used to produce the speech are predetermined by the utterance database. Synthesizing speech with one male and one female voice, for example, requires two separate speech databases. The size of the database may preclude the use of concatenative synthesis on devices with limited storage and memory. For instance, one high-quality AT&T Natural Voices speech database requires roughly 700 megabytes of on-disk storage and an average of 40 MB of memory during synthesis. As a result, concatenative synthesis is most attractive for applications that prize natural-sounding output over high rates of speech, flexibility in voicing, and system resources. Telephone voice response systems and public kiosks are two examples of systems best served by concatenative synthesis where casual users are quick to judge usefulness based on surface impressions (Duggan and Deegan, 2003).

The second method, *formant synthesis*, renders speech using a model of time-varying frequency bands in human speech spectra. The resulting speech is highly accurate at all speaking rates. Parameters such as head-size, aspiration, frication, gender, pitch fluctuation, and so on may be defined and configured to produce a practically infinite number of different voices. For instance, child, adult, and elderly sounding voices of both genders can be synthesized using the same model. These same parameters can be manipulated by the engine during synthesis to imbue speech with different prosodic or intonational qualities. A question like "Are you sure?" might be intoned as being very serious by changing the emphasis, rhythm, and pitch contour of the sentence. Since no speech database is required for synthesis, storage and memory requirements are relatively low. As a result, flexibility and small footprint are the main benefits of formant synthesis.

Still, formant synthesis is not perfect. To enable real-time generation, the model used by formant engines is an imperfect approximation. The speech produced is robotic and unnatural sounding when compared with real human speech. Therefore, this method is best used in applications that require extreme speaking rates, multiple distinct voices, and low overhead. Mobile devices and screen readers are two kinds of systems that benefit from formant synthesis. Realism is sacrificed simply to enable auditory interaction in the former case and to provide increased rates of output in the latter.

Regardless of what method is used, synthesized speech is able to convey any information that can be expressed in words to a user. The trade-off for such expressive power is time. Speech is a slow, serial method of communication. Pauses between sentences, stress on certain parts of words, and even pronunciation of certain syllables are just a few examples of the time dependent aspects of speech. As such, the rate of speech can be increased only so much before it loses information and becomes unintelligible. While expert users with visual impairments are able to listen to synthesized speech at rates up to 500 words per minute (Asakawa et al., 2003), more novice listeners are capable of parsing only 130 to 200 words per minute. Techniques such as removal of stop words, regular interval sampling, and dichotic time compression (i.e., the splitting of a single speech stream left-right channels overlapping in time) (Arons, 1994) can improve speech rates while maintaining critical information. Still, the results of these methods reduce the naturalness of the speech and likely require increased effort on the part of the listener to understand (Schmandt and Mullins, 1995).

2.1.2 Auditory Icons

Auditory icons are caricatures of real-world sounds, the sources of which are intended to represent sources of information (Gaver, 1986). The mapping from information source to its audio representation is accomplished by symbolic, metaphorical, and nomic analogies. Symbolic mappings are based on learned social conventions. The sound of wailing sirens indicating an accident has occurred is an example of a symbolic analogy. The user must be aware that sirens are used by emergency response vehicles to grasp the meaning behind the sound.

Metaphorical mappings are based on similarities between the concept being represented and the system of representation. The sound of a car starting may serve as a metaphorical representation of running a new program on a desktop computer, for example. The similarity exploited is that of the concept of *starting*, both a vehicle and programmatic process in this case. The listener must recognize this connection before the meaning of the sound becomes apparent.

With a nomic mapping, the meaning of a concept is encoded in the physics of the sound event. For instance, any sound is a nomic representation of its source. Of the three types, nomic mappings are preferred because they derive from everyday listening experience and require the least effort to learn. Gaver (Gaver, 1986) gives the example of a letter thrown into a metal mailbox as an analogy for the arrival of email. A sound that is weighty, crackles like paper, and reverberates indicates that the email is large, consists primarily of text, and is one of the only messages in the box at the time. On the other hand, a light, muffled sound indicates a small message throw into a mailbox with many other messages. The properties of new email are encoded into configurable parameters of the two objects (message and mailbox) acting as the sound source.

Auditory icons may be generated using a series of linear filters with simple impulses and step functions serving as input (Gaver, 1993). This approach is good for producing sounds of simple physical interactions (e.g., hitting, dropping, breaking) with highly configurable source parameters (e.g., object material, weight, fragility), but with limited realism. The resulting sounds are useful as nomic representations of mechanical processes (Gaver et al., 1991). Alternatively, recorded waveforms from real-world events may be passed through acoustic filters to simulate different source conditions. This method can produce realistic output for a wide range of sound events, but with limited configurability. Sounds parameterized in this manner have been used as nomic representations of widget state in graphical user interfaces (Mynatt, 1997). For instance, all menu items are represented with the same source icon, but those that are disabled are low-pass filtered to create a muffling effect. With training, users may come to recognize that disabled items sound like they are covered or blocked by another object and thus unreachable.

Auditory icons are useful for representing nominal types of information. A large number of conceptual objects can easily be labeled with uniquely identifiable sounds. Parameterized auditory icons can also capture ordinal relationships. Listeners can judge whether one instance of an icon is louder, faster, more muffled, etc. than another instance of the same icon, especially when the two are presented close in time. How much louder, faster, or muffled is not as easy to judge, however. As a result, auditory icons have limited utility in representing interval or ratio data.

2.1.3 Earcons

Earcons are structured musical motives representing concepts in an audio display through learned relationships. Differences in musical dimensions across earcons correspond with differences in one or more dimensions of the information they represent. For instance, earcons may represent items in car dashboard and mobile phone menus (Vargas and Anderson, 2003; Leplatre and Brewster, 2000). In this example, two earcons following the same melody, but played on different instruments, represent two, independent top-level menus. The shared melody conveys the shared status of the top-level menus while the different timbres distinguish their identities.

Earcon grammars define the relationships between individual earcons in terms of their musical qualities. Three methods of constructing earcon sets or grammars have been proposed (Blattner et al., 1989). First, grammars may consist entirely of representational earcons, simple sounds having only one short motive. All earcons in such a family are uniquely identifiable, but have no intended acoustic relationships. As a result, one-element families are useful for distinguishing nominal concepts through learned associations between the sounds and the information they represent. For instance, a short tuba melody may represent a link in a Web page while another distinct piano melody may represent an image. In a sense, one-element earcon families are equivalent to auditory icons with weak or arbitrary mappings to their data. A family of N representational earcons may represent exactly N concepts.

Second, earcon grammars may be created from compound earcons, two or more simple earcons presented in succession to form one, longer earcon. The compound earcon itself is analogous to a spoken phrase and its component earcons are like parts of speech. For instance, an earcon having a simple melody can convey a *copying* action. Another earcon having a dinstinct melody can represent *file* objects. A compound earcon can sequence these two simpler earcons to communicate *copying file*. Additional earcons representing modifiers, direct objects, tense, and so on can be added to the sequence to convey increasingly complex concepts (e.g., *copying a big file to a CD*). A family of compound earcons may, therefore, represent a number of concepts equal to the cartesian product of legal, simple earcon combinations. With the introduction of modifier concepts, compound earcons can effectively represent ordinal concepts in addition to nominal ones.

Third, grammars may be built from family earcons, sounds that inherit acoustic properties from other sounds in a designed hierarchy of relationships. The earcons at the top of the hierarchy represent the largest or most abstract concepts in the grammar. The immediate descendant represent slightly smaller, more concrete concepts within the groups defined by their respective parents. The same pattern continues through all levels of the hierarchy with atomic concepts defined at the bottom. The musical qualities of the earcons differ both across and down the hierarchy as a means of conveying separate branches and levels. Again, consider a series of nested menus. All top-level menus share the same rhythm and melodic contour, but differ in timbre. A submenu inherits the timbre and melodic contour of its parent, but removes some notes to reduce the complexity of the rhythm. A second level descendant may further change the rhythm or vary some other musical parameter such as octave to indicate its new depth. This pattern of inheritence and modification may continue for reasonably deep levels of nesting. Thus, family earcons are useful as a means of conveying nominal (e.g., menu or item identity), ordinal (e.g., relative depth of two items), and potentially interval (e.g., difference between two menu depths) information.

The earliest work on earcons recommended encoding information in simple musical qualities such as rhythm, pitch, octave, and loudness (Blattner et al., 1989). To the contrary, later work suggests that more complex qualities such as timbre, mode (e.g., major or minor), and consonance are better information channels for some applications (Lemmens, 2005; Hankinson and Edwards, 1999; Kline and Glinert, 1994; Brewster et al., 1993). Nevertheless, there is some consensus that, for family earcons, timbre, then rhythm, then register, then pitch, and finally loudness should vary as the hierarchy grows deeper. Chapter 4 speaks to this ordering given the properties of the human auditory sense.

2.1.4 Ambient Sound

Ambient sounds are continuous sound streams used to represent dynamic processes or persistent states (Sawhney and Murphy, 1996). Looping auditory icons or playing extended earcon melodies can aid monitoring of a real-world process over time. Changes in the dynamics of the sound can effectively indicate changes in the associated process. For example, a quickening in rhythm or increase in volume may indicate problems on a manufacturing line (Gaver et al., 1991), a flurry of requests on an email server (Farkas and Jeon, 2006), or an increase in activity in an office (Mynatt et al., 1998). Continuous sounds that lack such structured dynamics can serve as persistent reminders of static information. Recordings of unchanging real-world environments, for example, can provide retrieval cues for previously memorized information (Tan et al., 2001) and act as reminders of context (Mynatt and Edwards, 1995).

2.1.5 Audio Mixing

Mixing is the process used to combine multiple sounds into one auditory stream. A simple mixing algorithm adds scaled samples from individual sound waveforms to create the combined sound. Barring quantization error, the final waveform contains all of the information from the original, separate sound streams. Audio mixing thus increases the information bandwidth of the auditory channel by allowing two or more sounds to play simultaneously rather than sequentially.

Although mixed sounds overlap in time, listeners are capable of "hearing out" the original streams in many circumstances. Concurrently presented musical tracks, for instance, can be heard as individual songs aiding their individual selection during browsing (Fernström and McNamara, 1998). Likewise, multiple speech streams of news articles can be heard and selected when presented simultaneously (Schmandt and Mullins, 1995). Listeners can also distinguish simultaneously presented earcons as well as determine their musical qualities (McGookin and Brewster, 2004a). In general, a person can discriminate among five concurrent non-speech sounds with moderate accuracy, and three sounds with a much lower error rate (Lorho et al., 2001). The ability of a listener to segregate and group both serial and simultaneous sounds is discussed more thoroughly in Chapter 4.

2.1.6 Stereophony and Spatialization

Stereophony is a technique for producing sound that originates around the listener using two or more speakers. Segements of an auditory scene, called *channels*, are output on one or more speakers at varying levels of intensity. The result, from the listener's perspective, is that some sound sources are located in different positions in space. For instance, assume a person is watching a movie with six channel audio on his or her home entertainment center having five speakers (front-left, front-right, front-center, rear-left, rear-right) and one subwoofer. If the voices of a male and female character are encoded at highest intensity in the front-left and front-right channels, they will sound as if they are speaking to one another in front of the viewer. Furthermore, they will be slightly spatially offset from one another, most likely in correspondence with their locations on the screen. At the same time, if the ambiance of the current scene is equally encoded in all channels, the environment will sound as if it is surrounding the listener on all sides.

Spatialization is a digital signal processing (DSP) technique that makes sound appear as if it originates from a source in three-dimensional space around a listener wearing ordinary headphones. Unlike stereophony, the number of potential sound source locations is not limited by the number of physical speakers. Software DSP libraries are capable of producing spatial sound on modern personal computers using predefined head related transfer functions (HRTFs) (Burgess, 1992). The HRTF determines the frequencies reaching the eardrum for a sound source located at some azimuth, elevation, and distance away and producing a given frequency spectrum (see Chapter 4). In this technique, a set of HRTFs is generated by measuring the frequency response at the eardrums of a model human head for near-impulse sounds. The sounds are generated by a real source located in the far-field at a variety of elevations and azimuths.

At render time, HRTFs in the set are selected based on their proximity to a virtual sound source. The filters nearest the desired point are interpolated to produce two approximate HRTFs, one for the left ear and the other for the right. The resulting filter is then applied to sounds during playback to simulate their origin at the azimuth and elevation of the virtual source. The distance of the sound source, on the other hand, is usually simulated by simply modifying the volume of the sound.

The HRTF approach makes sound spatialization a commodity on modern computers. Nevertheless, it has some limitations which must be considered:

- The synthetic head used during sample is a generalized model. The HRTFs measured with this head may not produce the desired perception in all listeners, especially those with heads that vary greatly in size and structure from the model (e.g., children).

- Distance is not included in the calculation of most HRTFs. Virtual locations in the near-field (e.g., within 1 meter of the head) cannot be accurately simulated.
- Generating true impulse or step function sounds is difficult. The approximation introduces error into the recorded responses.
- Only a few levels of elevation are sampled during HRTF recording. Creating localizable sounds above and below the level of the ears is difficult for this and other perceptual reasons (again, see Chapter 4).

Spatialization has found two primary uses in software systems. First, positioning sounds in space helps disambiguate concurrent sounds. Spatialization is one of the best ways to aid detection, localization, and intelligibility of two or more simultaneous speech streams (Simpson et al., 2006; Best et al., 2005; Brungart and Simpson, 2003). Similar benefits are obtained when spatially separating concurrent earcons (McGookin and Brewster, 2004b). Second, the spatial positioning of a sound may represent information along another dimension. For instance, the spatial location of an audio news cast circling around the head can indicate the current temporal position in the audio stream (Kobayashi and Schmandt, 1997).

2.1.7 Other Techniques of Note

- Spearcons (Walker et al., 2006) are synthesized speech utterances increased in rate until they are no longer recognizable as such. The initial portion of the resulting sound is acoustically related to the original speech regardless of the magnitude of the speedup. Evidence suggests that presenting spearcons as hints before their fully spoken counterparts for menu names can improve user speed and accuracy in selection tasks. With respect to the techniques described above, spearcons are something of a cross between time compression for synthesized speech and auditory icons.

- Simulated reverberation, the echoing of sound waves off of physical surfaces, can improve distance perception in auditory displays. On the other hand, it hinders the localization of sound sources in azimuth and increases masking among concurrent streams (Shinn-Cunningham, 2000). Producing this effect is computationally prohibitive for complex geometries at present, further limiting its usefulness.

- Sheppard-Risset scales create the auditory illusion of a sound appearing to rise or fall in pitch forever. The scales are created by cycling a finite set of sinusoidal tones in frequency and passing them through a bandlimited, raised-consine filter. The resulting sound is useful for conveying direction information using pitch orientation or rate information using shift speed (Beaudouin-Lafon and Conversy, 1996).

2.1.8 Summary of Techniques

Table 2.1 summarizes the strengths and weaknesses of the techniques described in the preceeding sections.

2.2 Interaction Metaphors

Except for synthesized speech, the techniques above are meaningless unless the user can learn to associate sound with the information it represents. An auditory display requires structure to facilitate user learning, preferably one based on past experience and the task at hand (Demarey and Plénacoste, 2001). As such, auditory displays are often designed around metaphors that support analogical transfer between previous knowledge and interaction with the display.

Technique	*Strengths*	*Weaknesses*
Synthesized speech	Conveying linguistic information with exacting detail, no learning required	Slow rate of communication, potentially unnatural sounding, lack of context leads to imperfect pronunciation
Auditory icons	Conveying nominal and ordinal information, learning based on everyday listening, ease of playing pre-recorded sounds	Limited number of useful nomic mappings, difficulty of generating realistic sounds dynamically at runtime
Earcons	Conveying nominal, ordinal, and interval information, ease of producing dynamic music at runtime	Learning based on musical aptitude
Ambient sound	Conveying persistent state or dynamic processes	Potentially annoying, listener adaptation
Audio mixing	Conveying independent information streams simultaneously, enabling rapid user switching among streams	Potential for masking across streams, low upper limit on number of concurrent streams
Spatialization	Conveying spatial information, increasing presence in virtual environments, unmasking of concurrent streams	Difficulty simulating elevation and near-field distance, difficulty distinguishing front-back

Table 2.1: Auditory rendering techniques with their associated strengths and weaknesses.

Only a handful of audio metaphors are in use today. The following sections describe these metaphors and note some representative systems. More example implementations of each are introduced in later sections in this chapter.

2.2.1 Narrator

The narrator metaphor is conceptually simple. As the user interacts with an application, a synthesized speaker, the narrator, reports the state of the system. The narrator may change voices to represent different types of information, but only the one voice speaks at a time (i.e., a single, serial stream of speech). Non-speech sounds may be inserted into the narration to represent non-verbal information (e.g., embedded images) or to shorten or reinforce the presentation of text content.

Conventional screen readers (Thatcher, 1994) operate under the narrator metaphor. A user with a visual impairment interacts with an application using its native user interface (e.g., a GUI). The screen reader monitors this interaction and reports important events and changes in the state of the visual user interface. The user has some control over how the narrator, the screen reader, operates, but most of his or her interaction is directed at the application. The auditory display is largely passive.

2.2.2 Sonification

The sonification metaphor is used to present data that cannot be linguistically summarized by a computer. Dimensions of the source information are encoded in the dimensions of non-speech sounds by the auditory display. The sounds are played, either in series or in parallel, for a listener. It is then the responsibility of the listener to deduce the encoded information from the sounds heard. The display designer must choose appropriate mappings between data and sound dimensions to aid the listener in this task. For instance, ordinal rankings are more clearly conveyed by earcon register, an ordinal property, than by timbre, a nominal property.

Numerous projects have explored the sonification metaphor as an auditory alternative to visual presentation. The vOICe (Meijer, 1992) sonifies two-dimensionsal digital images by continuously sweeping a one pixel wide cursor horizontally across image. The system produces simple sinusoidal sounds with pitch mapped to the vertical location of a pixel under the cursor and volume mapped to the intensity of its corresponding pixel. The GUESS system (Kamel and Roth, 2001) supports exploration of geometric shapes using stylus input and audio output. In one configuration, a moving, spatialized tone traces out the contour of two-dimensional shapes in the elevation plane. Short beeps are sounded at line segment intersections, pitch doubly encodes elevation, and intensity indicates how close the user is holding the stylus to a nearby shape. Mereu reports on a similar system for exploring three-dimensional images with either tone pan, pitch, and volume or earcon instrument, pitch oscillation, and instrument family encoding spatial x, y, and z coordination (Mereu and Kazman, 1997). Ghez, et al. report on a realtime sonification system for communicating arm position and joint motion to people lacking the proprioceptive sense (Ghez et al., 2000). Rhythm, tempo, and pitch contour provide strong cues for coordinating movements during patient rehabilitation. Sonification of time-varying data is also possible and has been applied to climate data (Flowers et al., 2001), Web server traffic (Farkas and Jeon, 2006; Barra et al., 2001), and program debugging sessions (Finlayson and Mellish, 2005; Vickers and Alty, 1998).

2.2.3 Direct Manipulation

Direct manipulation is the operative metaphor on all modern graphical desktop environments. Software concepts are represented by persistent visual icons that the user manipulates to bring about a change in the state of the system. For instance, a desktop icon may be clicked to open the file it represents or dropped on a trash bin icon to delete the file. The benefits of direct manipulation include reliance on recognition rather than recall, user feelings of increased control over the interaction, and easy reversibility of incremental actions (Shneiderman, 1992).

Auditory direct manipulation is similar in design except that it uses continuous sound sources instead of images to represent objects. A person uses an input device, often a pointer, to select a sound source for manipulation. User actions such as dragging a source to a new location, dropping one source on another, or obstructing a source serve as input for the system. The goal of using this design is to reap the same benefits offered by its visual counterpart.

The Soundtrack system (Edwards, 1988) attempts to convert graphical direction manipulation interfaces into auditory equivalent. Soundtrack plays distinct tones indicating the identity of graphical widgets in a word processor as the mouse cursor passes over them. A visually impaired user explores the contents of the screen by moving the mouse, listening to the tones, and clicking on desired sound objects. The interface produced by this straightforward conversion from graphics to sound proved difficult to master for all but musically trained users, and was later even disparaged by its creator (Edwards, 1992).

More recent work revisits auditory direct manipulation for common desktop applications using design patterns as a guide (Frauenberger et al., 2005; Frauenberger et al., 2004). In this approach, common direct manipulation actions are mapped first to domain independent interactions and then to appropriate auditory forms. Frauenberger describes the design of an auditory equivalent to a graphical file system browser following this method. He identifies the key components of the GUI file browser (i.e., toolbar, menubar, folder tree, file list, context menus, popup messages), maps them to design patterns (i.e., command areas, container navigation, contextual menus, messages), and finally to auditory widgets in a virtual sound space. For example, the graphical toolbar becomes a bank of representational earcons while the menu turns into a grammar of family earcons with speech labels. As in Edwards' approach, there remains a tight correspondence between graphical widgets and auditory sound sources.

Auditory direct manipulation has been applied in other contexts as well. The auditory Towers of Hanoi game (Winberg and Hellström, 2000) represents ring identity using timbre and pitch, ring position using stereo panning, and stacking order by sound duration. The user configures the game to present sounds for all rings in series, in parallel, or with slight overlaps, and uses the mouse to select, drag, and drop the rings among the posts. Winberg applies similar concepts to supporting auditory drag and drop in a display with overview and zoom modes (Winberg and Hellström, 2003). The hierarchical browser developed by Savidis, et al. uses spatialization to position sound objects in a ring around the user's head (Savidis et al., 1996). The user points, gestures, speaks, and orients his or her head to select, move, insert, delete, and focus on particular objects in the ring. Only four objects are presented simultaneously at the front of the ring in an exploded view while the rest are crowded silently behind the user. The user turns the ring to bring silent sources to the front and into focus.

2.2.4 Conversation

The conversational metaphor draws on the familiar conventions of verbal communication as a basis for human-computer interaction. Typically, an audio display following this metaphor has a single synthesized voice acting as a conversation partner. The speaker questions the user, responds to user queries, and gives feedback about user commands while honoring conversational behaviors like turn taking, timing, feedback, repair, and grounding (Cassell et al., 2000; Brennan and Hulteen, 1995). Unlike in the narrator metaphor, the speaker acts as an active intermediary between the user and the application, not a passive reporter.

The Danish Dialogue System is an example of a conversational auditory display (Bernsen et al., 1998). The software provides a telephone voice response service for the ordering of airline tickets using segments of pre-recorded speech as output and voice recognition as input. Since the process of booking a flight is well-defined, interaction is system-directed. The computer asks the user a predefined question, the user responds by speaking, and the system decides whether to continue with another question, echo important information as a passive confirmation, or ask for a clarification. The system is designed to respect maxims of conversation such as quantity, quality, relevance, manner, and repair (Grice, 1975). Given the tendency of users to anthropomorphize computers, such features can make the system more natural and intuitive to use (Bickmore, 2004).

The conversational metaphor is useful beyond telephony applications too. The Rea virtual real estate agent embodies the speaker in a graphical avatar (Cassell et al., 2000). Avatar body language supplements speech output by acting as a secondary information channel, and lip synchronization improves speech intelligibility. Some researchers even suggest that conversation and direct manipulation are one in the same, and that conversation can serve as a useful model for designing audio, text, and graphical interfaces (Brennan, 1990).

2.2.5 Environments

The environment metaphor uses the sound of real-world places to represent virtual locations in software (e.g., a task in a business application, an area in a first-person shooter game). Continuous ambient sounds create an atmosphere that serves to identify a location, distinguish it from others, and encode details about the virtual place. Navigation through the application is accomplished by navigating from one auditory environment to the next.

Use of the auditory environment metaphor is abundant in modern video games. For example, Halo (O'Donnell, 2002) is a first-person shooter video game featuring realistic sound environments that reflect the properties of a visual 3D virtual world. The humming of engines and buzzing of electronics suggest the player is currently on an alien spacecraft. Muffled footsteps direct attention to the presence of an enemy in an adjacent room. When the door to the room opens, the late echos of chatter hint at its large size. These sounds supplement the visuals of the game, helping to orient the player and increase realism.

Auditory environments have been used to aid navigation in business applications as well. The Audio Rooms thought-experiment (Mynatt and Edwards, 1995) proposes organizing applications and content on a personal computer by placing it in named rooms. The user navigates from room to room by walking through doors, and learns what is in each room by listening to speech and sounds representing its contents. The Audio Hallway system (Schmandt, 1998) organizes continuous spoken news reports in rooms connected to a long, straight hallway. The use of spatialization and concurrent streams gives the user the experience of walking down a hallway and hearing people talking in each room passed. Sawhney (Sawhney and Murphy, 1996) and Lumbreras (Lumbreras et al., 1996; Lumbreras and Rossi, 1995) propose similar designs for auditory navigation of hyperlinked content. In both systems, sounds are used to identify the user's current browsing context as well as to indicate contexts reachable by following links.

2.3 Audio Aware Systems

A program with inherent support for auditory display is termed an *audio aware application*. Such an application uses sound hardware, mixing libraries, and system services available on most modern personal computing platforms to provide an auditory display. The information to be output and the possible ways of representing it in sound are coded directly into the application at design-time. When driven by the application model, this integration can result in an auditory display that is both usable and aurally pleasing.

Today, most desktop business software is either not audio aware or makes limited use of available audio capabilities. For instance, interaction with large office suites such as Microsoft Office or OpenOffice is entirely dependent on a graphical display. These applications do allow the embedding of audio clips into documents and the playing of sound effects during slide shows, but use of the programs themselves requires sight. Some programs find slightly more interesting uses for audio, such as making a sound when an email or chat message is received. Such uses are still limited to supplementing a visual display in minor ways. The fact is, the cost of providing non-trivial audio is difficult to justify for most projects when the model user is a sighted person looking at a monitor.

Despite this state of affairs on the desktop, there are at least three commercial domains in which audio aware systems have thrived: telephone voice response systems, gaming, and mobile computing. In addition, the research literature on auditory displays documents systems that have been implemented and tested, but almost never commercialized: audio desktops and desktop applications; sonification of maps, graphs, and diagrams; and audio aids for peripheral awareness. The following sections review representative systems in each of these areas.

2.3.1 Telephone Voice Response Systems

A voice response system (VRS) uses concatenative or recorded speech to create an auditory display heard over a standard telephone. A basic VRS presents a caller with a numbered set of options and asks the caller to select one. The selected option leads either to another menu or to a potential answer to a user's question. Exactly how the user navigates the menus and selects options varies from system to system. For instance, Resnick describes at least ten distinct designs for a VRS relying on a telephone keypad as the sole input device (Resnick and Virzi, 1995). Slightly more advanced systems allow the user to respond simply by echoing a spoken option or speaking a phrase from a limited vocabulary. The Danish Dialog System (Bernsen et al., 1998), for example, lets a caller state a desired flight departure time by speaking a time in a natural format.

The most advanced systems strive to make interaction with the virtual assistant more like conversation with a real receptionist. Both Chatter (Ly and Schmandt, 1994) and SpeechActs (Martin et al., 1996; Yankelovich et al., 1995; Yankelovich, 1994) ask more open ended questions that invite the user to control the flow and direction of the interaction. The systems are designed to handle errors in recognizing user responses, new application domains and associated vocabulary, and multitasking with user sidetracks and continuations. The design of these systems is discussed in more detail in Chapter 3.

Skantze and Dahlbäck (Skantze and Dahlbäck, 2003) apply earcons and audio icons to a telephone voice response system for building maintenance. The non-speech sounds play in the background while the user queries the system about a particular room. The sounds are representative of the status of a room. For instance, the sound for a water valve in a water main room may play loudly if the valve is fully opened or sound muffled if it is partially shut. A study of this system revealed that users prefer the use of audio icons over earcon and speech-only designs during building walkthroughs.

Though not technically voice response systems, two other telephone related interfaces are worth mentioning here. Steel describes a system for augmenting conversations between callers and human receptionists with audio icons and earcons (Steel et al., 2002). The sounds play when the receptionist is busy working on fulfilling a caller's request so as to indicate that the call is not on hold or disconnected and to suggest that the caller should not interrupt. A study showed that the use of non-speech sounds in this context decreases the number of inappropriate interruptions at the cost of caller satisfaction.

The TalkBack system (Lakshmipathy et al., 2003) turns the process of reviewing messages on a conventional answering machine into a dialog between the recorded voice of the caller and the recipient. The recipient interrupts the playback of a recorded message to inject his or her own spoken responses. The entire audio clip (i.e., original message plus inserted response) is then returned to the original caller who may repeat the process. Though the conversation is asynchronous, the majority of users who tested the system felt they were in a real conversation and created higher quality responses than if they had simply returned the call. Time-compression of spoken messages is also supported, but did not help most users.

2.3.2 Games

Modern commercial video games, especially those featuring rich, visual 3D environments, incorporate sound to enhance the realism of the game and set the mood of play. As mentioned earlier, Halo is a quintessential example of a game that incorporates "sounds to make it real" and "music to make you feel" (O'Donnell, 2002). Spatially located sounds, audio ambience, environmental effects, and spoken dialog immerse users in Halo's environments and alert them to important in-game events. A dynamic sound track adjusts to the activity of the player's current situation to create an emotional connection to the virtual world. Many first-person shooters and other adventure games created both before and after Halo make use of one or all of these techniques to some extent.

Commercial game designers have used audio primarily as a supplement to visual gameplay. There are examples, however, of games where audio is the primary medium and visual information is secondary. For example, the Band Brothers game for the hand-held Nintendo DS system encourages the cooperative production of music among multiple players. Each player chooses an instrument and tries to play his or her part by pressing one or more buttons in time with his or her partners. The Electroplankton game for the same platform provides a more open-ended experience where players use the system's touch screen to manipulate the production of dynamic music and its accompanying visual effects. Other popular games such as Dance Dance Revolution and Guitar Hero require the player to perform actions such as stamping on a footpad or strumming on a guitar shaped controller in time with music tracks.

The research literature describes numerous audio games that are playable entirely without vision. Games featuring fully navigable three-dimensional sound environments are useful for promoting active learning through exploration (Lumbreras and Sánchez, 1999) and teaching orientation and mobility skills to students with visual impairments in a safe, calm environment (Inman et al., 2000). Studies indicate that audio-only implementations of classic games such as Tic-Tac-Toe and Mastermind may serve as potential exercises for mental therapy and memory skill development (Targett and Fernström, 2003). Audio implementations of direct manipulation games (e.g., drag-and-drop Towers of Hanoi), first-person action games (e.g., avoid enemy bullets whizzing through 3D soundspace), and adventure games (e.g., puzzle solving in virtual acoustic worlds) show that not only is it possible to set games entirely in audio, but that such games may also prove enjoyable to both blind and sighted players (Winberg and Hellström, 2000; Friberg and Gärdenfors, 2004; Roeber and Masuch, 2005).

2.3.3 Mobile Devices

Portable computing devices such as cell phones, PDAs, and music players prize small form factor, long battery life, and user mobility as good design. All three of these aspects may be enhanced by auditory displays that enable eyes-free operation and smaller or non-existant screen space. At present, few commercial mobile devices enable fully eyes-free use via auditory displays. Specialized assistive technologies such as the Freedom Scientific Braille 'n Speak notetaker, Owasys 22C screenless phone, and VisuAide Trekker navigation guide use synthesized speech to communicate information to people with visual impairments. Some automotive dashboard navigation systems such as the Pioneer AVIC-Z1 are fully speech enabled for both audio input and output, though most on the market today use speech output as a supplement to a visual user interface.

Some of the earliest research literature on mobile auditory displays concerned navigation aids for people with visual impairments. The Sonicguide (Kay, 1974) takes the form of a pair of spectacles equipped with an ultrasonic transmitter on the bridge and two ultrasonic receivers on each side. Echoes detected at each receiver are presented to either the left or right ear. The difference in time of arrival of the signal enables the listener to determine the direction of an obstacle. The GuideCane (Borenstein and Ulrich, 1997) relieves the user from the burden of manually sweeping a set of sensors from side to side by placing the detectors in a wheeled cane. The motorized cane gives speech feedback about the current position of the walker using GPS and guides the user around obstacles by turning its wheels automatically. The wearable navigation system devised by Ross and Blasch (Ross and Blasch, 2000) combines a spatially positioned sonic carrot, spoken directional headings, and tactile feedback from a backpack to keep a walker on a planned route. The Drishti system (Helal et al., 2001) relies both on speech input and output for communicating travel routes and current travel conditions.

Recent research literature is growing rich with additional examples of audio displays for mobile computing. Earcon grammars have been used to supplement visual menus in car control interfaces (Vargas and Anderson, 2003), PDAs (Brewster et al., 2003), and mobile phones (Leplatre and Brewster, 2000). Studies of some of these systems suggest that family earcons may reduce the number of user input gestures and menu selection errors. The Dolphin system (McGookin and Brewster, 2002) uses concurrent, spatialized earcons to provide context cues beyond the visual area on small PDA screens. Though a user study of browsing theme park maps in Dolphin failed to show significant improvement in speed and quality of planned routes between rides, later work by McGookin shows how the simultaneous earcons might be improved to better provide context (McGookin and Brewster, 2004a; McGookin and Brewster, 2004b). The Personal Universal Controller (PUC) (Nichols et al., 2002) generates remote control speech interfaces for appliances such as stereos, televisions, and DVD players. The PUC system uses the *model-view-controller* (MVC) paradigm to generate both audio and graphical interfaces (view plus controller) from a single description of device capabilities (model).

Nomadic Radio (Sawhney and Schmandt, 2000; Sawhney and Schmandt, 1998) integrates the gamut of audio rendering techniques into a portable auditory display for remote information services such as email, voice mail, news broadcasts, and personal calendars. Streams are positioned in space according to their time of arrival in a circle around the listener. The priority of an information stream dictates its presentation as a way of avoiding unnecessary interruptions of user tasks. The faint sound of running water plays in the background to indicate message download progress. Auditory icons announce new events, confirm user input, indicate mode transitions, and declare exceptions. Speech synthesis gives both summaries of information as well as full details of a stream.

2.3.4 Desktop Applications

Though few commercial business applications take advantage of auditory displays, research prototypes demonstrate how audio may be incorporated into the desktop environment. Cohen and Ludwig describe one of the earliest audio equivalents for a desktop windowing system (Cohen and Ludwig, 1991). Instead of representing applications using visual windows and widgets, the system uses spatially positioned streams of speech to report application content. What the authors term *filtear*s—auditory filters causing muffling, reverberation, pitch shifting, etc.—act like auditory typography to emphasize certain streams and de-emphasize others. Headtracking and data glove input allow users to bring certain streams into focus and manipulate stream positions as they would windows with a mouse pointer. Using spatial audio in this manner to distinguish multiple information streams increases user memory and comprehension of content over a non-spatialized equivalent (Baldis, 2001).

Other researchers have concentrated on creating audio counterparts to visual widgets. Mitsopoulos and Edwards describe a methodology for designing auditory widgets at the conceptual (information or task to be represented), structural (the place of a widget in an auditory scene), and implementation (the physical dimensions of a sound) levels (Mitsopoulos and Edwards, 1998). The work includes a description of a possible design for an auditory list box widget starting with an analysis of the salient features of a visual list box and their mapping to multiple auditory streams. Frauenberger extends this approach using *interaction design patterns*, descriptions of common solutions to problems in user interface design (Frauenberger et al., 2005; Frauenberger et al., 2004). Whereas most collections of design patterns, called *pattern languages*, suggest implementations for visual interfaces, his pattern language describes solutions for supporting common user tasks in a medium independent manner. In one example, Frauenberger abstracts the interactions with a graphical file explorer application to the set of tasks they support: menu bars become the command area pattern, popup windows become the message pattern, trees and lists become container navigation patterns, and so on. He then implements an audio version of the same application by creating audio widgets enabling the tasks described by the patterns. The resulting audio interface is designed for the tasks at hand, not their original visual representations.

Audio-based Web browsers for people with visual impairments have become a popular research topic. The Brookestalk system (Zajicek et al., 1998) reads Web pages in full and summarizes them using extracted keywords and abridged text using synthesized speech. Goose and Möller describe a system that uses spatialized speech and sound to represent the relative locations of elements on a Web page. For instance, if the narrator is approaching a link while reading a Web page, a sound representing that link will grow louder and move toward the center of the virtual sound space until it is passed and fades into the distance. The AB-Web browser also maps a Web page into a virtual sound space, but, instead of reading through them linearly, allows a user to explore the layout and content of the page using an absolute pointing device (Roth et al., 2000; Roth et al., 1998). The related Websound tool provides a workspace for developers to associate sounds and other feedback with events and objects on a Web page (Petrucci et al., 2000). Home Page Reader (Asakawa and Itoh, 1998) is a commercial product from IBM that supplements the visual rendering of a Web page by the Internet Explorer Web browser with a complete auditory display using synthesized speech and earcons. Similarly, the FireVox extension (`http://www.firevox.clcworld.net/`) for the Firefox Web browser enables browsing of Web pages using synthesized speech without changes to the underlying browser.

Not all browser-related projects implement full auditory user experiences. Some only supplement the standard browsing experience with audio. Harper and Patel describe an extension for the Firefox Web browser that builds text summaries of Web pages by extracting topic sentences (Harper and Patel, 2005). Users with visual impairments may listen to the summaries using a standard screen reader to gain a gist of a page. The Audio Enriched Links system (Parente, 2004) is a plugin to the JAWS screen reader that provides both spoken and earcon previews of linked pages. The results from a simple study show that the use of such previews reduces the number of incorrect links followed by users with visual impairments during a targeted browsing task. Midgley and Vickers discuss another Firefox extension that uses hierarchical earcons to give feedback about a user mouse gesture, sequences of mouse movements representing an input command (Midgley and Vickers, 2006). The start and completion of legal gestures are confirmed with earcons distinguished in timbre, register, and rhythm while illegal gestures are represented with a dissonant earcon. An initial study shows a user preference for auditory gestures over their silent counterparts.

Besides Web browsers, few other audio aware desktop applications have been researched. Hudson details a system of earcons and auditory icons for summarizing the length, topic, sender, relation to important threads, presence of attachments, etc. of new email messages (Hudson and Smith, 1996). The software is not a replacement for an email client, but rather a way for a user to gain a peripheral glimpse of incoming mail without interrupting his or her current task. Stockman, et al. (Stockman et al., 2005) describe a prototype for playing audio overviews to Excel spreadsheets. Pitch is used to communicate cell value, while the system supports numerous modes of presenting ranges of data for comparison: sequential, simultaneous, interleaved, arithmetic, bounded. The user controls parameters such as sound timbre, duration, and frequency range. A basic evaluation of the system shows that users recognize spreadsheet sonification as useful for certain tasks (e.g., getting an overview of data, comparing data ranges, locating outliers), not for getting exact details on a cell by cell basis.

2.3.5 Maps

Auditory displays for geographic information can be classified by two categories. The first set includes displays that provide in situ aids for navigation. Examples of such systems were described in the mobile devices section above. The second set is composed of systems intended to give users with visual impairments the ability to explore and learn about maps. Jacobson proposed the first of such systems using auditory icons and speech synthesis for output and a touch pad for input (Jacobson, 1998). A user explores an auditory map in his system by tracing his or her finger on the touch pad and listening to the representative sounds and spoken names of locations. The Blind Audio Tactile Mapping System (BATS) (Parente and Bishop, 2003) extended this idea by using spatialized auditory icons to callout nearby areas of interest, adding queries for additional information at the current pointer position, supporting commodity haptic feedback devices, and enabling the construction of new maps from GIS data. The iSonic project (Zhao, 2006) takes auditory displays for maps in a different direction by sonifying choropleth maps. In iSonic, tonal sounds and speech are used to support exploration of abstract data over geographic regions rather than geography itself.

2.3.6 Graphs

Auditory graphs encode numeric information in properties of sound. The work by Brown, et al. (Brown et al., 2003; Brown and Brewster, 2003) describes a system and makes recommendations for representing line graphs in audio. Her design suggests using instrument timbre and panning to distinguish data series presented concurrently and scaled pitch to represent magnitude while giving the user control over serial versus parallel presentation, the loudness of each series, and the speed of the sonification. Peres and Lane (Peres and Lane, 2003) describe a similar system for sonifying box and whisker plots representing the minimum, 25th percentile, median, 75th percentile, and maximum of statistical distributions. The software uses sound pitch, stereo panning, and pitch+panning to represent the absolute magnitudes of serially presented box plots. Two user studies of the system suggest that pitches of long duration (9 seconds) alone are best for communicating distribution skew, central tendency, and spread, but that users prefer the redundant presentation. The aforementioned system for sonifying time-varying weather data (Flowers et al., 2001) is a third example of an auditory display for numeric data. Data series for temperature, rainfall, and snowfall are separated by timbre, rhythm, and melody while pitch communicates high-low temperatures and categorizes rain and snowfall levels.

2.3.7 Diagrams

Akin to auditory maps and graphs is the concept of auditory diagrams. Whereas maps and graphs depict geography and numerical information respectively, a diagram portrays more abstract information. For instance, visual class diagrams in the Unified Modeling Language (UML) use topology to convey relationships between software classes, geometry to convey component type, and text to describe class details. The Audiograf system (Kennel, 1996) provides an auditory representation of such node-link diagrams. The software uses speech to describe diagram nodes, their connections, and their attributes as the user drags his or her finger around a touch pad. Metadata describing the relationships among components must be present in the diagram for Audiograf to work. The similarly named Audiograph system (Alty and Rigas, 1998) focuses on representing diagrams composed of geometric shapes placed in arbitrary spatial arrangements and having no additional metadata describing their relationships. In Audiograph, sound timbre (x,y) and pitch (magnitude) are used to represent the position of the user's cursor and graphical objects. The attributes of an object such as its size and shape are communicated by tracing the outlines of shapes in music using the timbre and pitch mappings. An overview of the entire diagram is similarly produced by musically tracing shapes in one of three optional orders. Even the commands for the software such as undo and redo are represented by a representational earcon grammar. The previously described GUESS system (Kamel and Roth, 2001) also communicates diagram information using non-speech sounds, but in a 3D sound space to reinforce spatial concepts.

2.3.8 Peripheral Awareness

Gaver aptly describes sounds as existing in time and over space in contrast with visual objects that exist in space and over time (Gaver, 1997). Whereas seeing requires that a person face a particular direction and open his or her eyes to see a visual signal, hearing has no such requirements. A person may listen to an audible signal regardless of its point of origin without turning to face it or changing his or her visual gaze. For this reason, audio is an ideal medium for constructing peripheral information displays.

The Audio Aura system (Mynatt et al., 1998) is one example of an auditory display for peripheral awareness. Audio Aura uses both looping and ephemeral audio icons, musical tracks, and speech to communicate information about activities and events in an office environment. All of these cues are designed to be non-invasive, that is to avoid grabbing user attention with sudden changes and to fade into the background when not of interest to the listener. For example, in one audio icon set, the intensity of continuously rolling ocean waves indicates the current level of activity in the office while occasional cries of seagulls convey the number of new emails received.

Auditory displays for Internet server traffic provide additional examples of systems promoting peripheral awareness. The iSIC system (Farkas and Jeon, 2006) uses continuously generated music to represent email traffic on SMTP, spam filter, and mail spooler systems. The system is in production use by network administrators at Sheriden College. The work by Barra, et al. (Barra et al., 2001) shows that system administrators may monitor sonified Web server traffic for extended periods of time if they are allowed to choose their own music. Listener fatigue is reduced when music tracks of interest are played and manipulated instead of computer generated music.

2.4 Audio Unaware Systems

An *audio unaware application* is one that does not implement enough support for auditory output such that it may be used effectively by listening to it without looking at it. Nearly all modern desktop applications fit this definition, as their user interfaces are presented almost entirely on graphical displays. On their own, these applications are of little interest to this research. What is important, however, is how other programs are able to retrofit audio unaware applications with auditory displays. The following sections review *auditory agents*, programs that provide auditory displays for audio unaware applications.

2.4.1 Model-Based User Interfaces

Model-based approaches to producing user interfaces rely on a set of high-level specifications of application function and data, and the ability access both at runtime. These methods can be categorized according to various properties (Omojokun and Dewan, 2007). A major classification of interest in this research is that of manual user interface creation versus automatic generation. In the manual approach, the software developer handcrafts one or more user interfaces based on a single application model. For example, a developer can create user interfaces in a variety of formats for REST services (Fielding, 2000). As a more concrete example, consider a stock ticker service accessible via a URL. One developer might produce a graphical user interface plotting values fetched from the service over time. Another developer might produce an auditory display that speaks critical changes in stock value. Both views are written to work with the same, view-unaware stock ticker model.

Automated generation of user interfaces for application models is also possible, and can be split into two subcategories. The first classification includes systems that produce user interfaces based on heuristics. For example, SUPPLE generates graphical user interfaces for applications based on device type, form factor, input methods, and personal needs (Gajos and Weld, 2004), and can adjust the resulting UI on-the-fly as parameters change. SUPPLE and similar constraint-based systems require little to no user interface development effort, but by sacrificing direct control over the results. Though not explored in the research literature, such an approach could be used to produce auditory displays in much the same manner as visual ones, albeit under different constraints (e.g., number of concurrent sound streams instead of screen real estate).

The second category of generators is defined by systems that read high-level, user-provided specifications of application models to produce user interfaces in various forms. On the desktop, Scope (Beshers and Feiner, 1989), UIDE (Sukaviriya et al., 1993; Foley et al., 1989), and Dost (Dewan and Solomon, 1990) are early examples of generators using information about model data types, property names, supported actions, preconditions, and so forth to generate graphical and text-based user interfaces. These systems were intended to support application developers in producing effective UIs with minimal coding effort. They were not intended to support third-party generation of user interfaces for existing applications.

More recent research on specification-based generators has targeted networked appliances (e.g., televisions, DVD players, stereos). Various systems (Omojokun and Dewan, 2007) seek to produce software interfaces for simple devices based on the commands they support and state information they provide via their remote control protocols. For example, the PUC project *Nichols02* mentioned in Section 2.3.3 generates visual and speech interfaces from a high-level description of device functions, hierarchical groups of functions, and dependency information. The UI on the Fly project (Reitter et al., 2004), on the other hand, melds speech and graphics to produce a multimodal interface. These systems and others were designed to generate new user interfaces for existing appliances already having their own physical or software interfaces.

Generation or manual development of model-based displays for audio unaware systems is attractive when an application provides public access to its data model and business logic. If an application model is accessible via some well-defined API or protocol, a third-party agent can produce, manually or automatically, an auditory display for it. The quality of that display is determined both by the level of model access available to the agent as well as the ability of the agent to produce an effective display given the information at hand. Given full access to the model of an application, a well-designed agent could produce a display rivaling that of an auditory aware system.

2.4.2 Screen Reading

Unfortunately, the majority of existing desktop GUI applications do not provide direct, consistent, programmatic access to the state and function of their models. As an example, consider that, on the day of this writing, the CNET Top Ten Windows Downloads tracker (`http://www.download.com/most-popular-windows-software/`) lists only one application that offers a model-based API. Applications that do provide a public API often limit what information an external process can retrieve or manipulate for reasons such as security, data consistency, and protection of intellectual property. For example, Microsoft Office has an API for programmatically manipulating its primary document types: spreadsheet, document, slideshow, database. Yet this API does not cover all of the tasks a user can accomplish when working with these document formats in the Microsoft Office GUI.

Screen reading is an approach to enabling auditory display for GUI programs with closed models. Instead of relying on access to an application model, a *screen reader* indirectly inspects and controls an application via its on-screen widgets.

Console Screen Readers

In 1986, IBM released its first of many screen reader products, the IBM Screen Reader for DOS, a program capable of creating a synthesized speech display for applications running in the 80 by 24 character text-mode console. The software operated by peeking into the video buffer, extracting characters and cursor position, and speaking them to the user. The screen reader defined keyboard commands for querying information on the console immediately (e.g., read the next character, word, line), after a significant change (e.g., read new lines as they scroll into view), or on changes in specific screen regions (e.g., read status text when it changes in the ten bottom-right characters).

Console screen readers are still in use today, particularly in Unix-like environments where use of the text-only console is still practical and powerful. Open source projects such as Speakup (`http://www.linux-speakup.org`) and YASR (`http://yasr.sourceforge.net`) are available free of charge and operate on Linux virtual terminals. These new projects mimic most, if not all, of the features of the original IBM product. However, as most modern applications now run in GUI environments, simply relying on a text console for all work is not a solution for people with visual impairments, especially in the workplace.

Off-Screen Model Screen Readers

The advent of the graphical desktop brought with it a host of problems for screen reading. When running in graphics mode, the video buffer is no longer a convenient source of text content that can be directly reported to the user. Instead, it is filled with red, green, and blue color values void of semantic information at the programmatic level. The content of the video buffer is now a visual gestalt.

The first screen readers for GUI desktops overcame this problem by extracting information from earlier stages of the rendering pipeline. Messages from applications to video drivers proved to be useful, though unwieldy descriptions of the visual display.

The SoundTrack system (Edwards, 1988) was an early attempt at generating an auditory display from this low-level source. As mentioned previously, SoundTrack simply played sounds and spoke the name of widgets as the mouse pointer passed over them, but this interaction proved extremely difficult for users with visual impairments. In contrast, the IBM Screen Reader for OS/2 (SR/2) used its render hooks to reorganize the content of GUI applications into a speech display reminiscent of console screen readers (Thatcher, 1994). Thatcher argued that GUIs and text-based interfaces were different ways of doing the same thing, and that familiar text-based access methods should be the basis of GUI access. This sentiment is echoed, to some extent, by desktop screen readers to this day.

A push for component reuse in the 1990s drove applications and desktop environments to provide methods of programmatic information access. Applications and desktop environments began providing APIs for querying and controlling data models and user interfaces across process boundaries. Today, the two most popular screen readers in existence, JAWS (`http://www.freedomscientific.com/fs_products/software_jaws.asp`) and WindowEyes (`http://www.gwmicro.com/Window-Eyes/`), use the native Windows operating system API, application specific APIs (e.g., Internet Explorer COM interface), and document model APIs (e.g., Internet Explorer DOM interface) as additional information sources (Blenkhorn and Evans, 2000).

Screen readers that observe multiple, disparate information sources create an aggregate *off-screen model* (OSM) (Schwerdtfeger, 1991). Such a screen reader then uses this model as the basis of its output to the user. For instance, the off-screen model in SR/2 is fed by one source, the video render pipeline, and organized to support row and column queries for text, much like a console. The off-screen models for JAWS and its commercial kin are far more complex. These OSMs are fed by a special video driver, one or more component or document object model APIs mentioned above, and at least one accessibility API mentioned in the next section. This off-screen model permits four common auditory displays of screen content:

1. *Focus, selection and caret tracking.* The screen reader reports the source of the last important on-screen event caused by user input or application feedback.

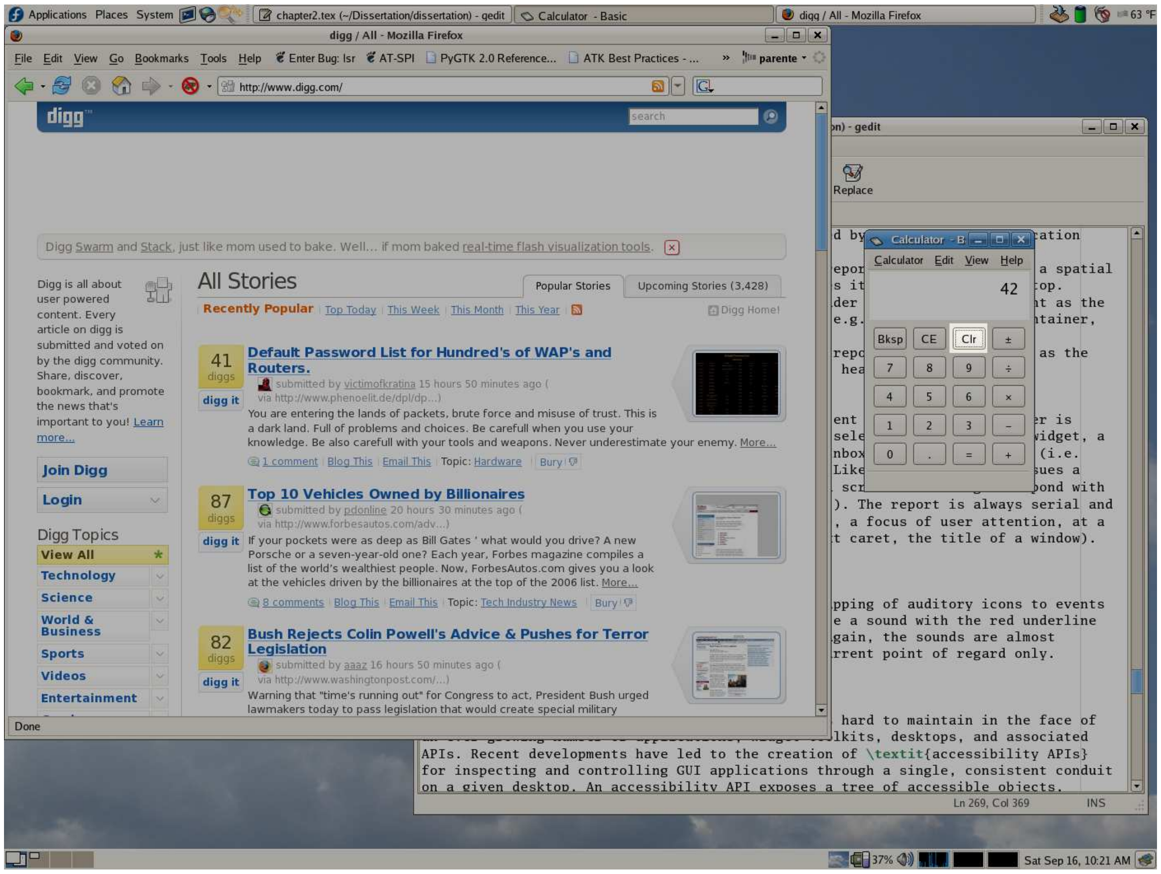
2. *Spatial review.* The screen reader reports the content under a spatial pointer such as the mouse pointer as the user moves it across the GUI desktop.

3. *Hierarchical review.* The screen reader reports widget content as the user explores them in a pre-order tree traversal (e.g., desktop, window, container, menu bar, file menu, and so on).

4. *Document review.* The screen reader reports document content as the user navigates by content type such as paragraphs, headings, and sections.

The primary auditory method for conveying the content of the OSM to the user is synthesized speech. For instance, when an item is selected in a tree view widget, a typical screen reader reports *tree item, Inbox, one of ten in two* (i.e., role, text, peer index, total peers, tree level). Likewise, when a user issues a request to *Read the next Web page link*, a screen reader might respond with *link, http://www.unc.edu* (i.e., role, text). The report is always serial and always concerned with one *point of regard*, a focus of user attention, at a time (e.g., a single widget, a character at the text caret, the title of a window). Figure 2.1 visually depicts the concept of a point of regard.

Some commercial screen readers also support the mapping of auditory icons to events and content. For instance, JAWS users can associate a sound with the red underline in Microsoft word to indicate a misspelled word. Again, the sounds are almost exclusively used to indicate information at the current point of regard only.



Digg Swarm and Stack, just like mom used to bake. Well... if mom baked real-time flash visualization tools.

Digg is all about user powered content. Every article on digg is submitted and voted on by the digg community. Share, discover, bookmark, and promote the news that's important to you! Learn more...

41 diggs digg it **Default Password List for Hundred's of WAP's and Routers.** submitted by victimofkratina 15 hours 50 minutes ago (via http://www.phenoelit.de/dpl/dp...) You are entering the lands of packets, brute force and misuse of trust. This is a dark land. Full of problems and choices. Be carefull when you use your knowledge. Be also carefull with your tools and weapons. Never underestimate your enemy. More... 1 comment | Blog This | Email This | Topic: Hardware | Bury

87 diggs digg it **Top 10 Vehicles Owned by Billionaires** submitted by pdonline 20 hours 30 minutes ago (via http://www.forbesautos.com/adv...) If your pockets were as deep as Bill Gates ' what would you drive? A new Porsche or a seven-year-old one? Each year, Forbes magazine compiles a list of the world's wealthiest people. Now, ForbesAutos.com gives you a look at the vehicles driven by the billionaires at the top of the 2006 list. More... 8 comments | Blog This | Email This | Topic: Tech Industry News | Bury

82 diggs digg it **Bush Rejects Colin Powell's Advice & Pushes for Terror Legislation** submitted by aaaz 16 hours 50 minutes ago (via http://www.washingtonpost.com/...) Warning that "time's running out" for Congress to act, President Bush urged lawmakers today to pass legislation that would create special military

```
d by                                          ation
epor                                          a spatial
s it                                          top.
der                                           t as the
e.g.                                          tainer,

repo                                          as the
 hea

ent                                           er is
sele                                          widget, a
nbox                                          (i.e.
Like                                          sues a
 scr                                          pond with
). The report is always serial and
, a focus of user attention, at a
t caret, the title of a window).


pping of auditory icons to events
e a sound with the red underline
gain, the sounds are almost
rrent point of regard only.


 hard to maintain in the face of
lkits, desktops, and associated
APIs. Recent developments have led to the creation of \textit{accessibility APIs}
for inspecting and controlling GUI applications through a single, consistent conduit
on a given desktop. An accessibility API exposes a tree of accessible objects.
```

Figure 2.1: Screen reader view of the GUI desktop. When the calculator window is activated and the *Clr* button receives the input focus, a screen reader reports *Calculator - Basic, push button, Clr*. The screen reader is typically silent about events and information outside this point of regard.

Accessibility API Screen Readers

The trouble with an off-screen model is that it is hard to maintain in the face of an ever-growing number of applications, widget toolkits, desktops, and associated APIs. Recent developments have led to the creation of *accessibility APIs* for inspecting and controlling GUI applications through a single, consistent conduit on a given desktop. An accessibility API exposes a tree of accessible objects, rooted at the desktop, each having attributes describing its role, state, attributes, and content, and methods for affecting them (e.g., Figure 2.2). The interface also includes support for observing important desktop events at the widget level such as text insertion, list item selection, the appearance of a new widget, and document reloading to name a few.

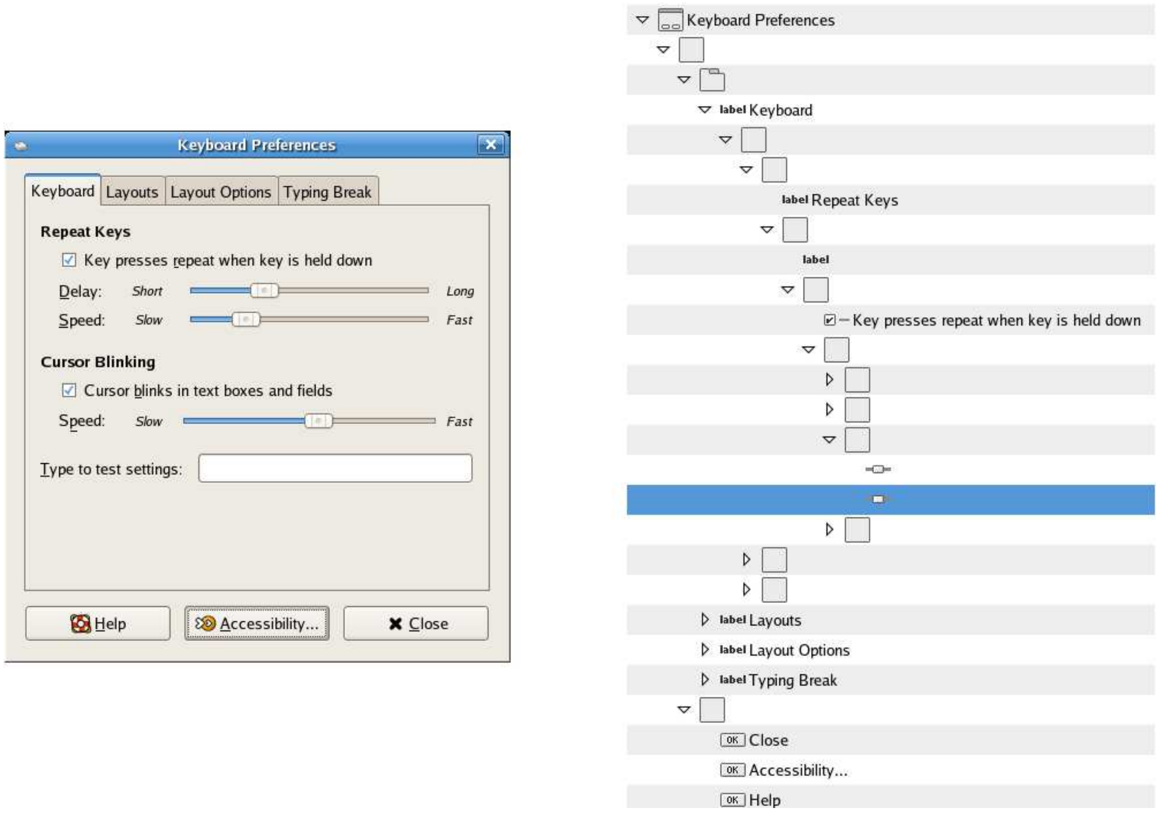


Figure 2.2: Accessibility hierarchy for a GUI dialog. The window on the left is the keyboard properties dialog from GNOME 2.14. The tree on the right shows the roles and names of the widgets exposed by the platform accessibility API. Note the existence of many unnamed fillers nodes that do not have corresponding visual counterparts.

Commercial OSM screen readers take advantage of accessibility APIs by using them as additional information feeds for their off-screen models. For instance, JAWS and WindowEyes both currently use the Microsoft Active Accessibility toolkit and plan to use Microsoft's improved Universal Interface Automation API as GUI information sources on the Windows desktop. More recent screen readers, however, discard the concept of an aggregate OSM and come to rely solely on accessibility APIs. For example, The VoiceOver screen reader (`http://www.apple.com/macosx/features/voiceover/`) for Apple's OS X desktop relies exclusively on the simply named Accessibility Framework for reporting information about application GUIs to the user. Similarly, the Gnopernicus (`http://`

`www.baum.ro/gnopernicus.html`), Orca (`http://live.gnome.org/Orca`), and Linux Screen Reader (Parente and Clippingdale, 2006) applications for the GNOME desktop environment use the Assistive Technology Service Provider Interface (AT-SPI) alone to report information to users with visual impairments.

Though accessibility APIs have eased the screen reader burden of maintaining a complex model, the screen reader view (i.e., the auditory display) has remained largely the same. A single stream of synthesized speech describes the current user point of regard, and, in some systems, auditory icons name events and widgets.

2.4.3 Improvements on Screen Reading

Screen reading is a low development cost method for making closed model, audio unaware applications accessible to people with visual impairments. Yet the usability of screen readers is limited by some critical factors:

1. *Application and platform cooperation.* The ability of a modern screen reader to inspect GUI applications is dependent on the support by the application, its widget toolkit, and the platform for standard accessibility interfaces. Lower-level hooks are currently being phased out for security reasons on platforms that still support them.

2. *Ties to the visual display.* Screen readers strive to report the exact content of the visual display to aid collaboration with sighted users. Whether reading the screen produces the best auditory display is not a primary concern.

3. *Loss of bandwidth.* Screen readers describe on-screen content with respect to a single point of regard at a time. Though events can clearly occur outside this narrow focus on the graphical, multitasking desktop, they are hardly ever reported in the auditory display.

4. *Tools assumed necessary and sufficient.* Screen readers do little to change the design of common graphical tools (e.g., the search dialog) to make them more suitable for interaction in audio.

5. *Speech overload.* Synthesized speech is used to convey almost all information.

Attempts have been made to overcome subsets of these problems without resorting to rewriting applications to make them audio aware.

Scriptable Screen Readers

Adding support for scripting to screen readers can help remedy problems with application accessibility and usability in audio. For example, the Linux Screen Reader has a script for the Metacity window manager that announces the names desktop windows when the user activates the window switcher (Figure 2.3), typically by pressing Alt-Tab. Without this script, the default screen reader logic would not announce the names of the windows because the window switcher gives focus to nameless icon widgets while showing the window names in an unfocused label widget. The script simply locates and reads the label text when the focus changes among icons instead of trying to read the icons themselves.

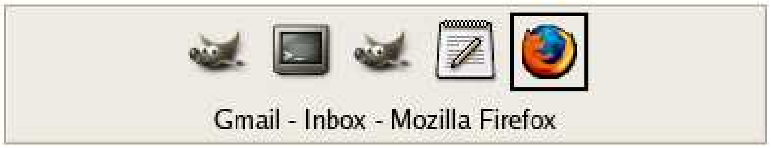


Figure 2.3: Metacity window switcher. The icons are unnamed in the accessibility hierarchy. A screen reader must be coded to read the label at the bottom of the dialog whenever a new icon is selected insead.

Scripting can also improve user control over applications. For instance, the JAWS screen reader has a script that defines new keyboard commands for Web page navigation in Microsoft Internet Explorer. Browsing Web pages is possible using the standard screen reader commands for reading by characters, words, and lines. However, user efficiency is enhanced by commands such as browsing by headings, links, form controls, frames, and other context specific actions.

Scripting is not without its problems, however. Scripts today are brittle in that if the structure of a GUI changes across versions, its associated script will likely break. The process of writing a script also does not generalize well as techniques to correct accessibility problems in one application may not work at all in another. For instance, the label for a text field may not be properly associated with the field in an application. A script may remedy this problem if a method of locating the correct label can be identified. If the label is located three levels up in the accessible hierarchy, the script can look at this level and announce the label properly when the text box receives focus. However, looking three levels up in the hierarchy is both application and version specific. A new release of the program may move the label elsewhere. A similar situation in an entirely different program may require completely different logic (e.g., looking two levels deeper instead of three levels higher).

Screen reader scripts are not employed to change the auditory display paradigm from single-stream narration of widgets. Essentially, such scripts exist to improve the ability of the screen reader to report on-screen visuals while offering shortcuts for common user commands.

Mercator

The Mercator system (Edwards et al., 1994; Mynatt and Weber, 1994; Mynatt, 1997) addresses the speech overload problem and relaxes some ties to the visual display. Instead of working directly with a graphical interface and listening to a screen reader narrate the interaction, a Mercator user browses the widget hierarchy while listening to auditory icons identifying their types. Mercator presents a list of auditory icons representing the widgets under a given branch at a certain depth of the hierarchy. The user browses this list using the keyboard. Upon hearing the auditory icon for a widget, the user may ask for a spoken report about the widget, select the widget to receive the application focus or activate it, or navigate into it if it is a non-leaf node in the hierarchy. Additional keyboard commands are available for returning to higher levels in the hierarchy, bookmarking key widgets, or jumping to the root of a window.

Mercator is interesting because it interposes itself as an auditory display between the user and graphical applications. When browsing, the user does not interact directly with the applications, but instead with the lists of auditory icons created by Mercator. Selection of a leaf widget in the list instructs Mercator to perform the necessary commands to give the widget focus or selection. In this manner, the user is shielded from some of the details of working directly with an audible GUI.

Nevertheless, the auditory display in Mercator is still based on graphical widgets, events, and interactions. In her dissertation on Mercator, Mynatt states that the display might be further improved by breaking all ties with the visual display:

> Although this work demonstrated creating a generic model of graphical interfaces, the models were expressed in terms of typical GUI constructs such as push buttons and windows. Modeling the graphical interface at this level helped support collaboration with sighted users since they also conceptualized the GUI at this level. *A more general goal is creating an abstract model of the graphical interface that is not expressed in terms of these constructs, but is expressed in terms of its affordances.* For example, what is the difference between selecting an item from a menu and selecting an item from a list? Abstract representations might be utilized for interface transformations as well as supporting reverse engineering of graphical interfaces. (Mynatt, 1997) (emphasis mine)

Emacspeak

The Emacspeak system (Raman, 1996b; Raman, 1996a; Raman, 1997) addresses the problems of application cooperation, ties to the visual display, use of visual tools in audio, and speech overload by adding audio output capabilities to the Emacs text editing environment. One of the core modules of Emacspeak is support for application-specific extensions that produce audio output for programs running in the Emacs environment (Raman, 2001). Unlike scripts for screen readers which run outside of the context of the target application, Emacspeak extensions use the Lisp *advice* facility, a system that allows arbitrary code to run before, after, or around functions inside the context of an Emacs application. This design allows Emacspeak extensions to produce audio displays based on application models, not their visual views, resulting in more natural auditory feedback.

The other core features of Emacspeak include low-level interfaces to audio devices and services for audio formatting of text using speech and auditory-icons. With these features, extensions can customize text parsing, pronunciation, and voice properties based on the semantics of text content. For instance, an extension aware of the current Emacs text editing mode can speak a line of Python code in a different manner than a line of English prose. Furthermore, extensions can use auditory icons as alerts, confirmations, indicators of change, nominal cues for content, prompts, and progress indicators (Raman, 2001).

System Access

System Access is a relatively new product from the Serotek Corporation (`http://www.serotek.com`). The system provides access to a fixed set of applications useful to a home user: email, chat, news, weather, and so on. While one could construct versions of these programs with dedicated auditory displays, System Access adapts existing GUI applications. But unlike screen readers, the resulting auditory display is not based on visuals, rather tasks users can accomplish in each application. This approach limits the number of applications users can access (the current version is not user scriptable), but provides an auditory experience customized specifically for the applications it does support.

2.5 Methods for Further Study

The preceding sections demonstrate the application of auditory displays to diverse problem domains. Though all of the systems discussed differ by design, it is possible to classify them along shared dimensions describing their techniques for display, support for user interaction, and methods of development. In the following sections, I use some of these dimensions as a means for identifying methods that might better balance auditory display usability with development effort. The topics that show promise here are the basis for the research, design, development, and evaluation described in the following chapters.

2.5.1 Built-in Versus Bolt-on

Creating software with built-in support for auditory display is currently hampered by the following:

1. *Primitive tools for audio output.* Numerous GUI widget toolkits exist as well as guides for creating usable, visual interfaces. In contrast, libraries for audio output function at a much lower level (e.g., play a waveform) and practitioner guidelines for creating usable auditory displays are still a topic for research.

2. *Lack of a strict separation between application model and view.* Graphical user interfaces are tightly integrated into most applications. Adding a new, auditory display requires a software rewrite.

3. *Limited visible use cases.* Graphical displays work well enough for most users in most situations. At present, auditory displays are developed for special-purpose applications (e.g., assistive technologies, telephony applications) not mainstream productivity software.

A purely technological solution may resolve the first problem. Overcoming the remaining two, however, requires changes to the current practice of application development. For instance, a move toward producing applications with open models and loosely-coupled user interfaces may make tying auditory displays to applications as easy as binding graphical ones. Moreover, such a shift may invite additional research on practical uses for auditory display. While these changes may benefit the development of auditory displays, they have yet to gain significant traction.

In this work, I follow a bolt-on approach to auditory display in order to side-step the latter two problems. A recent trend toward providing rich accessibility APIs for inspecting and controlling GUI software makes retrofitting tenable without direct access to the application model via an application-provided API. Support for the accessibility API is often provided at lower levels by the widget toolkit and desktop platform. As a result, an external agent can access information exposed by standard visual controls and format it for auditory presentation. Furthermore, the agent can provide higher-level tools for auditory output and account for all usability issues without repeating the work in each application.

This reasoning leads me to the following choices for further research and design. The remaining sections in this chapter state additional design choices in the same bullet list format.

- *Decision:* Use accessibility APIs to create a bolt-on auditory display for closed model, audio unaware GUI applications.

- *Decision:* Provide higher-level tools for sound output including synthesized speech, auditory icons and earcons, mixing, and spatialization.

2.5.2 Data Versus Task

Screen readers that currently use accessibility APIs focus on describing the graphical display to a listening user. The purpose of this design is to enable interaction across diverse applications and facilitate collaboration between sighted and blind users, both at the widget level. For instance, it is possible for a screen reader user to interpret and execute the instruction *Click the third button in the top toolbar* in various programs.

The problem with this design is that it produces an auditory display based on visual concepts a user cannot see or may never have seen. A usability study conducted by the Trace Center (Barnicle, 2000) identifies fifty-eight unique obstacles encountered by JAWS screen reader users of varying skill levels. Barnicle classifies these problems into five categories, and goes on to suggest possible sources and potential solutions as shown in Table 2.2.

Category	*Cause*	*Solution*
Efficiency	Navigation through complex interfaces, large collections, long documents	Better search tools, direct keyboard access
GUI controls	Lack of contextual information	Better semantic information describing widget meaning and intent
Unaware	Lack of contextual information	Notification of changes beyond the current point of regard
Wrong command	Lack of consistency	Consistent user experience across applications
Software bug	Lack of consistency	Consistent support for accessibility APIs

Table 2.2: Barnicle's (Barnicle, 2000) screen reader usability obstacles, their sources, and potential solutions. These empirical findings echo the first four screen reader shortcomings identified in Section 2.4.3.

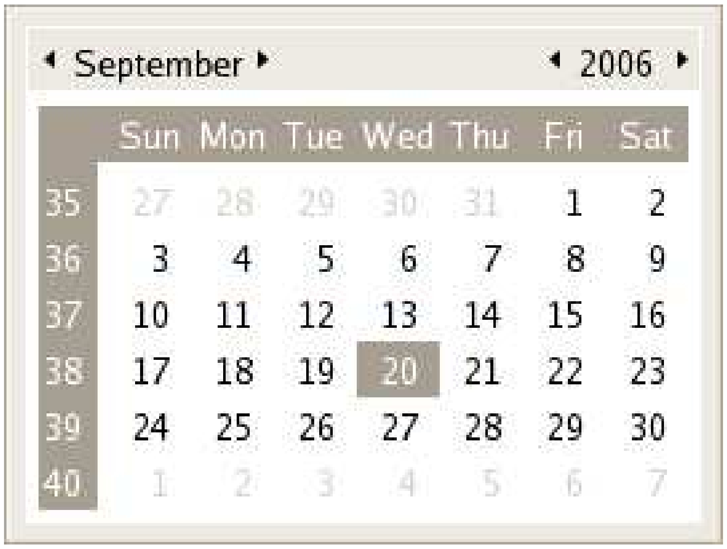
In my interpretation, these problems stem from the fact that GUI auditory agents are *data-oriented* auditory displays. Their goal is to describe widgets, the data of the graphical desktop. This method is akin to sonification in that the burden of interpreting the data falls to the user. For instance, when the widget in Figure 2.4 receives focus, a typical screen reader might report *table, table cell, column Wed, row 38, 20* (i.e., container role, role, column header, row header, cell contents). The user must then deduce by exploration that the point of regard:

- is in a calendar,
- on a Wednesday,
- on the twentieth day of September, and
- in the thirty-eighth week of the year 2006.

Not all of these facts are immediately apparent. The user must navigate to other widgets to learn the name of the month and the year, and use context to determine the table widget is serving the purpose of a calendar. Raman summarizes a similar example in his work on Emacspeak:

> A screen-reader speaks what is on the screen without conveying why it is there...The mismatch between features of the visual display and its aural counterpart are immediately apparent from the complete breakdown of the spoken output in this example. Yet, as humans we successfully speak dates with no difficulty so the breakdown is clearly not due to any inherent inability to speak this information. (Raman, 1997)

In contrast, the original graphical display is *task-oriented*. The date applet in Figure 2.4 uses visual widgets as a way of representing its supported tasks, namely browsing the days of a month and navigating across months and years. An equivalent, auditory



	Sun	Mon	Tue	Wed	Thu	Fri	Sat
35	27	28	29	30	31	1	2
36	3	4	5	6	7	8	9
37	10	11	12	13	14	15	16
38	17	18	19	20	21	22	23
39	24	25	26	27	28	29	30
40	1	2	3	4	5	6	7

Figure 2.4: The GNOME date applet.

display would enable these same tasks, but not necessarily with the same representation. For example, if the date applet were audio aware, it would likely report the point of regard more naturally: *Wednesday, September 20th, 2006. Week 38.* Furthermore, navigation to other days and months would probably not require navigation away from the calendar to different contexts. An auditory display would make switching months and years immediately apparent in its report, just as the GUI makes these functions visible alongside the view of the current month. To support keyboard input, the date applet might report *Use arrows to browse the current month. Page-Up/Page-Down to change month. Tab/Shift-Tab to change year.* Compared with the screen reader, the usability of this display would be significantly better for the following reasons:

- The description of the current task requires no guesswork or exploration to interpret.
- Affordances are immediately apparent.
- Visual concepts such as widgets are absent.
- Fewer input gestures are required to execute critical tasks.

In this research, I explore how producing task-oriented auditory displays for GUI applications might improve usability over the screen reader norm. This tactic requires a major departure from traditional screen reading: the system changes from being a passive reporter of the interaction between a user and a GUI, to an active agent interposed between the user and the underlying application. The agent described in this work, named Clique, is responsible for hiding the details of the graphical display from the user. It creates an auditory display designed to support the tasks in the underlying application and the abilities of the user as a listener. Since user attention and input are directed at the auditory display, Clique can provide a consistent experience across applications. Methods of information display and tools for browsing, searching, and querying information are unified across applications, not disjoint and based on visual widgets and dialogs.

The difficulty in this approach is the same one faced by model-based user interface generators: most GUI applications do not directly expose information about the tasks they support. Rather, they currently tend to use accessibility APIs to describe their graphical user interfaces. This mismatch between my goals and the available information suggests the need for an adaptation layer. In my work, I demonstrate how a simple scripting API can provide a task abstraction starting from information about on-screen widgets. The down-side to this approach is that development effort is required to write the scripts that enable Clique to operate at the task-level. However, I show that scripting requires far less work than rewriting GUI-based software from scratch to make it audio aware while retaining a significant increase in display usability over the "zero-effort" approach of screen reading.

The design of a typical screen reader and my proposed agent are depicted in model-view-controller form in Figures 2.5 and 2.6 respectively. Clique more closely approximates the design of an audio aware application shown in Figure 2.7. The user need only interact with Clique, not Clique and the GUI application. Furthermore, the model for the auditory display is based on a description of the tasks supported by the application model, not solely on its graphical view.

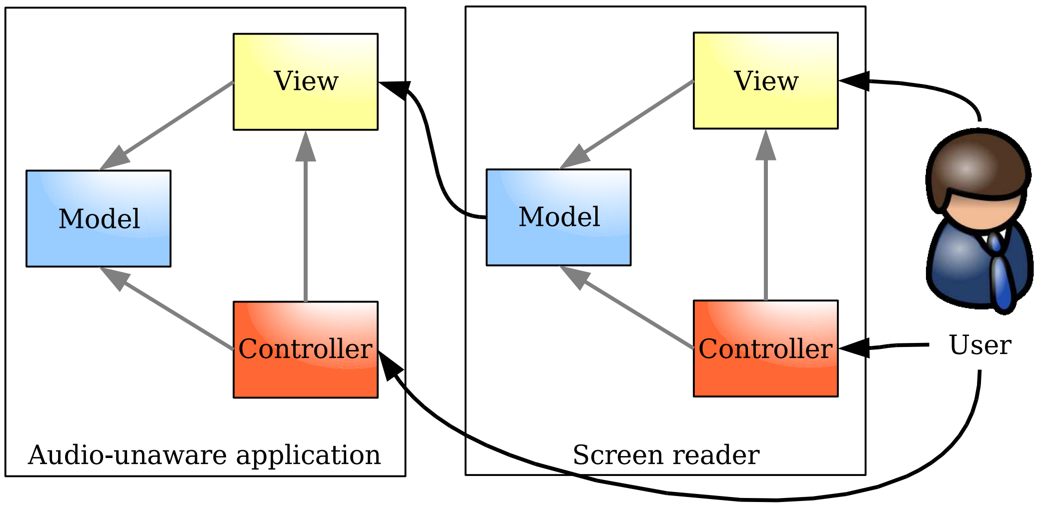


Figure 2.5: Screen reader model-view-controller diagram. The model of the screen reader is based on the graphical view. The user must interact directly with GUI applications and the screen reader while listening to its view.

I acknowledge that my chosen approach sacrifices ease of collaboration between sighted and blind users in terms of GUI widgets. However, I am unconvinced that users can work together effectively only at this level for two reasons. First, widgets are merely a convenient way of visually representing acts a user can perform (Novik and Pérez-Quinoñes, 1998). Users can recast any widget-based communication (e.g., *Please click the third menu item labeled "Write a new message"*) in more abstract terms (e.g., *Please start a new message*). Second, both loosely and tightly coupled collaboration are possible and important (Pinelle, 2004). For instance, users involved in tightly coupled work might benefit from a shared set of references such as labels on GUI widgets. On the other hand, users working on loosely coupled tasks in parallel may have need for common GUI terms only when synchronizing with one another.

In this research, I do not attempt to evaluate blind and sighted user collaboration when the auditory display is task-based versus screen-based. Nevertheless, I do not

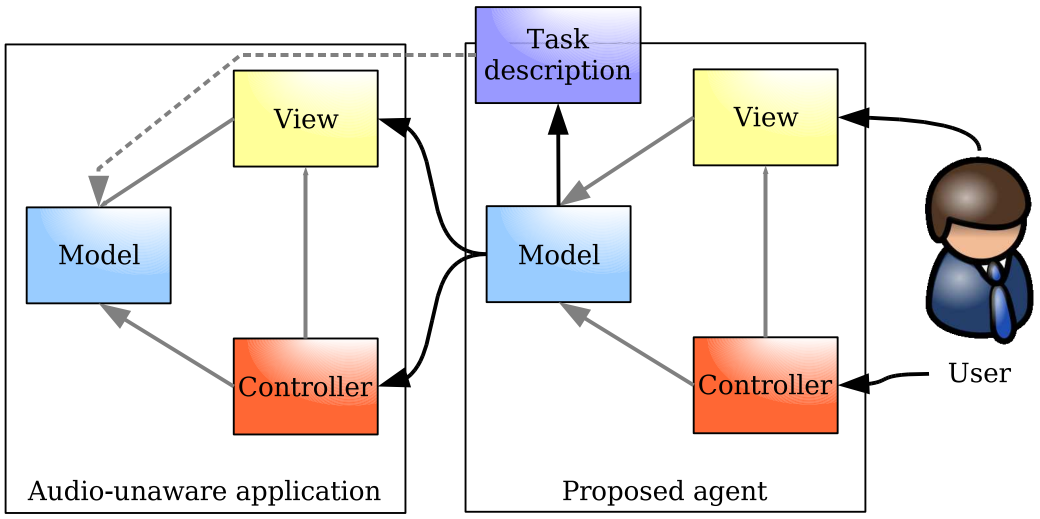


Figure 2.6: Proposed agent model-view-controller diagram. The model of the proposed agent is based on descriptions of the tasks represented by a graphical view. The user interacts solely with the agent while listening to its auditory view. The agent takes responsibility for updating the underlying applications.

believe a task-based auditory display generated from a visual one precludes all possibility of effective communication for the reasons given above.

- *Decision:* Interpose the auditory display between the user and GUI applications to unburden users from details of the graphical display.

- *Decision:* Provide consistent tools for browsing, searching, and querying information across tasks.

- *Decision:* Use scripting to support auditory display of user tasks rather than their graphical representation.

2.5.3 Speech Versus Iconic Sound

One of the key attributes of the graphical desktop environment is its effective use of both written language and visual icons. For instance, the text label *Delete* succinctly describes a supported command, while its placement on a button widget suggests how

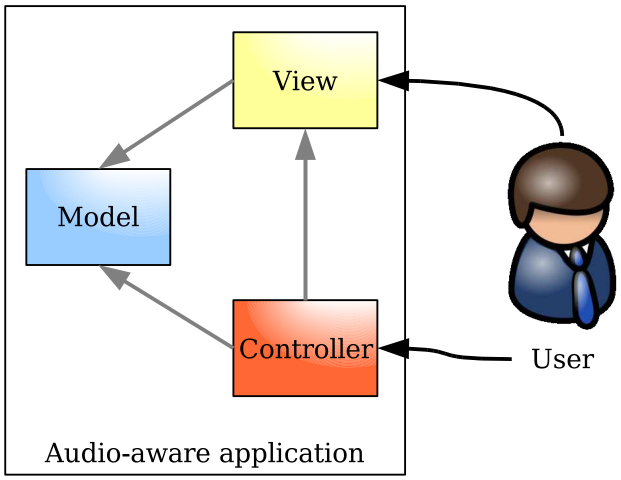


Figure 2.7: Audio aware application model-view-controller diagram. The audio view is built into the application and designed specifically for the application model. The user interaction directly with the application.

to invoke it. Paragraphs of text convey the information in a document to a user, while a paper icon represents the document as a whole for various file operations. Labels and icons are rendered within other, larger window icons representing sets of tasks or even entire applications. Text and icons are applied throughout the display in ways appropriate to the information represented, the supported user tasks, and the visual capabilities of the user.

An effective auditory display might take advantage of both lingual and iconic techniques as well. Synthesized speech, while powerful in communicating abstract concepts, is a sluggish output channel. Auditory icons, which are good at indicating nominal concepts, require learned mappings and have potentially ambiguous meanings. Earcons, useful for relating concepts, are difficult to interpret without training. Various systems have used these techniques to represent information, but few have blended them to the same effect as a GUI mixes text and icons. Screen readers, in particular, are heavily reliant on speech, even though the desktop they read is highly iconic.

I integrate earcons, auditory icons, and speech in the design of Clique to improve its presentation capabilities beyond what is possible with speech alone. I base the application of these techniques on their strengths and weaknesses with respect to both the type of information to be conveyed and the abilities of the listener.

- *Decision:* Use synthesized speech to convey information accurately.
- *Decision:* Use auditory icons, real-world sounds, to name concepts (e.g., applications).
- *Decision:* Use earcons, musical motives, to relate concepts (e.g., starting and ending a task).

2.5.4 Serial Versus Concurrent

Multitasking is one of the strengths of graphical environments. The simultaneous existence of multiple windows with numerous controls enables random access to visible information. A user need only look at the right spatial location. Furthermore, while a user is focusing on a particular widget in a particular window, other tasks can continue to run in the background and report information in the periphery. Popup tooltips, flashing windows, and spinning icons are common indicators of activity outside the user's point of regard. When designed properly, these methods of notification convey important information without annoying the user or distracting him or her for extended periods from the primary task.

Existing auditory agents produce output that is almost entirely serial. With the exception of a few auditory icons played alongside a single stream of speech, random access to information is prohibited. For instance, if a screen reader says *checkbox, Work offline, checked*, the user must wait for the first two utterances to complete before hearing the third.

Not only does a single stream force a waiting period on the user, it also limits user awareness to a single point of regard at a time. For instance, if a Web page finishes loading while the user is busy in another program, a single stream display has three choices for dealing with the system-driven event:

- Ignore the event.
- Announce the event immediately.
- Announce the event some time later.

The first choice precludes the user from ever dealing with the event. The second approach interrupts output about the user's current task, possibly to the annoyance of the user. The third tactic runs the risk of delivering stale information to the user. All three are less than ideal.

It is interesting to note that the second and third options are essentially equivalent to the first in screen readers. Screen readers nearly always interrupt their output when they encounter a user-driven event in order to appear responsive. That is, no matter when a screen reader decides to announce a system-driven event, the next user event will interrupt it. In the case of the Web page, if the user happens to issue a command just as the screen reader starts reporting that the page has loaded, the page load announcement will be unintentionally stopped without the user ever realizing it.

I use concurrent streams in Clique to reduce the time required to access desired information as well as to improve contextual awareness. Simultaneous streams of speech give users faster access to spoken segments that would otherwise appear much later in a serial stream. Mixing auditory icons and earcons with speech streams allow the auditory display to summarize activity without drawing lengthy attention from the primary task. I balance the use of concurrency with the listener's ability to process simultaneous sound streams.

- *Decision:* Use concurrent audio to extend user awareness beyond a single point of regard, reduce latency to select information, and support persistent audio.

2.5.5 Ephemeral Versus Persistent

Gaver (Gaver, 1997) describes visual objects as existing in space and over time. The content of a graphical display is *persistent*, but the physical dimensions of the screen limit how much of it can be displayed at once. The user may easily review what is visible by simply shifting his or her gaze over the screen. When content is hidden, it may be revealed with a few mouse clicks, and positioned such that it remains visible for quick review.

Sound, Gaver explains, exists in time and over space. Speech, auditory icons, and earcons in an auditory display are *ephemeral*, but where and when they are presented to a listener is practically unbounded when mixing and spatialization are available. But while multiple streams may be presented in parallel, their content may only be reviewed for as long as the streams are playing. Once a stream concludes, the user must either hold it in his or her memory or ask the auditory display to repeat it.

Continuous ambient sounds have been used in audio aware systems for sonification, mapping, and peripheral awareness, but have not yet been applied when retrofitting GUI software. Current auditory agents produce entirely transient displays. After describing the current point of regard, such a display falls silent until the next user action. If the user is distracted momentarily, did not understand a change in point of regard, or simply forgets the current context, he or she must ask the agent to repeat itself. When working across multiple tasks, remembering which task is active and its current state is an additional burden on the listening user. Alternatively, asking the display to repeat itself numerous time slows interaction. This lack of persistent contextual information is exactly one of the problems noted in Table 2.2.

I use continuous streams of audio in Clique as a way of providing persistent reminders about user tasks. Each task is uniquely identified by a continuous environmental sound indicating the active working context. Again, I balance the number of persistent streams active at any one time with the listener's ability to process multiple streams.

- *Decision:* Use persistent audio streams as identifiers for on-going tasks and processes, and aids for contextual awareness.

2.5.6 Assistive Tech Versus Universal Design

The intent of screen reader systems is to enable access to GUI applications for people with visual impairments. No sighted person uses a screen reader today. All retrofitting approaches based on accessibility APIs and on-screen widgets are considered *assistive technology*. To the contrary, people with vision do interact with audio aware software through telephones, on mobile devices, and in video games. When these applications allow people with and without vision to perform the same tasks through the same interface, the programs are considered to follow *universal design*.

The correlation between the bolt-on approach and users with special needs is interesting. Nothing innate to the retrofitting approach limits the potential user population to people with visual impairments. Granting sighted users with access to familiar GUI applications in eyes-busy situations would certainly be a boon. In addition, a larger user base for auditory agents would benefit both developers and users by driving up sales and lowering purchase costs. The unanswered question, therefore, is why auditory agents for closed-model applications are only used as assistive technologies.

I hypothesize that the usability problems associated with screen readers limit their audience to only those users who *must* use them. I posit that correcting these problems will not only improve computer access for people with visual impairments, but will also make the same, bolt-on auditory displays attractive to sighted users in situations not conducive to visual computing. I test this claim by evaluating both blind and sighted user efficiency and satisfaction when working with Clique.

- *Decision:* Design and evaluate the auditory display with both blind and sighted users.

2.5.7 Summary

This chapter reviewed the numerous techniques, designs, applications, and methods of production for auditory displays. The final section classified this previous work along dimensions of a design space for auditory information display. From this information, I made a series of design choices to pursue in my research in hope of better balancing auditory display usability and development cost. The next two chapters provide additional details on how these choices can be realized while satisfying the needs of the user as a office worker and a listener. The two chapters following describe how I implemented these choices in a software auditory display and how well they fare in evaluations of usability and development effort.

Chapter 3

Requirements of the Working User

This chapter states design requirements for an auditory display intended to support the user in the role of an office worker. The first section reviews the information seeking, content creation, and mediation behaviors of the user. The second section describes the design, execution, and results of a user study aimed at uncovering additional user needs and preferences specific to interaction in the auditory domain. The concluding section summarizes the behaviors identified throughout the chapter as a reference for following chapters.

3.1 Working Behaviors

A first requirement of designing an effective auditory display is to identify what tasks people are trying to accomplish and how they typically go about completing those tasks. Defining how an auditory display can enable the necessary behaviors for getting work done is only possible when both the tasks to be completed and the desired methods for completing them are known. The scope of this work, as outlined in Chapter 1, effectively answers the question of what tasks are of interest: office computing tasks such as writing email, Web browsing, file management, scheduling, and so forth. Answering how a display might effectively support such tasks is more involved and requires an understanding of how people work with a computer as a tool.

At a most basic level, a person's use of a computer consists of giving input to and getting output from the machine. Three broad categories can classify behaviors intended to advance current or future tasks toward completion:

- *Information seeking* is the process of exploring an information space to find content of varying levels of interest.
- *Content creation* is the process of encoding human knowledge in a digital form for later use.
- *Mediation* is the process of reconciling differences between the capabilities of the user and machine in hope of improving future interactions (e.g., asking for confirmation of an irreversible action).

The following three sections provide a basic review of previously developed models of these behaviors. The outcome of this review is a set of seeking, creation, and mediation behaviors that an auditory display should support, and, in some cases, recommendations for how it might support them.

3.1.1 Information Seeking

Information seeking in any electronic environment can be thought of as a set of related activities which a user may employ to satisfy his or her information needs. The current motivation for seeking information is one factor that determines what activities are selected for use. Numerous studies of browsing and searching behavior has led to an agreement on roughly three (Marchionini, 1995) or four (Choo et al., 2000; Wilson, 1997) basic modes of seeking. These modes range from casual, almost random browsing of information to highly-selective, intentional search for key content. The most recent work by (Choo et al., 2000) names these modes as follows:

- *Undirected viewing* occurs when a user is exposed to information with no specific need in mind.

- *Conditioned viewing* occurs when a user directs viewing to information about selected topics.

- *Informal search* occurs when a user actively looks for information to deepen knowledge and understanding of a specific issue.

- *Formal search* occurs when a user makes a deliberate effort to obtain specific information about a particular issue.

It is important realize that these modes are not mutually-exclusive over the duration of a task (Catledge and Pitkow, 1995). User needs vary from casual perusal to deliberate information intake and back again as the knowledge about the task at hand changes (Bates, 1989). Similarly, differing motivations may be present across user tasks in a multitasking environment. An effective interface must account for these motivations as well as rapid transitions among them.

The progress made toward an informational goal is a second factor that determines the selection of activities. Seeking can be broken down into subprocesses that the user performs while satisfying an information need. Early stages such as defining the problem and choosing a search system lead to later activities such as examining search results and extracting information. More advanced stages are more likely to be encountered when earlier stages have been completed or are at least underway. Marchionini describes one such decomposition of the seeking process into stages that develop in parallel (Marchionini, 1995):

- Recognize and accept an information problem.

- Understand the problem.

- Choose a search system.

- Formulate a search query.

- Execute the search.
- Examine the results.
- Extract information.
- Reflect, iterate, or stop.

Ellis names a somewhat similar set of stages in his research on Web seeking behaviors of scientists and engineers in an industrial firm (Ellis and Haugan, 1997):

- Starting, or identifying sources of interest.
- Chaining, or following-up on references in material.
- Browsing, or scanning top-level content.
- Differentiating, or filtering information based on its perceived usefulness.
- Monitoring, or receiving reports from sources of interest.
- Extracting, or systematically working through a source for content of interest.

While Marchionini's model is expressed as a series of steps that are iterated until completion, Ellis' breakdown represents only a loose ordering of activities, or *moves*, in the seeking process. These moves are acquiescent with the four modes of searching and browsing stated earlier: the user may transition among any of them them at any time, both within and across tasks. Once again, an interface must offer support for performing these activities as well as switching among them.

Combining Choo's four seeking modes with Ellis' six moves yields a behavioral model of information seeking based on user motivations and activities. The rows and columns of Table 3.1 represent such a crossing. Choo, et al. successfully used this model to both predict and describe user seeking behaviors in a Web environment by entering typical examples of Web browser use in the table cells (Choo et al., 2000). Their research revealed a correlation between modes and activities such that starting and chaining moves primarily occurred during undirected viewing; browsing and differentiating mostly occurred during conditioned viewing; differentiating and extracting were the primary activities during informal search; and extracting was the main move during formal search.

Assuming the correlation identified by Choo, et al. holds reasonably true in electronic environments beyond the Web, their model can be used as a predictor for general seeking behaviors. The cells in Table 3.1 contain domain and medium agnostic examples of activities that an information display should support.

	Start	*Chain*	*Browse*	*Diff.*	*Monitor*	*Extract*
Undirected viewing	Identifying collections of interest	Reading summaries of collections				
Directed viewing			Reading summaries of content within collections	Reviewing known content	Tracking content changes	
Informal search				Reviewing known content	Tracking content changes	Reading relevant content of interest
Formal search					Tracking content changes	Reading targeted content of interest

Table 3.1: Information seeking modes and moves. The cells in the table show examples of domain and medium independent behaviors.

A Framework to Support Information Seeking

The research just reviewed identifies behaviors of users as information seekers but makes few concrete suggestions about how software should support those behaviors. More guidance for designing user interfaces to best support browsing and searching is required.

One approach is to rely on the use of metaphor to help users transfer knowledge from other domains, such as the physical world, to the digital environment. The Audio Rooms project demonstrates one application of metaphor, relating interconnected, acoustical environments to the organization of file system data (Mynatt and Edwards, 1995). The Web Forager project represents another use of metaphor for structuring the presentation of Web pages as pages in books that can be thumbed through, placed on a desktop, or filed in a bookshelf (Card et al., 1996). Choosing an appropriate metaphor is a loosely defined activity, however, and may prove difficult if no source domain matches the target very well (Carroll et al., 1997). Furthermore, a selected metaphor says nothing about what aspects of the information space a particular design should emphasize.

Another approach is to build an interface to accommodate empirical evidence of seeking behavior within a particular task domain. For example, noting that users follow a hub-and-spoke style of navigation on the Web suggests that Web sites should be designed with the home page acting as a hub and provide prominent links to oft visited spoke pages (Catledge and Pitkow, 1995). In another example, evidence that information seeking behaviors on the Web often parallel foraging behaviors studied by biologists (Pirolli and Card, 1999) implies that increasing *information scent* can lead users to information of interest more quickly just as increasing olfactory scent can draw one to food (Koman, 1998). The problem with designing an information space around empirical results is that data must exist for a domain and medium of interest. Such is not currently the case for auditory displays of office computing applications, except for results indicating that literally reading the screen causes usability problems (Barnicle, 2000; Edwards, 1991).

A framework that offers domain and modality independent advice for structuring an information seeking interface is better suited for this line of research. The AgileViews framework satisfies these criteria by suggesting an interface structure segmented into six views of an information space (Geisler, 2003; Marchionini et al., 2000).

1. An *overview* summarizes the entire information space.

2. A *preview* summarizes a particular piece or pieces of content.

3. A *primary view* details a particular piece or pieces of content.

4. A *peripheral view* notes information related to that currently displayed in another view.

5. A *review* notes content previously displayed in another view.

6. A *shared view* depicts other people's view of a piece or pieces of content.

The six AgileViews are supportive of the seeking behaviors discussed earlier. For instance, in the starting phase, an overview of the information space is important in helping the user decide whether and where information of interest might be found. In the directed chaining and browsing phases, previews and peripheral view assist in deciding what information to peruse next. Reviews are particularly important in the differentiating and monitoring phases as comparisons must be drawn between previously viewed and current content of interest. The primary view is most important during extraction since it provides the most detailed information.

The AgileViews structure has been applied to at least three interface designs. The Enriched Links project demonstrated how visual pop-ups could provide previews, overviews, and shared views of linked, but yet-to-be-visited Web pages to supplement the standard primary Web page view (Geisler, 2000). The initial findings from the Audio Enriched Links project showed that speech previews of linked Web pages could reduce the number sidetracks made by users with visual impairments while seeking a target page (Parente, 2004). The Web interface for the Open Video Project provided storyboard overviews, fast-forward and excerpt previews, and related link peripheral views as supplements to standard video streams (Geisler et al., 2002). The successful application of AgileViews across these diverse information spaces and media suggests its potential for structuring an information seeking auditory display.

3.1.2 Content Creation

The converse of finding and consuming digital information is creating it. Images, movies, sounds, and music are as commonplace as more traditional character-based text on personal computers today. While supporting the creation of each of these types of content is important, the constraints outlined in Chapter 1 limit the scrope of this work to text content alone. The following review of content creation, therefore, focuses on the text editing behavior of users. Nevertheless, Chapter 7 includes some speculation on how to support editing and creating of additional types of content.

Text Editing

Most of what is known about the text editing comes from the analysis of line and screen based word processors performed in the early 1980s (Roberts and Moran, 1983; Embley and Nagy, 1981). Though modern GUI text editors differ significantly in appearance from their predecessors, support for the fundamental operations are still the same: entering, deleting, and navigating. At this basic level, the results of studies of editing with non-GUI applications remain valid today. Moreover, the limited bandwidth of the audio channel makes audio-based editors more akin to the line editors of yesterday than the visually oriented, *what you see is what you get* (WYSIWYG) editors of today.

The minimal features required in any text editor include the ability to insert new characters, review and move the point of insertion, and delete previously entered characters. These fundamental actions are usually mapped to simple keystrokes. Since manipulating text a character at a time is tedious for large bodies of text, most text editors extend these concepts to words, lines, sentences, paragraphs, and even arbitrary chunks of text to improve editing efficiency. Text selections enable replacement and movement of groups of characters and key chords are often used to navigate by more than one character at a time. Modern GUI editors also support the use of a pointing device to indicate chunks of text and jump the text caret to new locations. However, since the functionality of a pointer such as a mouse is inherently tied to vision, it is of no interest in this discussion.

The analysis of text editing as practiced by knowledge workers and administrative assistants over 178 active person-hours of work revealed that the use of editor features is grossly non-uniform (Whiteside et al., 1982). One-half of all keystrokes were used to insert or append characters, one-fourth to move the text caret to a new location, one-eighth to delete characters, and the remaining fraction for all other advanced features combined. These findings were consistent across workers, tasks, and two different editor implementations suggesting these common features should be heavily optimized. Furthermore, the study data revealed that when users switched away from one activity (e.g. entry) to another (e.g. deletion) they tended to return the previous activity next instead of selecting another one. Maximizing the efficiency of alternating between pairs of editing subtasks in audio is, therefore, worth some effort.

Text Editing via Speech Displays

The basic features mentioned above have been mirrored in speech displays for text editing. For example, editing support has been a critical aspect of screen readers since their inception (Thatcher, 1994). In lieu of a displaying a visual caret, screen readers announce the current character, word, or line of text touched by the caret. Navigation proceeds according to the same concepts with the current character, word, or line being announced as the caret moves. Insertion and deleted of text results in announcements of what is inserted and what is deleted. The emacspeak system (Raman, 1997; Raman, 1996a; Raman, 1996b) provides similar concepts of browsing and editing text at various levels of granularity. In addition, different syntactic units are spoken using different voices as a cue for distinguishing them, and task-specific definitions of text chunks are defined to enable more efficient navigation and feedback (e.g. navigation by blocks in source code).

Though not studied directly, the relative percentages of keystrokes used to insert, move, delete and perform other actions are likely to be the same in editors with speech displays as in visual editors. The modality of the editing feedback is unlikely to change how much of each activity the user performs to complete an editing task. The amount of time spent performing each action, on the other hand, is likely to be greater in audio because of the relatively slow speed of spoken versus visual feedback. Hence, limiting how much feedback is required to communicate the results of an editing action to the user is a reasonable requirement for supporting text editing in audio.

3.1.3 Mediation

Mediation is the act of rectifying differences between two partners to achieve a common goal. Human communication is laced with mediation gestures that serve to pace, repair, structure, and validate a conversation. For instance, if I say to my friend "Let's meet at the corner restaurant for lunch," I will require some feedback indicating that he has understood the request, is free to join me around lunch time, knows where the restaurant is located, and so on before an actual lunch engagement can be made or rejected. Moreover, I may need to ask more questions or make additional statements if I suspect we do not have a common understanding or have encountered problems when making lunch dates in the past. I must also give my friend time to think about, acknowledge, or question my request before I interject with more information (Brennan, 1998).

Human-computer interaction, though an impoverished form of communication in comparison to human conversation, is both rapid and complex enough to require mediation behaviors. In any computer interface, some portion of the input to and output from the computer does not convey task-related content, but is rather intended to improve the process of communication itself. For instance, some command line programs display a growing series of dots to indicate progress and to ensure that the user understands some process is taking a while to complete. Many GUI programs pop-up confirmation dialogs asking the user to confirm or cancel the last action performed before making an irreversible change. Complex GUI programs often allow the user to map key presses to commonly used commands to speed future interactions.

Studying the mediation behaviors present in the interaction between two humans can inform the design of an effective auditory display. First, research has indicated that people tend to anthropomorphize computers during use. The expectation that the computer will help mediate the interaction is a direct consequence of this user view (Bickmore, 2004). Second, models of human conversation have been successfully applied to the design of speech based interfaces in the past such as the Danish Dialog System (Bernsen et al., 1998), SpeechActs (Martin et al., 1996), and Chatter (Ly and Schmandt, 1994). Though these systems focused on speech input in addition to output, their methods of cooperating with the user are just as valuable to the design of audio output alone.

Cooperation and Grounding

In any purposeful conversation, two partners adhere to the *principle of mutual responsibility* (Clark and Wilkes-Gibbs, 1986). The participants try to establish, roughly by the initiation of each new contribution, the mutual belief that the listeners have understood what the speaker meant in the last utterance to a criterion sufficient for current purposes. If, for instance, my friend does not respond within a reasonable amount of time after my proposition of "Let's meet at the corner restaurant for lunch," I am bound to follow up by asking if heard me, if he is too busy, if he does not want to go, and so on. If he acknowledges with "OK," then it is now my responsibility to propose or ask for a mutually agreeable time. If he instead says "OK. I'm free at 1 PM for an hour," then he has both contributed his interest in my initial request and his set of constraints to which I must respond. These coordinated contributions will continue until a decision has been reached about going to lunch.

The responsibility of *grounding* a conversation, or working to establish a shared state of knowledge, must also be shared in human-computer interaction. If the computer does not play a part in grounding, the user will be overburdened with checking the state of the machine after every action (Brennan, 1998; Brennan and Hulteen, 1995). Brennan gives an excellent example:

> Consider what happens when Don returns from lunch and logs into his computer. He means to copy some files into a public directory so that his supervisor can review them. He types, "copy report.97 public." The system returns a prompt. Then he copies another file by typing, "copy budget.97 public." Again, a prompt. Later, he is surprised to discover that *public* does not contain his two intended files after all. It turns out that he had forgotten to create a *directory* called "public" before trying to copy his files, and instead wrote the files, one after another, into a *file* named "public," the second file overwriting the first. Many DOS and UNIX users have experienced this kind of mishap. They soon learn to check to see whether their commands have had the desired effect ... Such checking behavior is a way of grounding with an uncooperative operating system.

A simple correction to this problem would be to report the actual consequences of Don's command (e.g. "Copied x to file y"). Direct manipulation interfaces, when designed appropriately, excel at providing this kind of timely feedback (Shneiderman, 1992). Don would have a hard time reproducing this mistake in a GUI desktop environment. The absence of the public folder as a target for drag-and-drop, the need to rename the two files to "public" after copying, and the overwrite confirmation dialog would act together to prevent him from completing his task incorrectly. No checking behavior is required as the interface actively contributes to the grounding process.

Deciding how much feedback is sufficient for both parties is another aspect of cooperative interaction. Each conversational partner maintains a dynamic *grounding criteria* that determines how much information is given or sought based on past interactions (Brennan, 1998; Brennan and Hulteen, 1995; Clark and Wilkes-Gibbs, 1986). If my friend and I eat lunch together often at 1 PM, I would feel less inclined to decide on a specific time to meet. If Don had experienced problems copying files previously, he would be apt to seek more immediate confirmation of his actions.

Timely and appropriate feedback is an equally, if not more important requirement in an auditory display. Checking the results of every action by issuing a query and listening to a stream of speech is very tedious work for the user. On the other hand, automatically speaking the consequences of every action is likely to grow annoying and slow interaction. The auditory display must find middle ground between these two extremes.

Conversational Maxims

How two partners carry out the grounding process is another topic of interest. Grice first described a set of maxims followed by two conversation parts to ensure a cooperative exchange of information (Grice, 1975). Though the idea of grounding was not developed until later, his maxims can be seen as guidelines for maximizing the efficiency of the grounding process:

- Quantity
  - Make your contribution as informative as required.
  - Do not make your contribution more informative than is required.
- Quality
  - Do not say what you believe to be false.

  - Do not say that for which you lack adequate evidence.

- Relation
  - Be relevant.

- Manner
  - Avoid obscurity of expression.
  - Avoid ambiguity.
  - Be brief.
  - Be orderly.

Bernsen, et al. (Bernsen et al., 1998) extended the Gricean Maxims for use in structuring spoken feedback from telephony applications in the Danish Dialog System. As evidenced below, the additional maxims were added primarily to account for the asymmetry in abilities and knowledge between the user and the computer and the need for conversational repair:

- Informativeness
  - Be fully explicit in communicating to users the commitments they have made.
  - Provide feedback on each piece of information provided by the user.

- Manner
  - Provide the same formulation of the same question (or address) to users everywhere in the system's interaction turns.

- Partner asymmetry
  - Inform the users of important non-normal characteristics which they should take into account in order to behave cooperatively in spoken interaction. Ensure the feasibility of what is required of them.

  - Provide clear and comprehensible communication of what the system can and cannot do.
  - Provide clear and sufficient instructions to users on how to interact with the system.

- Background knowledge
  - Take partners' relevant background knowledge into account.
  - Take into account possible (and possibly erroneous) user inferences by analogy from related task domains.
  - Distinguish, whenever possible, between the needs of novice and expert users (user-adaptive interaction).
  - Take into account legitimate partner expectations as to your own background knowledge.
  - Provide sufficient task domain knowledge and inference.

- Repair and classification
  - Enable repair or clarification meta-communication in case of communication failure.
  - Initiate repair meta-communication if system understanding has failed.
  - Initiate clarification meta-communication in case of inconsistent user input.
  - Initiate clarification meta-communication in case of ambiguous user input.

The original plus extended maxims provide a basis for making an auditory display *feel* cooperative to the user. They complement the knowledge as to what information should be emphasized in the output from the machine (information seeking) and what kinds of input should be optimized (content creation, grounding contributions) with guidance as to how and when output should be given and input accepted.

Multitasking and Interruptions

Modern office computing involves a tremendous amount of multitasking. Consider an administrative assistant who is composing an email to a colleague when he suddenly receives an instant message from his boss. The boss asks the assistant to schedule an emergency meeting with the employees in the division later that day. He also requests that the meeting notice include a link to the Web page showing the division's successful sales for the last quarter, and that he be notified when the email has been sent. The urgency and authority of the boss's request might cause the assistant to interrupt his current task and follow a procedure such as the following:

- start a new email to the workers in his division,
- begin a Web search for the quarterly report,
- inform his boss that he's working on his new task,
- copy the link to the report Web page into the new email,
- send the meeting notice email,
- notify his boss that the task is complete, and
- return to his initial task of emailing a colleague.

The assistant must serialize some dependent subtasks (e.g. send the meeting notice after it contains the link) but will likely work on some tasks in parallel (e.g. confirm the boss's request by instant message while waiting for the Web search to complete). The computer must support these serialization, parallelization, and rapid task switching behaviors.

The theory of discourse structure proposed by Grosz and Sidner (Grosz and Sidner, 1986) develops a model that can describe how two conversational partners manage multiple tasks. Their theory represents a conversation in terms of three components:

1. *linguistic structure*, or the sequence of utterances divided into discourse segments,
2. *intentional structure*, or the purposes of the the entire discourse and its segments, and
3. *attentional state*, or the current focus of the conversation on the objects, properties, and relationships of a particular segment.

A key concept in this theory is that of the *focus space stack*. Grosz and Sidner explain that the conversational participants maintain one or more stacks of focus spaces, or sets of references, associated with discourse segments. By each contribution, the participants either refer to concepts in the active focus space at the top of the main stack, push a new focus space onto the main stack by introducing a new discourse segment and its related concepts, pop the active focus space off the top of the main stack by achieving its intended purpose, move one or more focus spaces off the main stack to an auxiliary stack for later completion, or move one or more focus spaces off an auxiliary stack to the main stack to resume a suspended discourse segment. Figure 3.1 depicts a possible progression of the focus space stack for the administrative assistant example given above, assuming the assistant and the computer are two conversational participants and the user tasks are the discourse segments.

The idea of using one or more dialog stacks was applied in both the Chatter (Ly and Schmandt, 1994) and SpeechActs (Martin et al., 1996; Yankelovich et al., 1995; Yankelovich, 1994) interactive speech systems. Both systems maintain one or more stacks of dialog contexts which the user or computer may implicitly push, pop, move, or refer to by his or her contribution to the interaction. For instance, the user might say "Start a new email to John" causing the computer to push a new email dialog onto the top of the main stack. Next, the user might say "Go back to the Web browser" to push the new email dialog onto an auxiliary stack and make the Web browser dialog the active one at the top of the main stack. Later, the computer might announce "Incoming

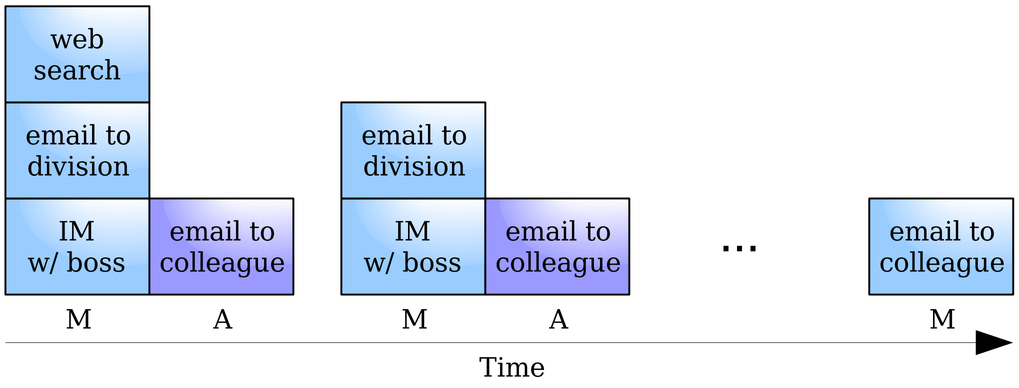


Figure 3.1: Focus space stack progression over time. Focus spaces representing the assistant's subtasks needed to fulfill the boss's request are pushed onto and popped from the main stack (M). An auxiliary stack (A) is used to maintain the assistant's original task of emailing a colleague. Only pops are shown in this diagram.

call from Jane" and let the user decide whether to put the call at the top of the main stack or move it to an auxiliary stack.

Though an auditory display does not necessitate speech input on the part of the user, structuring the output in terms of active (main stack, top frame); related but inactive (main stack, lower frames); and unrelated, inactive (auxiliary stack frames) task dialogs is still possible. The success of this approach in past systems and evidence for its existence in human to human conversation make it an attractive solution for supporting multitasking and interruptions.

3.2 Interaction with an Ideal Speech Display

The discussion above provides general knowledge about the behavior of working users without particular attention to the domain of their tasks or the modality of interaction. To identify behaviors specific to doing office computer work via an auditory display, I conducted a qualitative study of the interaction between computer users and an ideal speech display: a human expert controlling a standard PC. The goal of this study was to determine what support users need to complete tasks such as checking and writing email, browsing the Web, scheduling appointments, and managing files when the ability of the display to provide requested information is not a limiting factor. Audio recordings made during the study reveal that, even in this ideal condition, users have difficulty remembering task content long enough to make use of it, knowing what to do next without suggestions, and breaking away from graphical user interface concepts. The recordings also indicate that users filter and summarize large bodies of information, navigate by searching, manage out-of-context alerts, and batch repetitive operations. These elicited behaviors both complement and supplement the user models already discussed.

3.2.1 Design

The study was structured as a role playing scenario involving a single participant and me in each session. I asked the participant to assume the role of a personal assistant to the chief executive officer (CEO) of the imaginary XYZ corporation. I explained that the four responsibilities of the CEO's assistant were the following:

1. Completing the tasks assigned by the CEO via email.
2. Helping fellow XYZ employees schedule meetings with the CEO.
3. Answering emails from people outside the company.
4. Running errands by car or on foot for the CEO.

I stated that, in the past, the assistant had found it difficult to meet the first three requirements while carrying out the fourth. The assistant simply could not walk or drive safely while using the laptop, and had to either take frequent breaks to work on the laptop or save all the computer work for when he or she returned to the office. The CEO was not particularly happy with either solution.

I went on to describe the CEO's latest attempt at overcoming this problem. The CEO had recently purchased a new software package for the assistant's laptop. Once installed, the package would allow the assistant work with any program on the laptop by conversing with it as if it were a real, intelligent person. I stated that I would be playing the part of this new application throughout the scenario.

I explained that the CEO had high hopes for the new software. The CEO believed it would allow the assistant to continue running errands while answering email, scheduling appointments, and looking up information on the Web. To test this theory, the CEO was now asking the assistant to perform an initial, thirty minute evaluation of the new software before taking it on the road. The participant was told he or she would be performing this evaluation in the role of the assistant.

To summarize, I reiterated that I would be acting as an audio-only intermediary between the participant and a typical desktop computer. The participant would not be allowed to see the computer screen or touch its mouse and keyboard. Instead, the participant would be expected to speak to me to give commands and ask questions, which I would carry out on the computer or answer appropriately. I stated that the participant's goal was to see how many tasks assigned by email he or she could successfully complete during the thirty minute trial.

Rules

After introducing the scenario, I read aloud the rules to which I would adhere while playing the part of the auditory display:

- I would perform user commands exactly as specified, if possible.
- I would remember information, if specified (e.g. "Skip all emails from bob@xyz.com", "Remember that URL for later").

- I would speak in order to respond to user questions and to report feedback from an application.
- I would never interrupt the user but will always allow the user to interrupt me.
- I would notify the user of important events during breaks in the conversation.
- I would ask the user to clarify vague commands (e.g. "Handle all my email for me").
- I would guide the user out of intractable problems.
- I would not look at the user in order to prevent communication via back-channels (e.g. hand gestures, facial expressions).

To further clarify the rules and give the participant an idea of how the interaction might proceed, I next played three pre-recorded audio examples. The audio segments featured a female user conversing with a male auditory display in the following manner:

1. The user asks me to open email program, start a new message, states the recipient, and asked what other information is needed to complete the email. The auditory display reports the other fields that need to be filled in before the message can be sent.
2. The user asks me to open a Web browser, interrupts the display as it starts to read the first page that loads, and tells the display to announce only the title of the page and the number of links when a page first loads. The display responds by providing the request page name and number of links, then announces that three new messages just arrived in the email program. The user asks to hear the subjects of the new messages.
3. The user asks me to return to her email program and requests that the auditory display read the sender and subject of the current message. The display states the

sender and subject. The user asks for the body of the message, and, upon hearing that it contains a link, asks the display to activate it immediately.

Tasks

The tasks to be completed by the participant were initiated by email messages received by the participant throughout the study. At the start of the scenario, ten such emails were located in the participant's inbox awaiting processing. Up to four new emails arrived later during the session at predetermined times or in response to user actions. Table 3.2 gives an overview of the emails sorted by their default ordering in the inbox.

Not every email was reviewed or received by every participant. Some participants ran out of time before reading all of the emails. Other participants never satisfied the pre-conditions for later messages.

Discussion

At the completion of the scenario, I asked each participant the following series of questions about the experience. Additional questions were asked as appropriate to help clarify some responses and delve deeper into topics of interest to the study.

- Do you have any comments or questions?
- What task was the easiest to complete? The hardest?
- How well do you think you did in completing the assigned tasks?
- What questions did you ask repeatedly? What commands did you give repeatedly?
- What feedback from the auditory display did you find useful? Useless?
- How did this experience compare with the way you normally use a computer?
- What did you like the most about the interaction? Like the least?

#	*Request*	*Possible solution*
1	CEO requests 1/10/05 be set aside in his calendar for a site visit, asks that all current appointments be rescheduled	Visit calendar, block off day for site visit, email two people to reschedule their meetings
4	XYZ employee Leo Marcos asks for a copy of an email the assistant sent to him a week earlier about the new XYZ product	Find message in sent mailbox, forward to requester
5	CEO asks that the proposal document attached to an email from Bob Brown be forwarded to him when it is received	
6	CEO informs the assistant of the location drive, folder, and zip file of the new company catalog (H drive, documents folder, catalog zip), states which file in the zip can be sent to people outside the company (catalog for customers)	
8	Non-XYZ employee asks for a copy of the company catalog	Attach file found in drive, folder, zip archive given in #6
9	XYZ employee Amy Georges asks to set up a meeting for 1/10/05 at 11 AM, asks for a suggested meeting time as close to her preferred time as possible but no earlier if her preferred time is not available	Visit calendar, review appointments on 1/11/05, suggest 1 PM
11	XYZ employee Bob Brown says he is unable to send the proposal via email, provides a link to his Web site, username, and password for the assistant to download it	Follow link, find proposal link on page, enter username and password, report link broken to Bob
12	XYZ employee Leo Marcos says thanks for the forwarded copy	
13	CEO asks for a direct link to the video library Web page on the UNC-CS department Web site	Visit UNC homepage, visit local search page, search for "video library," visit first search result link, copy URL, email URL to CEO
14	XYZ employee Amy Georges accepts the suggested time slot or reiterates her desire for a suggestion	

Table 3.2: User tasks in the ideal auditory display study assigned by email. Email numbers missing from the table were spam messages that could be ignored or deleted. Receipt of the final four emails was dependent on completion of previous tasks.

- Do you ever listen to news or music while performing other tasks? If so, what tasks?
- Would you passively listen to your email or Web pages while performing those tasks? Would you actively browse them?

3.2.2 Execution

Participants

I recruited twenty people familiar with sending and receiving email, browsing the Web, and managing files to participate in our study. Seven participants (four male, three female) had a visual impairment while thirteen participants (ten male, three female) did not. Four participants with visual impairments (one male, three female) had been blind since childhood while the remaining three developed the impairment later in life. All but one of the participants with visual impairments relied on a screen reader in their daily computer use. The exception was a user who had enough residual vision to interact with the computer by sitting very close to the display or with the help of a magnifying lens.

Sessions with the first three participants (two males with vision, one female without) were deemed pilots. These sessions were used to debug the test procedure and give me practice in the role of the auditory display. The results from the pilot sessions were not considered in the analysis below.

Setting

All twenty, one-hour study sessions were conducted over a two month period in the spring of 2005. Sixteen of the twenty sessions were performed in a dedicated lab space at UNC where auditory and visual distractions were nearly non-existent. The remaining four sessions were conducted at the Library for the Blind in Raleigh, NC to accommodate participants who could not travel to UNC. A room at the library was dedicated to our use and helped keep distractions to an acceptable minimum.

In both environments, a computer was on hand to assist me in the role of the auditory display. Applications representative of those needed to complete the required tasks, copies of the emails assigning tasks, and mock-ups of the necessary XYZ company Web pages were pre-installed on the computer. The participant and principle investigator were seated such that participant could not read the computer screen. Participants were allowed to see me to reinforce the idea that they were interacting with an unconstrained auditory display. To avoid communication with the participant via backchannels, I consciously avoided looking at the participant, making gestures, or changing my facial expression.

Analysis

After running all of the study sessions, I transcribed each of the audio recordings into plain text files. Each transcription included the communication between participant and me for the thirty minute scenario and follow-up discussion. In addition, the transcriptions indicated any pauses longer than three seconds during the scenario and any laughter or other non-speech vocalizations. Once complete, I reviewed the transcriptions for accuracy while listening to the original audio recordings.

I used Weft QDA (`http://www.pressure.to/qda/`) to code segments of each transcription according to patterns of behavior seen across participants. Twenty-one codes resulted from this first pass. I next analyzed the initial code batch for the purpose of grouping codes into more general categories and eliminating codes not related to my current research. Seven themes emerged from this analysis. Finally, I coded each transcription again with labels representing these seven categories. Table 2 gives the initial and final codes.

Initial code	*Final code*
Memory aid	Memory
Memory problem	
Reference problem	Prompting
Autohelp	
Perfect understanding	
Visual thinking	Visual thinking
Summary	Summaries
Filtering	
Speed	Searching
Ordering	
Alert	Alerts
Batch operation	Batch operations
Mobility	Uncoded
Pause	
Like	
Dislike	
Confusion	
Chanel saturation	
Novice experience	
Voice input	
Task switch	

Table 3.3: Initial and final codes for the ideal display study.

Validation

To validate my codings, I used a triangulation technique. I first recruited one former and one current computer science graduate student to aid me. Each person was familiar with my research but not knowledgeable about the details of this study. I gave each validator a unique simple random sample of fifty segments drawn out of the total population of coded segments (520) and a random sample of uncoded segments. I also provided the validators with a list of the seven final codes, their definitions, and a few fabricated conversation segments exemplifying each. I then asked the validators to label their samples with zero, one, or more of the codes per segment.

I next compared the validator codings with my original codings. For each segment in a sample, I counted the number matches between the validator codes and my codes.

I summed the number of matches in a sample and divided by the number of codes I assigned to the segment to produce a scoring metric:

$$score = (overlap)/(mine)$$

Using this metric, I found that 63% of the codes from the first validator and 80% of the codes for the second validator matched my eight-category coding. These scores give me some confidence that the themes I identified do indeed exist in the transcripts.

It is important to note two shortcomings of this metric. First, it provides no information about cases where the validators assigned more codes than I did. I purposely chose to exclude this information from the metric so that it would only validate my codings, not suggest additional codings per segment. Second, the metric is negatively affected by differences in interpretations of each theme. For instance, I used the memory code to label segments in which participants used the auditory display as a memory aid. On the other hand, the first validator consistently used the prompt code in these cases resulting in a 6% reduction of the first score. This reduction, and possibly others, is artificial with respect to indicating theme discordance. In this case, both the validator and I noted a theme present across a number of segments, but labeled it differently.

3.2.3 Results

Of the seventeen non-pilot participants [1], only three (SM03, SM09, SM15) correctly completed all six tasks requiring non-trivial solutions (see entries with possible solutions in Table 3.2). Out of the remaining participants, six people (SF05, SM06, SF10, SM11, BF14, BM22) completed five of the tasks, five people (SF07, BM08, SM12, SM13, BF23) completed four, one person (BF20) completed three, and two people (SM04, BF21)

[1]In the participant IDs, B and S indicate users with and without visual impairments respectively and M and F indicate gender. The numbers are unique identifiers.

completed exactly one task. Five of the seventeen ran out of time before completing all of the tasks while the others quit early believing they had completed all of their work.

The codings of the study transcriptions provide a more detailed view of the performance of the participants. The following sections explain the themes identified by my codings. Each section is laced with a non-exhaustive set of examples demonstrating the theme discussed. The word choices and grammar in the examples are taken from the original transcriptions without editing in order to preserve the flow of the original conversations. Since the behavior of participants across divisions by gender and impairment was quite similar, the participants are considered as a single population in most cases. Divisions are mentioned only when they add value to the discussion.

Working Memory

The transience of auditory information (What did you say?) is a known source of interaction problems (Lai and Yankelovich, 2003). In this study, the transience property resulted in participants repeatedly forgetting information before it could be put to use in completing a task (183 segments). Participants often forgot critical advice given in one email by the time it was needed to fulfill a request in a later email. For example, the complete location of the company catalog given by the CEO in email #6 was often forgotten by the time it was needed to answer email #8:

> *SF07 - User incorrectly attaches entire zip file instead of the catalog in it*
>
> User: Click on document folders.
>
> Alright. There's one file here.
>
> User: Is it called catalog.zip?
>
> Yes it is.
>
> User: Attach that file. OK. Go back to mail program. Go to next message.

> *SF10 - User forgets catalog filename in zip archive*
>
> OK. What do you want to attach?

User: H drive, the document folder.

Alright.

User: Open the zip.

OK. It's open.

User: And send catalog.pdf.

There's no catalog.pdf.

User: What? OK, read me the CEO product catalog email one more time.

The fact that participants forgot information across emails is not surprising since the length of time between reviewing the two related emails could be quite large. However, participants also tended to forget key content immediately after reading it. For instance, as participants began to act on the task in email #11, they failed to recall the information needed to complete it:

SM11 - User forgets password

User: Follow link proposal.

OK. And a window comes up. It says its title is prompt label is enter xyz username and password.

User: Enter username xyz. Password. [6 sec pause] Go back to my email program.

OK.

User: Read me the last email.

The same problem arose in completing the tasks for emails #1 and #9. Some participants did not make note of the all day site visit in #1 after rescheduling all existing appointments. Many participants failed to suggest a new meeting time or check for availability when responding to #9:

BM22 - User forgets that Amy specified alternative meeting times

User: Send her a, reply to her, that "The CEO has a site visit and will be tied up all day."

OK.

User: "Does she have any other preferred meeting times? Please advise."

OK.

Participants who recognized the limitations of their working memory coped with the problem in three ways. First, when faced with memory loss, some participants interrupted their current task, switched contexts to locate the missing information, and then resumed the primary task with the forgotten information in mind. This method proved tedious and time consuming as some tasks had to be interrupted multiple times to review and remember multiple chunks of information (e.g. Web site URL, target filename, username, password, what to do with the file once found, what to do if not found). Participants commented on the shortcomings of this approach in the post-scenario discussion:

> SM04: I had to recall and check a lot of things and to check something you had to go through all the steps of switch to inbox, find the message, open the message, and read it again. Which ... took some time to and concentration to focus on accessing those messages.

> BM08: The hardest task to complete was, is remembering where I am. OK? In other words, when I went from an email to a Web site and then, then I had to say OK. Now, what was I supposed to be looking for on this Web site.

> SM12: I repeatedly asked what was just said in an email, basically, so I'm trying to complete some task that might require a couple of steps and I would need to remember exactly which steps I want to be taking.

> BF23: Well, actually because I found myself trying to as, as you were reading the emails I found myself trying to figure out exactly what I'm going to say

to do what needs to be done. Then I forget what needed to be done just that, you know, because I'm trying to figure out what it is I'm trying to say. [ha ha]

Second, participants guessed forgotten content with the expectation that the auditory display would correct them based on the partial information given in their guesses. This strategy was effective only because the auditory display was task-aware and knew what information the participant had forgotten.

SM04 - User specifies "outside the company" to recall a name

User: OK. Good. OK. So actually we'll go back to the calendar.

Mhmm.

User: And and tell the guy from the outside company Frank or something with F.

Bill Johnson.

User: Bill Johnson. OK. No F. So if we could email Bill Johnson.

Third, participants occasionally asked the auditory display to remember information for them. This approach was the most effective of the three as it alleviated some of the user's memory burden and made his or her intentions explicit to the display (i.e. it did not have to infer what information the user had forgotten or wanted to recall at a later time.

SM09 - User asks to remember key information

It says [message body content through password].

User: Remember that link, username, and password.

OK. It says [last sentence].

User: Open that link.

SF05 - User asks to enter previously remembered information

User: Hm. Click on the proposal link.

OK. And a dialog comes up title “prompt” label “enter”

User: OK. Enter the username and password given in the email.

OK. Alright. Entered. Want me to OK the dialog?

User: Yea. Click OK.

Participants also commented on this strategy in the follow-up discussion. The second comment offers an explanation of why more participants did not use the display as a memory resource.

SM13: Probably could have used you more for remembering stuff instead of, you know, I’d be going one thing and then go back and read that email again instead of just, you know, like, remember some of the stuff in the email.

BF20: I guess I was thinking that it’s like I’m supposed to be telling you what to do and I’m supposed to remember. You know, I didn’t, I didn’t think to, to, to say well, oh, that’s right, you’re looking at it or you can, or you can remember it or whatever. I was trying to do it in my head. And instead of using the resource that was available I guess.

Prompting and discovery

The invisibility of information (What can I do here?) is a second source of problems in auditory displays (Lai and Yankelovich, 2003). During our scenario, participants had trouble discovering avenues to take to advance their current task because of the invisibility of potential actions. Many times, users started their exploration by asking very specific questions requiring yes or no answers. In most situations, this approach was inefficient as it generated only one potentially new bit of information per exchange.

SM09 - User asks four yes/no questions

User: Look for a zip file with the name of the file that was in the email.

There's no proposal.zip on the page.

User: Are there any subfolders?

There's no folders on a Web page.

User: Ah, there's no link to a zip file on that Web page?

Nothing explicitly saying proposal.zip.

User: Are there any links on that Web page?

Yes.

After asking a few such narrow questions and failing to identify a potential next step, participants began to broaden their questions. Less directed questions generally proved more useful in helping participants decide what to do next.

SM03 - User asks two yes/no questions before broadening

User: Is there any link on the page with the name of "videos?"

No.

User: OK. Is there a search box?

No.

User: OK. [4 sec pause] How many links are there?

Roughly 30 I'll say.

User: OK. Forget this. Go to www.google.com.

The most common method of discovering available options, however, was to fall back on display initiated prompts (40 segments). As stated in the study rules (see Section 2), I promised to guide the user out of intractable problems. Since all users had trouble completing at least one task during the scenario, every user received help from the display at least once. But instead of stating a next correct move toward completing a task (something a task-unaware display would never know), the display asked the user a question introducing one or more novel possibilities for what to do next. Prompts were also used to improve inefficiencies in the interaction after they were repeated multiple times.

> *SF10 - Display offers a summary of a Web page to reveal new options*
> Alright. And you got to a back. [sic] A page comes up that says Bob Brown at XYZ Corp.
> User: OK. Enter. Is there a place to put a pass, username and password?
> No. There's no text boxes. Would you like to know what's on the page?
> User: Yes, please.

> *SM03 - Display offers to remember oft-forgotten and reviewed information*
> User: OK. Go back to that first message and find me that date that the CEO told me to schedule.
> OK. The date is January 10th, 2005. Would you like me to remember that?
> User: Yes, please.

When asked what aspects of the interaction they found useful and useless, participants often made positive comments about the display initiated prompts and never stated that they were useless.

> SF05: The clarification stuff was very useful, like when I was trying to get to my sent items folder and I thought it was called something else and you said, you know, "Would you like me to read you a list of what mailboxes you have?" Cues like that were very useful.

> SF07: The use, the more useful parts where I was like making a mistake and the interface like gave me a cue to say that this was a mistake. Like if I tried to open some Web page and I said go to this link and then it said this link doesn't exist so that told me something wrong.

It is important to realize, though, that users might not view prompts initiated by a less ideal, software auditory display as favorably. Nearly all of my prompts were specific to the participant's current task and problem. More general prompts for potential actions might be seen as less useful or even annoying.

Visual Thinking

When struggling to find the next step toward completing a task, sighted participants sometimes introduced visual concepts into the interaction or reported that they were trying to picture a visual interface for their current task (17 segments). These participants appeared to fall-back on their memory of what a task might look like when faced with difficulty completing it in audio.

> *SM12 - User mentions minimizing*
>
> User: [6 sec pause] This is hard. Minimize that request. Minimize that item and go back to the original email and read the subject, read the email to me again.

> *SM03 - User tries to visualize Google search page*
>
> User: Ah. OK. Search for the text search results.
>
> OK. Not found.
>
> User: Damn. I gotta remember what the Google search page looks like. Alright. So. Is there. Search for the text "results."
>
> OK. OK. Yep. Found.

Users with visual impairments mentioned nearly as many visual terms throughout the scenario (14 segments). At first, this result seemed counterintuitive as I expected these users to avoid talking about less intuitive visual concepts. I realized, however, that these participants a forced to deal GUI concepts daily as screen readers are designed to mimic the exact features of the visual display.

BM08 - User introduces maximizing and minimizing when switching tasks

User: OK. Minimize the scheduling application. Maximize the email. And respond to it and tell her that there's a sales meeting at the same time that she wants to schedule a meeting. She needs to know. Hold on just a second.

OK.

User: Stop. Maximize the scheduling application.

Sure.

...

User: OK. Minimize the scheduling application. Maximize the email application. Tell her we have an opening for 12 o'clock, will that be acceptable?

BM22 - User implies network is obscured by other tasks

OK. Alright. Minimize everything and go to My Network Places.

What is interesting about these results is that most of the visual concepts introduced were unnecessary. For instance, there was never a need for a participant to minimize a window before attending to another task: the idea of a window occlusion is foreign in an auditory display. I never told participants that they had to performs these actions and sometimes even responded that they were unneeded. Still, participants continued to mention them, making the interaction less efficient than need be.

Some sighted participants also commented on the lack of persistent visual cues during the scenario and in the concluding conversation. These segments give additional evidence that users were frustrated with trying to remember too much information.

SM11: All that stuff without visual cues to say, oh, there's a little bold faced email up there, I need to go back. I had to remember, at the very least, I had to remember I needed to ask if there was anything else to do.

SF07: ... I think I asked you to open the inbox many times and repeatedly read through the messages to make sure I had processed every email. Because that's the biggest thing I missed. Like on a visual, I missed the visual cue for, because if I was looking at the screen I could pinpoint immediately if one, if there was some message I hadn't processed but I just had to go through again and again to make sure that I had looked at every message and taken care of every message and wasn't skipping anything.

Content summaries

All seventeen participants used summaries to gain an overview of large bodies of content and to reduce the amount of time spent weeding through content irrelevant to the current task (142 segments). The first goal was accomplished by asking questions about the quantity of a certain type of content. Participants used quantities such as the total number of email messages, the length of a single message, the total number of hyperlinks on a page, and the amount of a Web page left unread to prioritize tasks and weigh alternatives for completing them.

SM11 - User decides to skip reading a long message for the time being

User: How long is that message?

Four minutes maybe.

User: What's the next message? Subject?

BM22 - User decides to finish reading a page

User: How many, how much, how much bigger is, how big, much bigger is the page?

There's four more links past that.

User: What are they?

Participants accomplished the second goal by asking questions intended to filter content according to a set of guidelines. Participants selectively reviewed content based on attributes such as the author of an email, the existence of an email attachment, and the type of content on a Web page in order to avoid dealing with currently unimportant information.

SM06 - User filters by sender and time, then by existence of an attachment

User: Find all of the emails sent by Leo in the past week.

Sent by Leo?

User: Sent to Leo in the past week.

OK. And there are two.

User: Do either of them have attachments?

Yes. One does.

User: Select that email. Forward it.

A couple participants took even more advantage of intelligent filtering by asking the auditory display to apply permanent filters to their queries. For instance, one user customized the reading of the Google search results page to include only the search result links. Another indicated he only wanted to hear the senders and subjects of email messages and then, perhaps, more information.

SM06 - User asks to customize the reading of Google search results

User: From now on when you read link from, when you read Google, read the titles of the page, titles of the pages with links that come after the word "results".

OK. So all links that come after the word "results?"

User: Yes.

OK.

User: And no links such as “cached” or “view as html.”

OK.

Some users commented on the benefits of the summaries in avoiding lengthy reports and in customizing the auditory display. The second statement below was made during a comparison of the participant’s previous experience with screen readers and her experience in the study scenario.

SF10: Well, it was helpful ... when you asked if I wanted a summary of the page instead of just starting to read everything.

BF14: I just have the overwhelming thought that, you know, it wasn’t bombarding me with stuff, you know, it was specifically what I asked for and that was very, very nice. ... Like I said, I really like the fact that it tells me what I want to know. It’s very succinct.

Navigation by Search

In addition to using summaries to sift through content, participants employed two search strategies to locate details of interest (63 segments). The first approach amounted to a keyword search with a user-specified partial target name or a property serving as the search term.

BF14 - User searches for a partial email address and asks for more details

User: [4 sec pause] Find next message from CEO.

OK. The next one. It’s found. So you want me to read the subject or message body or?

User: Read the message body.

The second method of searching used recently reviewed content as the search query. For instance, a participant would ask to search for the email address “he just heard” or a the name of a file "mentioned in the last email." This approach relieved the burden of remembering key terms from the user and is an implicit version of the third memory-shortage coping strategy mentioned previously (see Section 3.2.3).

> *BF23 - User asks to find a message sent to the sender of a recently read email*
>
> User: Find sender. Find message sent to sender of the open email.
>
> OK. Yep. Leo Marcos. There's two of them.
>
> User: OK. Read subject.

Unobtrusive Alerts

In the rules of the scenario, I stated that the auditory display would inform the participant of any important events during breaks in the conversation. These notifications often concerned the arrival of new emails and asynchronous error dialogs while users were off completing unrelated tasks. Users reacted to these notifications by either handling them immediately or ignoring them for the time being (51 segments).

> *SM15 - User continues current task after an alert*
>
> User: Return to the windows explorer.
>
> And a new email just arrived for you.
>
> User: OK. I'll ignore it for now. Go into the documents folder.

> *BF20 - User suspends current task after an alert*
>
> A page comes up titled publications UNC-CH computer science.
>
> User: Computer science?
>
> Mhm. That's the title of the page.
>
> User: OK.
>
> And a new email just showed up for you.
>
> User: Oh. Let's check the email and get out of this.

Some participants expressed favorable opinions toward the notifications. One participant even said that he regretted not being interrupted by new mail alerts because he was confused about new messages appearing unnannounced in his inbox.

> SM04: [5 sec pause] It was good how it was just a subtle message that there was a new message as opposed to just leaving it at that was good.

> BF14: I'm used to so much feedback from my current system (JAWS) and a lot of that is useless. But. I found it highly useful to know when a new message arrived.

On the other hand, one participant did state that he found notifications pointless in this particular scenario.

> SM03: In the setting that I was working in, you know for this, the fact that it told me whenever a new message popped up was pretty useless. Because I was, you know, just going through them in a list and I would have gotten there eventually. If I had been doing something else, you know, like I was doing word processing for a while and then it told me a new, a new email came, that would have been useful ...

Batch operations

Finally, a few participants created batches of simple operations that they later applied to complete similar tasks. The use of batched actions appeared in only a few places (9 segments), but at least one participant found them noteworthy for their potential in eliminating redundant tasks.

> *SM03 - User asks to send a similar message to another recipient*
> User: Then go on to the next entry in the calendar program on that date.
> Sure. This one says [second entry content].

> User: OK. Email that person with the same message body as the previous email.
>
> OK.
>
> User: And the same subject as well.
>
> Same subject. And it's done. Would you like me to send it?
>
> User: Yes.

> SM09: And there was another one where the response with a whole bunch of text that went, oh right, I just said send another message to that person with the same body. That's an amazing shortcut that, so kind of the amazing shortcuts were the easiest.

These results are aligned with those reported by Bennett in his study of user interaction with a human speech interface for circuit diagrams (Bennett and Edwards, 1998). Bennett found that his users tended to explore diagrams element by element rather than by larger structures. Few participants in this study attempted to group work into larger tasks, mirroring Bennett's observation of a user tendency toward using low-level, atomic interactions.

3.2.4 Conclusion

The results of this study suggest a number of interaction problems and preferences for which an auditory display should account. User working memory appears to be stressed both by the amount of task-critical information encountered and the length of time between its discovery and its use. A computer display, with practically infinite, perfect memory, should provide some facility to offload the burden of memorization and recall from the user. For instance, a speech-only display might provide an audio tape-recorder feature allowing a user to record snippets of speech from the display and review them later while completing tasks. A display capable of using non-speech sounds might pair ambient sounds with certain tasks in order to improve user awareness of the current context and aid later recolection of information from that context (Tan et al., 2001).

Users appear to be unaware of what features are provided by the auditory display or what avenues are available for advancing their current task. An audio display should make clear its capabilities (Bernsen et al., 1998) and suggest possible actions to take toward completing a task. Announcing this information automatically is a potential solution for helping novice users, but one that is likely to annoy people more familiar with the display. Providing a user command that causes the display to announce its features and possible actions in the current context is a reasonable alternative, as long as users are aware of its presence.

User searching and browsing behaviors noticed during the study mirror the models of such behaviors described earlier in the chapter. Participants appear to switch among browsing summaries of content, browsing filtered content, and searching for specific content. The transition among these behaviors appears seemless and free-foem such that browsing does not necessarily precede searching and vice versa. To support such free, rapid movement among navigation techniques, a display should avoid having explicit modes for searching versus browsing and require little input from the user in specifying search or summary criteria. For instance, if a user is giving input to a display using a keyboard, he or she might specify a search criteria by holding a modifier key while typing a search string. The display might respond to each character typed with informatoin about the first match. Continuing to hold the modifier while pressing the arrow keys might move among the available matches. To return to browsing, the user would only need to release the modifier key (Raskin, 2000).

Most users in the study appear to desire timely notifications of important events outside the current task. The display must manage the flow of alerts to avoid constantly distracting the user uncessarily and allowing alerts to grow stale. The display should also consider the importance and relevance of alerts in their delivery to the user and presentation. For instance, an auditory display might allow a user to select sources of information that should be monitored and announced no matter what the user is doing. A single-channel auditory display might batch and make these announcements during breaks in the interaction. A multi-channel display might use speech or sound to alert the user immediately to a change in a monitored source, even while the primary channel is actively giving output.

Few users appear to improve the efficiency of repetitive tasks by asking the display to remember lengthy sequences of commands and repeat them at later times. More users might take advantage of such a feature, however, if its availability were made plain. The "sluggishness" of interaction with an audio display caused by the speed of human speech might be offset by enabling complex commands to be batched and executed with only one confirmation of the command at the very end. A software auditory display might support this ability by, for example, allowing users to record macros and replay them later in part or in full.

Finally, sighted users appear to desire ties from an auditory display to the visual display when memory is taxed or methods of completing a task are unclear. Blind users also appear to refer to visual concepts, but likely only as a matter of training and convention. Providing memory aids and the ability to query features of the auditory display may implicitly mitigate the need to introduce visual concepts into the audio interaction.

3.3 Summary of Behaviors

Table 3.4 summarizes the behaviors discussed in the proceeding sections. The table lists guidelines for supporting each behavior and references the literature suggesting each guideline.

	Guidelines	*References*
1	Allow casual and directed browsing	(Choo et al., 2000; Wilson, 1997; Marchionini, 1995)
2	Support searching given partial information	(Ellis and Haugan, 1997), user study: searching
3	Make switching between browsing and searching effortless	(Catledge and Pitkow, 1995), user study: searching
4	Provide a "home" for starting the seeking process	(Choo et al., 2000)
5	Elucidate connections between content objects	(Choo et al., 2000; Geisler, 2003; Marchionini et al., 2000)
6	Enable filtering of content objects	(Choo et al., 2000), user study: searching
7	Notify users about relevant content changes	(Choo et al., 2000), user study: alerts
8	Support the transfer of content objects to new contexts	user study: memory, batch operations
9	Allow for reviews of past content	(Geisler, 2003; Marchionini et al., 2000), user study: memory
10	Summarize content objects with previews	(Geisler, 2003; Marchionini et al., 2000), user study: summaries
11	Establish context with overviews	(Geisler, 2003; Marchionini et al., 2000)
12	Provide details on demand	(Geisler, 2003; Marchionini et al., 2000)
13	Let queries and commands be directed at particular views	(Geisler, 2003; Marchionini et al., 2000)
14	Support rapid transitions among views	(Geisler, 2003; Marchionini et al., 2000)
15	Allow navigation through text by character, word, and chunk	(Raman, 1997; Thatcher, 1994)
16	Participate in grounding	(Brennan, 1998), user study: prompting
17	Contribute information at an appropriate level of detail	(Bernsen et al., 1998; Brennan and Hulteen, 1995; Grice, 1975)
18	Support multitasking with interruptions and task switches	(Martin et al., 1996; Ly and Schmandt, 1994; Grosz and Sidner, 1986)
19	Let queries and commands be directed at particular tasks	(Martin et al., 1996; Ly and Schmandt, 1994; Grosz and Sidner, 1986)
20	Respect maxims of quality and manner	(Bernsen et al., 1998; Grice, 1975)

	Guidelines	*References*
21	Explain limitations of the interface	(Bernsen et al., 1998), user study: prompting
22	Enable repair	(Bernsen et al., 1998), user study: visual thinking

Table 3.4: Guidelines for supporting the behaviors of the working user. References to the relevant literature are provided.

Chapter 4

Designing for the Listening Worker

The purpose of this chapter is to apply knowledge about how people hear to the design of an auditory display supporting the ways in which they work. The first section of this chapter reviews the physical and psychological aspects of human audition and highlights characteristics that must be accommodated in the design of an auditory display. The second section develops guidelines for producing maximally usable auditory displays based on the perceptual abilities of the listening user as applied to support the working behaviors identified in the previous chapter.

4.1 Human Audition

A requirement of designing an effective auditory display is understanding how people detect sound, perceive sound streams and their properties, and extract useful information for processing. These actions represent three loosely defined stages of human hearing. Each stage has its own set of parameters and constraints that determine what information flows to the next step and what information is ultimately available for conscious use. The next three sections are dedicated to discussing these phases. The goal of these sections is not to describe exactly *how* the auditory system works, but rather *what* consequences result from its functioning that impact the design of an auditory display.

4.1.1 Detection

The ability of the human auditory system to detect a sound is determined by the physical properties of sound waves in air and their interaction with the structures of the outer, middle, and inner ear. The following sections cover these topics in order to identify guidelines to which an auditory display must adhere to ensure a sound is at least detectable. Unless otherwise noted, the descriptions and values in this section come from Gelfand (Gelfand, 2004). For reference, Figure 4.1 gives a rough outline of the anatomy of the ear.

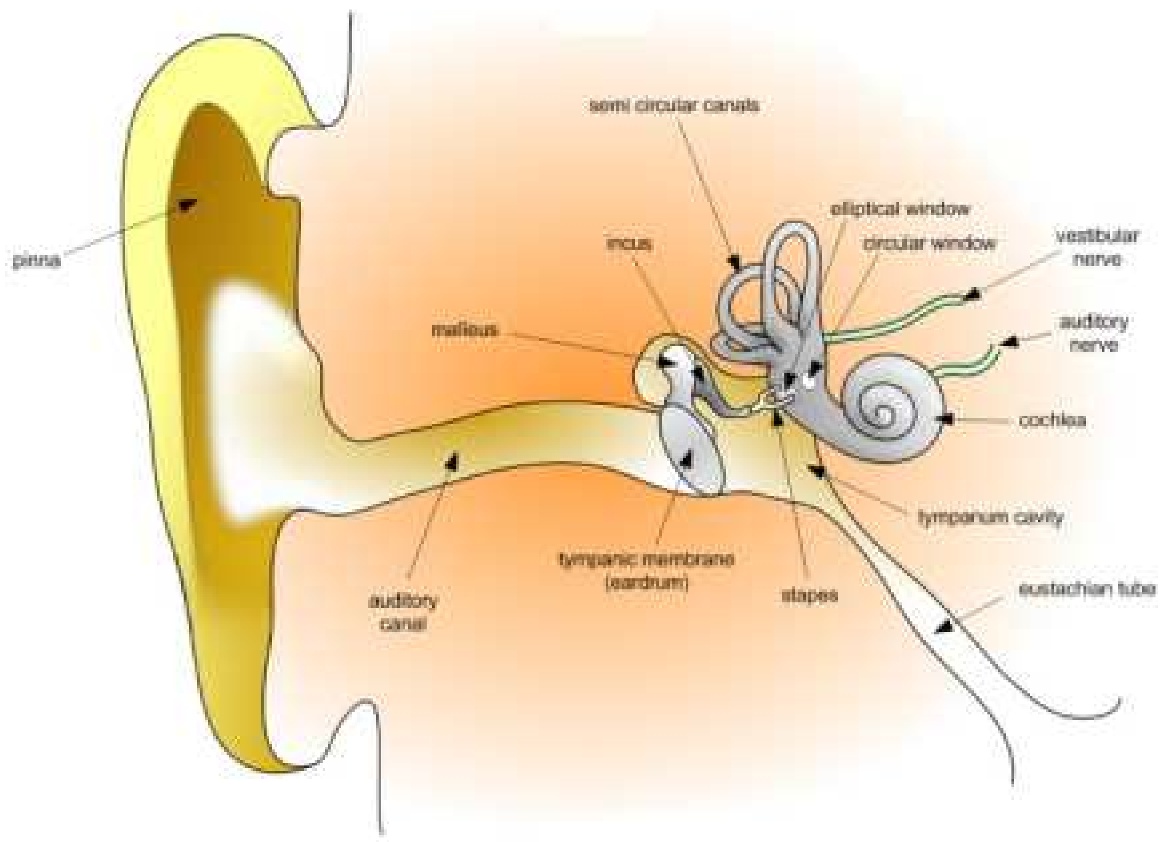


Figure 4.1: Outer, middle, and inner ear. Public domain image retrieved from `http://en.wikipedia.org/wiki/Image:HumanEar.jpg`.

Physical acoustics

In physical terms, a *sound* is a longitudinal pressure wave traveling through an elastic medium (i.e., air). A simple sinusoidal sound wave may be characterized along three basic dimensions: amplitude, frequency, and duration. The *amplitude* of a sound wave is the instantaneous magnitude of air molecule displacement caused by the wave at a single point in space. The *frequency* of a sound is rate at which fluctuations occur at that location. The *duration* of a sound is the length of time over which the pressure oscillations continue at that location. The first two of these concepts can be extended to concepts useful for characterizing more complex sounds: intensity and spectrum. *Intensity* is the power exerted by the sound wave on an area. The intensity of a sound falls off with the square of the distance from a sound source in a *free field*, or environment with no sound wave reflections. The *spectrum* of a sound describes the magnitude of its sinusoidal frequency components as determined by Fourier's theorem. Individual sound spectra can be combined into a single spectrum representing the intensities of all frequencies reaching a point.

Given just these fundamental properties, it is possible to name some rough absolute thresholds and just noticeable differences (JNDs) for sound detection based on empirical population data. The least intense sound a human listener is able to hear is 10^{-12} Watts per m^2 (0 dB IL) while the most intense sound a listener can tolerate without pain is around 1 W per m^2 (120 db IL). The JND for intensity is approximately 1 dB though both it and the thresholds are frequency dependent. The minimum frequency a listener can hear is 20 Hz and the maximum is roughly 20,000 Hz, though the recommended limits for auditory display are 80 Hz to 10 kHz (Walker and Kramer, 2004). The JND for frequency can be expressed linearly using the unit of *cents*, where $1¢ = 1200 * log_2(f_2/f_1)$. With this measure, the JND is roughly 5¢ though both the JND and thresholds for frequency are intensity dependent. The minimum amount of time a sound must persist for detection is negligible while the maximum amount of time is practically infinite.

However, the ability of a listener to determine the tonal properties of a sound requires that it continue for roughly 10 to 30 ms. Again, these times are frequency dependent (Beament, 2001). Any auditory display produce output well within these limits.

Outer Ear

The outer ear is a set of physical structures that shape the spectrum of all sound waves reaching the listener. The shoulders, head, *pinnae* or outer folds of cartiledge and skin, and ear canals all contribute to the head related transfer function (HRTF). The HRTF is the Fourier transform of the impulse response at the eardrum to a sound located at some azimuth, elevation, and distance from the head. It acts as a filter defining which frequency components are attenuated and which are amplified before reaching the tympanic membrane and passing through to the middle and inner ear. For reference, Figure 4.2 depicts the general shape of the outer ear transfer function for a sound presented directly in front of the listener.

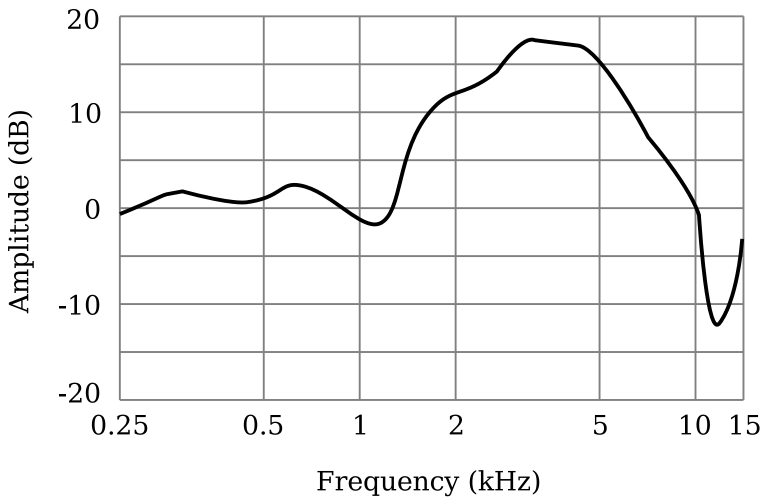


Figure 4.2: Impulse response of the head related transfer function for a source located at 0° azimuth.

The roughly spherical shape of the head acts to obstruct some sound waves and to introduce a delay between the arrival of others at the two ears. When a source is placed directly in front of or behind the head (0° or 180° azimuth), its sound waves arrive at the two ears in phase. When the source is offset so it is closer to one side of the head than the other, the sound waves incident on the ears are affected in two ways. If the wavelength of the sound is shorter than that of the diameter of the head (0.18 m or 1.9 kHz), it is blocked by the head. This *sound shadow* results in the attenuation of high frequency sounds at the far ear, creating an *interaural intensity difference* (IID). On the other hand, if the wavelength of the sound is longer than that of the diameter of the head, it diffracts and wraps around to reach the other ear. The extra distance traveled to reach the opposite ear results in a phase delay as compared with the wave received by the near ear, creating an *interaural phase difference* (IPD). Both of these effects are important cues in the localization of the azimuth of a sound source. Nevertheless, neither helps in distinguishing sounds to the front of the head from those behind: the cues are symmetric about the dorsal plane through the ears. Head movement is required to effectively determine if a sound is coming from the front or the back.

The two pinnae, what are often called the ears themselves, feature complex folds of skin and cartiledge. These folds cause numerous reflections of sound waves reaching the pinnae, resulting in constructive and destructive interference of the waves. The primary result of this interference is the creation of a large attenuation, or notch, in the HRTF impulse response at about 10 kHz. This notch shifts depending on the positioning of a sound source above the azimuthal plane, and is the primary cue for determining source elevation.

Sound waves incident on the shoulders are reflected in various directions, including upward toward the head and pinnae. The extra distance traveled by these reflected waves causes them to arrive slightly later than the primary wavefront. The slight echo is a secondary cue in the determination of sound source elevation.

The two ear canals, one per ear, are S shaped tubes leading to the middle ear. A single ear canal is open peripherally near the center of the pinna and closed medially by the tympanic membrane. Besides providing a conduit for sound to reach the middle ear and protecting the ear drum from damage, the closed tube anatomy of the canal produces standing waves at frequencies $f = nc/4L$ where c is the speed of sound, L is the length of the tube, and n is the harmonic number. For the average adult with ear canal length of 2.3 cm, sounds at approximately 3700 Hz resonate in the ear canal, intensifying waves at this frequency reaching the ear drum.

Producing sounds that appear to originate in three-dimensions outside the head requires that a computer auditory display faithfully model HRTFs. Thankfully, computer software and hardware are now capable of interpolating HRTF responses created using far-field ($>$ 1 meter) pseudo-impulse sources around a generalized head to simulate sound in 3 dimensions (Brewster et al., 2003). However, commodity products are still limited in their ability to produce believable elevation cues as well as to support the head tracking required to aid front-back localization (see Chapter 2). Therefore, an auditory display requiring no special hardware is practically constrained to placing virtual sound sources between -90° and 90° azimuth in front of the user at 0° of elevation.

Middle Ear

The primary function of the middle ear is to rectify the impedance mismatch between air and the fluids of the inner ear. Without the middle ear, all sound waves reaching the inner ear would be attenuated by roughly 20-30 dB, severely decreasing the dynamic range of human hearing. The middle ear overcomes this problem in two ways. First, the structure of the *ossicle* bones in the middle ear makes them an effective lever arm. The mechanical advantage results in a gain in force on the entrance of the inner ear over the force exerted by the sound wave on the ear drum. Second, the surface area of the tympanic membrane driving the ossicular chain is an order of magnitude greater than that of the bony plate pushing on the inner ear. This areal relationship results in an additional gain.

A secondary consequence of the anatomy of the middle ear is that it acts as a high-pass filter. The ossicular chain can be modeled as a spring-mass system with friction. The mass of the system is trivial considering that the ossicles are the tiniest bones in the body and the friction is negligible since the ossicular chain is suspended in air by only a few ligaments and tendons. The remaining component of the system, springiness or conversely stiffness, dominates the definition of the middle ear transfer function. The stiffness component allows high frequency sound waves to pass most easily while attenuating low frequency sounds. Figure 4.3 depicts the general shape of the transfer function of the middle ear.

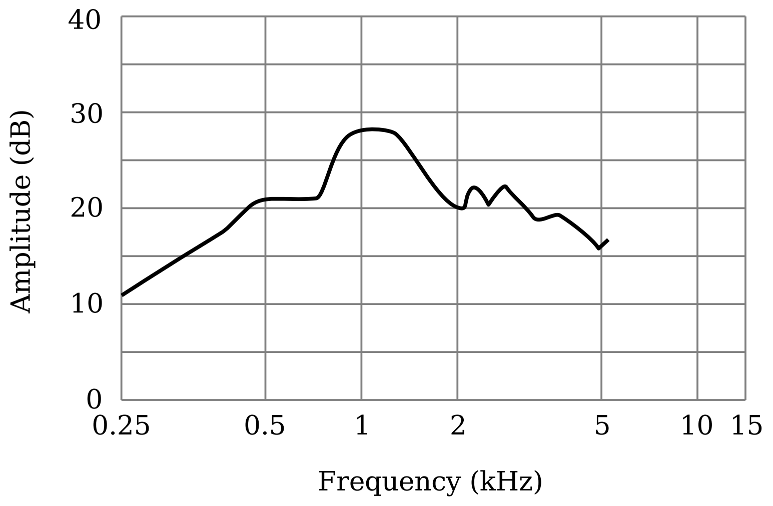


Figure 4.3: Middle ear transfer function with input at the tympanic membrane and output at the oval window of the cochlea.

The stiffness of the middle ear is not constant and is directly affected by the *acoustic reflex*. Shortly after an intense sound is first heard, the tendons attached to the ossicular chain tighten to protect the delicate structures of the inner ear from damage. The resulting increase in stiffness of the system further attenuates the intensity of low frequency sound waves. The intensity required to trigger the reflex varies with the center frequency of a noise band and the bandwidth of the sound, roughly 90 dB IL as a rule of thumb. The intensity, bandwidth, and frequency also have a proportional effect on the latency between the onset of the sound and the reflex action.

An auditory display must avoid relying on low intensity, low frequency sounds to convey information as they are likely to be partially filtered by the middle ear. A display must also avoid producing sudden, intense sounds that trigger the acoustic reflex and inadvertently attenuate the low frequency components of other sounds.

Inner Ear

The inner ear, primarily the *cochlea*, is responsible for the transduction of mechanical sound wave vibrations into electrical signals in the central nervous system. Figure 4.4 depicts a cross section of this spiral organ, including its primary internal structures. Figure 4.1 shows the cochlea as a whole from an external vantage.

The *stapes* ossicle of the middle ear pushes on the *oval window* at one end of the of the cochlea creating pressure differences within the cochlear fluids. The pressure differences create traveling waves on the *basilar membrane* that peak at a location particular to the frequency of the original sound waves: high frequencies near the base, low frequencies near the apex. Tiny *stereocilia* on the tops of *hair cells* seated on the basilar membrane bend against the *tectoral membrane* near the front of the traveling wave. The lateral bending of the stereocillia triggers an increase in the release of neurotransmitters at the base of the hair cells. The released neurotransmitters trigger responses in *afferent* nerve cells, those connected to the hair cells and sending signals to the brain.

The outer three rows of hair cells running up the length of the cochlea are *motile*, able to shrink and elongate spontaneously and independently. When the stereocilia atop these cells are bent laterally by the traveling waves, the cells actively push against the tectoral membrane. This added force on the tectoral membrane serves to increase its motion and thus the response of the highly innervated, single row of inner hair cells. The effect of this active process results in the "sharpening" of the neural response at points along the basilar membrane coinciding with the most intense frequencies in the spectrum of the original sound.

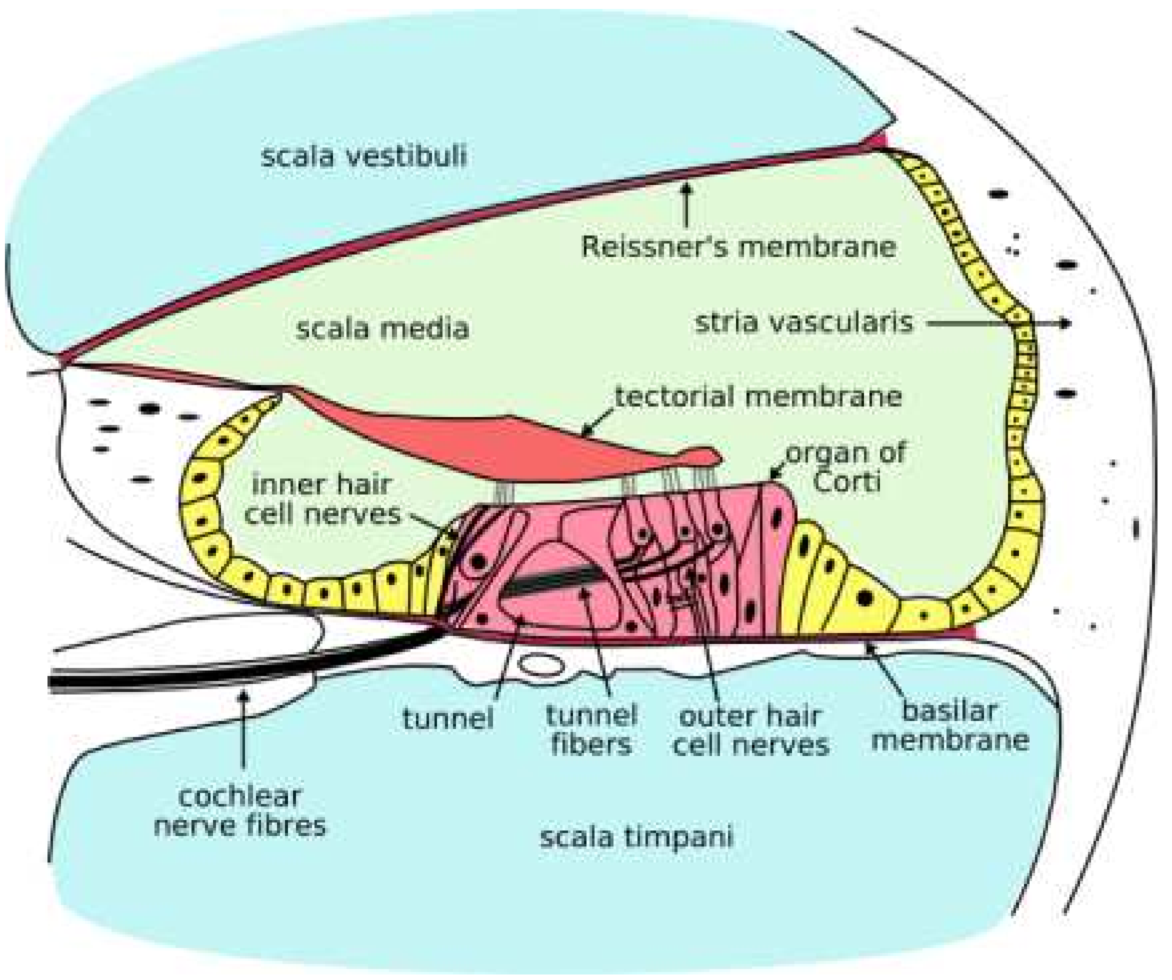


Figure 4.4: Cross section of the cochlea. Image retrieved from `http://en.wikipedia.org/wiki/Image:Cochlea-crosssection.png` and licensed under the terms of the GNU Free Documentation License `http://www.gnu.org/copyleft/fdl.html`.

The primary benefit of the sound sharpening caused by the inner ear is an improvement in the frequency resolution of human hearing. However, this active process makes the inner ear inherently non-linear and results in a number of side effects. One outcome is that of *two-tone suppression*, or the tendency of an intense sound to inhibit nerve fiber response to a less intense sound near to the first in frequency. For instance, if a pure tone at 400 Hz is played at 30 dB IL simultaneously with a pure tone at 450 Hz and 60 dB, the nerve fiber response to the 400 Hz tone is inhibited by nearby fibers responding to the 450 Hz tone. Depending on the magnitude of the inhibition, the listener might not detect the 400 Hz tone at all.

Another result of the active cochlear process is that of *distortion products*. *Aural harmonics* are produced at integer multiples of an intense stimulus having only one fundamental frequency (e.g., $2f_1$, $3f_1$, $4f_1$). *Combination tones* result from the interaction of the fundamentals and harmonics of two simultaneous sounds. *Summation* and *difference* tones with frequencies $f_2 + f_1$ and $f_2 - f_1$ result when two simple sounds are presented at intensities well above the threshold for hearing. *Cubic difference tones* occur at frequency frequencies $2f_1 - f_2$ and are generated when two stimulus sounds are presented at moderate intensity levels. Interestingly, combination tones can interact with one another and the original stimulus tones to produce *secondary combination tones*, implying that the tone stimuli are physically present in the cochlea.

An auditory display that relies on the presentation of multiple simultaneous sounds must separate intense sounds from less intense sounds in frequency in order to avoid suppressing the quieter of the two. The display must not rely solely on pure tones with similar frequencies to convey information because the multiple tones are likely to be perceived as one tone with intensity varying over time. Also, the display should not rely on the ability of the listener to count the exact number of sounds heard or judge the absolute frequencies as "ghost" frequencies might be introduced by the cochlea. Overall, though, the extraordinary dynamic range and resolution of the cochlea gives an audio display a large space in which to work.

Auditory filters

More abstractly, the cochlea can be viewed as a series of overlapping bandpass filters through which sounds of various frequencies are detected and encoded as neural signals. The bandwidth of these *auditory filters*, or *critical bands*, varies with frequency and ranges from roughly 90 Hz wide below 200 Hz to 900 Hz wide around 5000 Hz. Mappings of the filters reveal that they have longer high frequency tails than low frequency tails and that the high frequency tail falls off slower for filters centered around low frequencies.

This filter abstraction provides a useful framework for explaining a number of auditory phenomenon. If two simple tones close in frequency are played simultaneously, their points of maximum excitation on the basilar membrane will fall into the same critical band. As a result, the two tones will actually be perceived as one *beating* tone with intensity varying on a cycle equal to $f_2 - f_1$.

Auditory filters are also important in understanding *peripheral masking*: when one sound (the masker) diminishes the ability of the listener to detect another sound (the target) presented at the same time (*simultaneous masking*), immediately before (*backward masking*), or immediately after (*forward masking*).

Simultaneous masking occurs when spectral content from the masker falls within the same auditory filter(s) as the target such that the intensity threshold for detecting the target is elevated. For instance, band-limited white noise centered around a target tone will mask the tone even if the noise intensity is less than that of the tone. Low frequency sounds are also capable of masking remote high frequency sounds due to the long tails of auditory filters covering higher frequencies.

Forward and backward masking occur when a masker, presented either before or after a target tone, has spectral content that falls into the same auditory filter as the target. Forward masking may be the result of the listener hearing the target as a continuation of the masker, up to roughly 150 ms after the masker ceases. Backward masking is not as well understood and has an effect on targets up to 50 ms before the masker starts.

An auditory display must mitigate masking by ensuring that wide-bandwidth and low frequency sounds do not become so intense that they mask other, simultaneously presented high frequencies. Similarly, the the timing of sounds in the display must be such that a wide-bandwidth sound does not mask another sound immediately preceding or following it.

4.1.2 Perception

Once a sound has been detected and encoded as a neural signal, the auditory pathways in the brain analyze the sensory data to produce the experience of sound. The following sections describe these parameters as well as how the brain tends to create discrete sound objects from the neural spectrum encoded by the ears.

Parameters of psychoacoustics

Loudness is the perceptual equivalent of intensity, but is not linearly related to it. In fact, loudness is both frequency and duration dependent. Sounds in the 1 kHz to 5 kHz range are louder than those outside this frequency band when played in isolation in a free field. Moreover, a tenfold increase in sound intensity within an auditor filter is perceived as only a doubling in sound loudness. In other words, ten sources producing sound at approximately the same frequency or with similar spectra will only sound twice as loud as one source alone. Loudness is also duration dependent in that sounds lasting less than 1 second are experienced as being louder than sounds lasting more than 1 second (Gaver, 1997).

The perception of *pitch* varies logarithmically with frequency. This relationship can be expressed linearly again using the units of cents. One hundred cents is the equivalent of a *semitone*, the interval between two musical notes such as C and C# or E and F. Twelve hundred cents is the equivalent of a musical *octave*, the interval between two harmonics of a fundamental. Overall, the human auditory system can distinguish approximately 1500 separate pitches thanks to the sharpening mechanism of the cochlea (Nave, 2006). Pitch is also dependent on sound loudness. Sounds above 2 kHz appear to rise in pitch when their intensity is increased while those below 2 kHz seem to drop with an increase in intensity (Nave, 2006).

The *position* of a sound relates it to the location of its source in space. As mentioned earlier, the structures of the outer ear play a fundamental role in shaping the sound

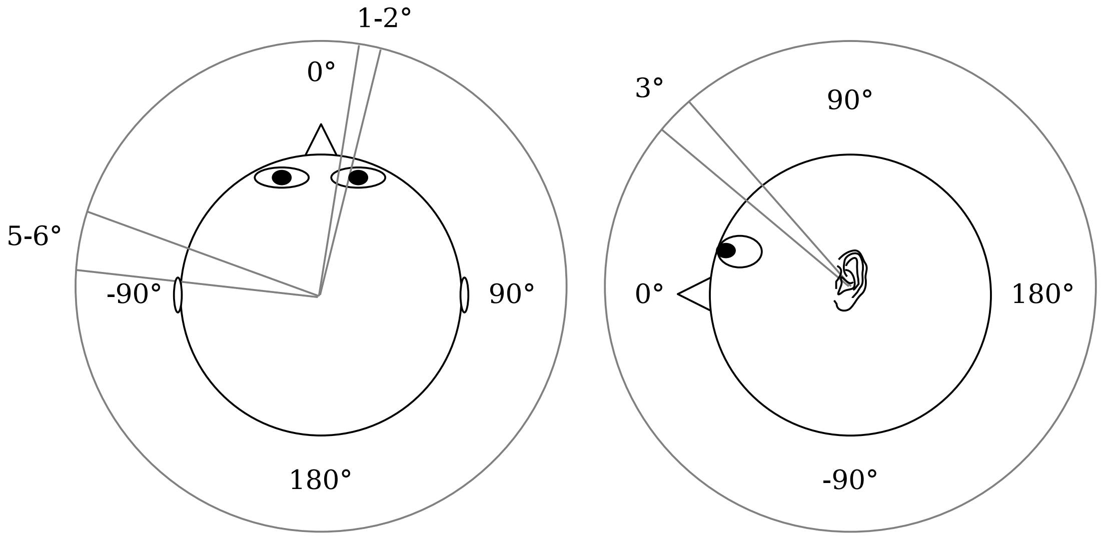


Figure 4.5: Resolution and axis for sound localization in azimuth (left) and elevation (right).

spectrum such that later processing by the brain can determine source azimuth on the medial plane, its elevation above that plane, and its distance from the head (see Figure 4.5). The resolution for localization in azimuth is about 1-2° of arc in the direction of the nose (0° azimuth) and 5-6° on each side of the head (±90° azimuth) for pure tones (Walker and Kramer, 2004). Discrimination of sounds in front of the listener from those behind is poor without head movement as the phase and intensity differences are symmetric about the dorsal plane (Gorny, 2000). An estimate of elevation resolution is not as readily available as that for azimuth, but the results from at least one recent study suggest 3° of arc over the range -40° to 90° (slightly below to directly above the head) for sounds having some frequency content between 6 kHz and 10 kHz (Susnik et al., 2005). The distance to a sound source is determined by its intensity, the attenuation of its high frequencies due to damping caused by air, and reverberations in the listening environment. In all three dimensions, localization error is reduced when using broadband sounds instead of single frequency tones (Neuhoff, 2004).

Timbre is perception of the spectrum of a complex sound that allows a listener to distinguish two sounds having the same pitch and loudness (Gaver, 1997). The timbre of a sound is largely determined by its harmonic content and its dynamic properties over time such as periodic fluctuations in pitch (*vibrato*), fluctuations in loudness (*tremolo*), and the suddenness of its onset and offset (*attack* and *decay*). A sound must persist for at least 60 ms for its timbre to be fully determined, in contrast to the shorter 10-30 ms requirement for discrimination of its pitch. This extended duration requirement can be reconciled with the theory that timbre is processed in higher levels of the brain than other properties. These areas need more samples over time in order to identify long-term spectral and dynamic properties (Nave, 2006).

The *rhythm* of a sound describes its pattern of onsets and terminations over time while its *tempo* describes the rate at which the pattern repeats. Like timbre, tempo and rhythm are concepts requiring integration at higher levels of the brain since they are inherently measures of changes over long periods of time. Based on the perceptual processor cycle time of 70 ms given by the Model Human Processor (Card, 1983), detecting individual beats in tempos up to 840 beats per minute should be possible.

Harmony is the vertical dimension of music characterized by the use of simultaneous pitches called *chords* which are perceived as single entities. *Melody* is the horizontal dimension of music defined by series of sounds varying in pitch, loudness, timbre, and duration grouped as a succession of conceptual entities. *Consonance* is produced by a chord that is stable, harmonized, and often described as pleasing to hear while *dissonance* is formed by a chord that is unstable, anharmonic, and usually said to be unpleasing to the ear. The feeling produced by consonant and dissonant sounds are dependent on the culture as well as the style of the music. In most music, however, harmonic dissonance is introduced to create tension in the piece and later resolved to a consonance as a means of release. A dissonance left unresolved has a tendency to leave a listener expecting the consonant release while a piece consisting only of consonant harmonies is often heard as boring (Burns, 1999).

Auditory Scene Analysis

The neural signal produced by the cochlea represents the entire frequency spectrum of sound detected by the ear. This spectrum might, for instance, include components from many sound sources such as a cell phone ringing, a radio playing, a car horn honking, and the wind rushing by as a listener drives his car. Yet, quite amazingly, a listener is still able to distinguish the voice of his passenger from all of these other sounds. How the auditory system accomplishes this feat is described by the theory of auditory scene analysis developed by Alfred Bregman (Bregman, 1990). Understanding the predictions made by this theory is a key aspect in designing an auditory display that can present more than one channel of information simultaneously.

Auditory scene analysis is the process by which perceptual *streams* are formed. A stream is a perceptual grouping of the parts of the encoded spectrum that "go together" (Bregman, 1990). In streaming, portions of the spectrum are integrated into discrete perceptual objects based on their likeness in terms of psychoacoustic parameters. Conversely, portions of the spectrum dissimilar in terms of the psychoacoustic parameters are segregated into separate streams. The streams that result from this analysis are an approximate representation of the discrete sound sources and events that produced the spectrum encoded by the cochlea.

At least three basic tenets underly this theory. First, Gestalt principles govern the parsing process. Similarity, proximity, continuity, and common trajectory determine what constitutes a stream (Ueda et al., 2005; Bregman, 1990). Second, portions of the available spectrum tend to be allocated exclusively to streams. Information from one sound source is rarely a contributor to more than one perceptual object (Bregman, 1990). Third, multiple streams may result from the segregation process, but only one stream, part of a stream, or grouping of streams can be attended to consciously at a time. Attention may select only one object as figure while the rest become background (Valkenburg and Kubovy, 2004; Barber et al., 2003; Duncan et al., 1997).

Primitive processing

Conceptually, the scene analysis process takes place in two stages. The first stage, primitive analysis, is an innate, bottom-up process whereby streams are produced according to correlations of spectral and temporal cues (Huron, 1991). Primitive processing occurs pre-attentively and its resulting streams are held in a temporary auditory image store (Nager et al., 2003). The streams in this store are those that can be later selected for attentive processing (Valkenburg and Kubovy, 2004). The diagram shown in Figure 4.6 depicts auditory scene analysis up through this phase.

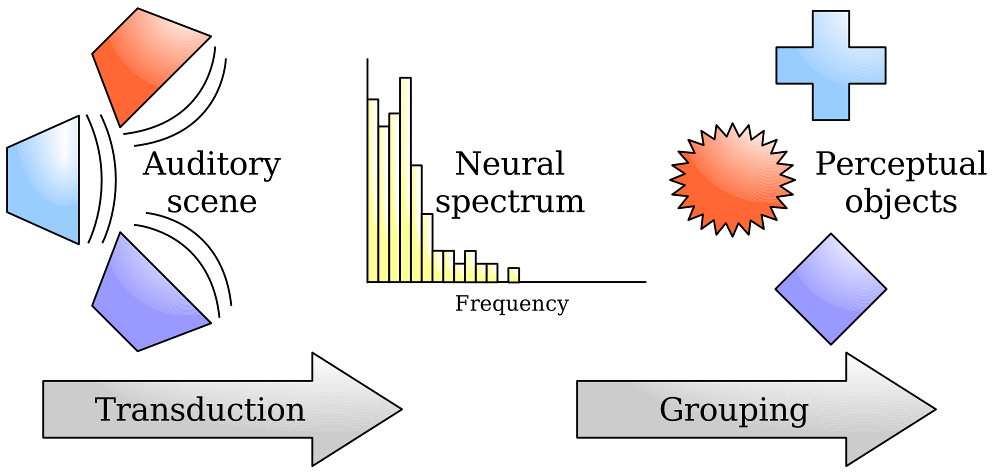


Figure 4.6: Sounds from the environment are detected by the ears and encoded as a neural spectrum. Primitive auditory scene analysis forms perceptual streams by integrating similar spectral cues in frequency and over time. Based on a diagram from (Valkenburg and Kubovy, 2004).

Primitive analysis occurs along two dimensions: frequency and time. At any given point in time, the primitive process segregates the spectrum encoded by the cochlea into streams representing simultaneous, yet distinct sound sources. It is this operation that allows a listener to identify the sound of the ringing cell phone and the car radio as two, distinct sources. Over time, portions of the spectrum are integrated into streams representing coherent, persistent sources. This second action is what allows a listener to perceive, for instance, speech in terms of words and phrases instead of a collection of disjoint guttural, throaty, and clicking sounds.

The factors known to influence the segregation of sound into streams are listed in in Table 4.1 (Arons, 1992; Bregman, 1990). Varying the parameters in the first column over time increases the chances of sound from a single source splitting into more than one perceptual stream. Making concurrent sound sources dissimilar in terms of the parameters in the second column decreases the likelihood that the independent sources will be perceived as one. The occurrence of sudden changes in these parameters or extended periods of silence ($>$ 1 second) cause the immediate recalculation of streams (Cusack et al., 2004).

Over time	*In frequency*
Frequency	Frequency
Pitch	Onset/offset synchrony
Timbre	Regularity of spectral spacing
Center frequency (noise)	Binaural frequency matches
Amplitude	Harmonic relations
Location	Parallel amplitude modulation
Shortening of gaps	Parallel gliding of components

Table 4.1: Factors affecting primitive auditory scene analysis. The ordering in the table is arbitrary as the relative ordering and interactions of the parameters is not well established at present. (Bregman, 1990)

Presenting multiple channels of information simultaneously in an auditory display requires a design that accounts for these innate rules. A multi-channel display should separate channels in space, pitch, timbre, loudness, starting time, and trajectory to assist their segregation into disparate, non-conflicting streams. Moreover, the same display should keep a given channel fixed in space, pitch, timbre, loudness, and at a high tempo to ensure the channel stream remains fused. To force a reset of stream calculation, a display need only make a sudden change to its parameters.

4.1.3 Conception

After sounds have been segregated into streams and placed in the auditory image store, conscious attention selects a stream for further processing using knowledge stored in long-term memory. The next section details this process including constraints on stream selection, how selected streams are associated with learned knowledge, and how attention can affect the primitive process.

Schema-based Processing

The second phase of auditory scene analysis is a learned, top-down process in which one primitive stream in the auditory image store is selected for conscious inspection. This process occurs attentively and uses working memory to hold information about a stream of interest. Streams pulled into working memory serve as both retrieval cues for *schemas*, representations of knowledge stored in long-term memory, and as cues for storing new schemas (Card, 1983). Retrieved schemas influence the listener's conception of the attended sound: a stream of tones may be recognized as the melody of known tune, a stream of phonemes may be heard as an intelligible sentence, and so on. Schema-based processing may also influence primitive processing such that streams are formed to more readily match a recently retrieved template (Bey and McAdams, 2002) while attention to a particular stream provides maintenance of its segregation from other perceptual objects (Cusack et al., 2004). The diagram shown in Figure 4.7 depicts this second phase.

As mentioned previously, a tenet of the theory of auditory scene analysis is that only one stream formed by the primitive process may be chosen for attentive, schema-based processing at a time. While the primitive system *segregates* the sound spectrum into

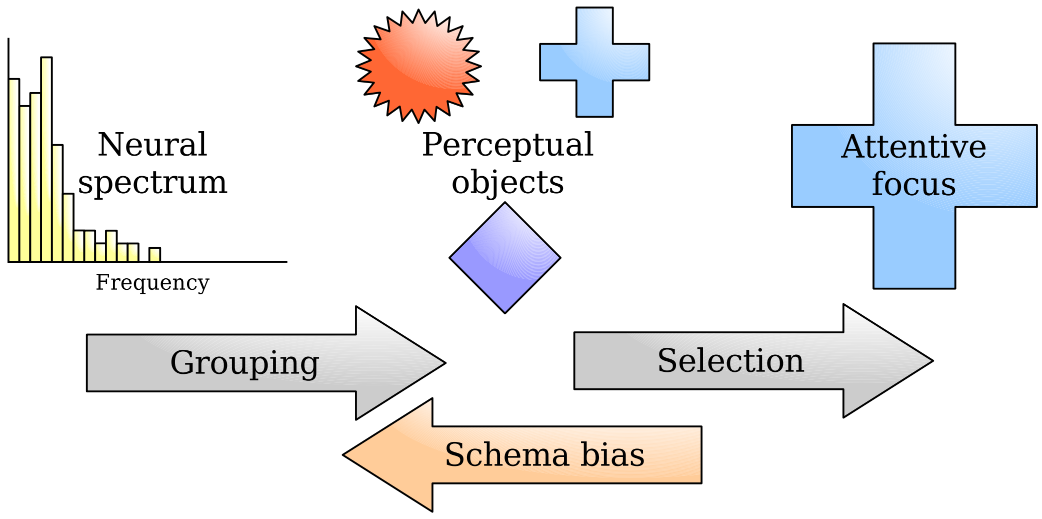


Figure 4.7: Sound streams produced by the primitive segregation and grouping processing are available for attentive processing. One stream is selected at a time for processing and the act of selecting can bias the primitive process. Based on a diagram from (Valkenburg and Kubovy, 2004).

multiple perceptual objects, the attentive system may only *select* a particular object for further processing. Thus, a listener may hear music on the car radio and his passenger talking at the same time, but may pay attention to and process only one stream at a time. The listener cannot consciously attend to and understand both streams at once. Any apparent ability to do so likely derives from a rapid switching of attention among primitive streams held in the auditory store, not from true parallel processing (Barber et al., 2003). Once selected, the stream becomes the sole figure of auditory attention while all the other streams become background (Valkenburg and Kubovy, 2004). Still, there is evidence that background speech streams may have a priming effect, an increase in speed or accuracy of information detection, on conscious listening (Rivenez et al., 2004).

The efficiency with which listener may select a single stream for attention depends on the similarity of that stream relative to other streams in the auditory buffer (Bey and McAdams, 2002; Arons, 1992; Cherry, 1953). If primitive segregation produces two streams that share many of the same characteristics listed in Table 4.1, directing conscious attention to and fixating on one of the two streams will be difficult. Moreover, if the two streams share similar semantic content matching the same learned schema, processing of one of the two streams will be difficult in the presence of the other (Arons, 1992). For instance, if the driver tunes the car radio to a news station with an announcer that sounds strikingly similar to his chatty passenger, the driver will have trouble focusing attention solely on the passenger's voice. The ability of the driver to understand what his passenger is saying will be further inhibited if both the radio announcer and the passenger are talking about the news. What the newscaster says about national politics will confuse and become mixed with what the passenger might be saying about local politics (Wickens and Hollands, 2000).

Streams segregated by primitive analysis may be regrouped by attention (Bey and McAdams, 2002; Bregman, 1990). Conscious listening can cause separate streams to fuse into one in order to match an active template. Again, the efficiency with which streams may be fused in schema-based analysis is dependent on the similarity of the streams to one another and the degree to which they match the target template. For example, say some notes in familiar melody are shifted an octave higher. When the melody is presented in this manner, a listener will likely hear two sound streams—one for the lower notes and one for the higher notes. If informed of the name of the melody, however, the listener will be able, with some effort, to fuse the two streams into one fitting the learned schema for that melody. Nevertheless, if pitch shifted portions of the melody are made even more dissimilar to the rest of the notes, for instance, by transposing them by a number of octaves or presenting them in a different timbre, integration of the two separate streams by conscious effort will become more difficult.

In the same vein, a part of a fused stream may be "listened out" by the attentive process such that it is pulled into its own stream. This slicing operation is not the equivalent of primitive segregation, though. The listener is aware of almost none of the information left behind after the selection (Bregman, 1990).

A sudden change in a stream not only resets the streaming mechanism, but can also pull conscious attention to it (Ueda et al., 2005). For instance, if the car radio tuner suddenly breaks and starts putting out static instead of music, both the driver and passenger will likely pay immediate attention to the radio stream, regardless of whether or not their attention was previously directed elsewhere. The same behavior holds true for the sudden onset of a new auditory stream. If a police car suddenly turns on its siren nearby, both vehicle occupants will start attending to the new sound and processing its meaning. In both cases, the ability of a stream to involuntarily drive listener attention to it is a function of its similarity to other streams, the magnitude of the change, and the effort the listener has invested in listening to another stream.

The resources allocated for attentive processing of sound streams are independent of those used for primitive analysis. Investing effort in understanding a particular stream does not inhibit the streaming process from segregating the remaining sound content at the same time (Kline and Glinert, 1994; Wickens, 1991). Similarly, the resources used for processing of auditory information are separate from those applied to other modalities. A listener is capable of identifying the presence of separate auditory and visual targets when they overlap in time whereas identification of two targets is more difficult when they are presented in the same modality (Duncan et al., 1997). There also appears to be separate resource banks for verbal versus spatial processing as well as for input processing versus response preparation (Wickens, 1991).

An auditory display must be consistent in how it presents information in order to facilitate the formation and retrieval of sound schemas from long-term memory. The presentation of multiple, concurrent streams of information is possible and enables the switching of listener attention among them. Ensuring the distinction of streams in terms of the parameters discussed in the previous section improves the efficiency with which the listener may select a particular stream for conscious processing. Since the listener is limited to attending to one stream at a time within a modality, however, a display must not place critical content in separate, current streams. On the other hand, the display need not account for listener attention to and responses in other modalities as different modalities, stages of processing, and type of processing draw on different resources.

4.1.4 Summary of Audition

A gross summary of human audition, as it relates to the design of an auditory display providing multiple concurrent streams of information, may be encapsulated in the following points.

1. Ensure all sounds are detectable. All auditory information should fall well within the absolute limits of hearing.

2. Recognize the existence of suppression, distortion products, and masking. Avoid using pure tones, sounds close in frequency and time, and intense broadband sounds.

3. Account for the mapping between physical acoustics and psychoacoustics. Adjust physical source parameters so its sound is heard at the desired psychoacoustic level and pitch.

4. Place independent, simultaneously presented content into maximally distinct streams. Make streams distinct in terms of space, pitch, loudness, timbre, starting time, and trajectory.

5. Support fusion of a given stream of information over time. Keep sound position, pitch, loudness, and timbre fixed over time.

6. Require attention to only one stream at a time. Never place critical information in two concurrent streams without an option for reviewing each alone.

7. Facilitate attention switches among streams. Make streams distinct in terms of their psychoacoustic parameters as well as their semantic content.

8. Pull attention toward important streams but avoid grabbing attention accidentally. Create sudden changes in streams to drive attention to them. Avoid sudden changes to prevent involuntary attention switching.

9. Provide consistency in presentation. Facilitate the learning and recall of sound to concept mappings by reinforcing the mapping throughout user interaction.

4.2 Guidelines to Support the Listening Worker

It is now possible to state guidelines that maximize the effectiveness of an auditory display for office computing with respect to its utility and usability. The former quality is defined as the ability of the display to provide functionality for completing user tasks (i.e., can it do what the user needs?). Ensuring the auditory display supports the behaviors reviewed in the previous chapter guarantees adequate display utility. The latter quality is defined in terms of the ease with which a user can learn to use the display, use it efficiently, remember how to use it, avoid or correct errors, and remain satisfied using it (Nielsen, 1993). Designing support for information seeking, text editing, and mediation around the abilities of the user as a listener lays a foundation for producing a usable auditory display.

4.2.1 Views as Streams

The concept of an AgileView (Marchionini et al., 2000; Geisler, 2003) maps conveniently to a set of perceptual auditory streams describing some aspect of an information space. All streams within the set act in concert to create the view while all other streams outside the set serve to complement it. For instance, a single continuous sound stream might convey an overview of the user's active task. On the other hand, two primary speech streams might work together to report the characters and words the user is inputting into the active task respectively. Previews, reviews, and peripheral views can manifest in a similar manner.

Providing multiple views of an information space is only one of two requirements in the AgileViews framework. The other states that the user interface must support rapid navigation among the views (Table 3.4-14). In the auditory domain, the fastest possible navigation requires that the user simply switch his or her attention among perceptual streams without giving additional input to the computer. An auditory display can achieve this attention-driven navigation by presenting multiple views as concurrent auditory streams. For instance, if the primary view consists of a stream of speech while the overview is a looping sound, a user navigates between these two views by first listening to one stream and then the other. Such a multi-stream configuration may not increase information throughput under the assumptions of the theory of auditory scene analysis (i.e., single stream attention). But, if the display presents streams in a manner such that the listener can segregate and select them easily, it can obtain optimal times for navigation.

Nevertheless, a concurrent stream display cannot represent all of the AgileViews at once and remain understandable. A listener can separate roughly three concurrent streams into independent entities, even when they are designed for maximum segregation (Nager et al., 2003). A listener can identify and draw meaning from one non-speech sound among up to five sounds with information encoded in their acoustic properties (McGookin and Brewster, 2004a; Lorho et al., 2001; Gorny, 2000). Speech, however, is not so easily selected and decoded in the presence of other speech streams. A listener can focus on and understand one stream of speech among approximately three total speakers before acoustic and information masking becomes too great (Ueda et al., 2005).

These guidelines suggest that an auditory display must selectively serialize and preempt streams for views.

Various factors affect the decision as to which streams are concurrent, which are serialized, and which are available on user request. The display could ensure that streams for views used most frequently are audible as often as possible to support rapid navigation by attention switching. The design could allow streams carrying information about the current user task take precedence over those that refer to peripheral content. If possible, the display could serialize streams reporting conflicting content. To some degree, the user could also have control over which streams are active based on his or her current needs.

In a view-stream structure, the working user will attend to the streams for the primary view most often. The display should allow these streams to report information to the user any time he or she requests it. The user will require frequent hints about his or her working context to overcome the ephemeral nature of audio and his or her limited short-term memory. The display might support overview streams that play continuous or intermittent sounds as reminders. The user will selectively choose content to explore in detail based on its relevance to the current task. The display might provide preview streams summarizing content before it reaches the primary view to aid this decision making process. These three views (primary, overview, and preview) are candidates for concurrency because of the user's need to access them quickly and frequently.

The remaining two views, peripheral and review, are primary candidates for serialization and on-demand playback. The user will attend to information about inactive tasks only if it catches his or her interest. The display must ensure peripheral streams report secondary information in a timely fashion before it becomes stale, but also in a manner that avoids distracting or confusing a user attending to the other, more important views. The user will request reviews of past content as a means of overcoming masking and confusion among concurrent streams. Furthermore, the very nature of a review stream as a history of information previously presented in the other streams precludes it from a concurrent existence with those other streams.

Concurrent Views

Supporting concurrent stream-views requires designing for fusion within streams, segregation across streams, selection of individual streams, and establishment of relationships among streams forming a view. To this end, the acoustic properties of concurrent streams in separate views must differ along as many dimensions as possible. Starting playback in various streams at different times aids segregation. Spacing streams geometrically on the azimuthal plane serves as a major cue for stream separation (Brungart and Simpson, 2003). Alternating the gender of neighboring speech streams and assigning a single unique voice to each stream helps the distinction. Using different timbres, rhythms, pitches, and melodic/harmonic structures prompts separation of musical sound streams. Choosing sounds with drastically different frequency content disassociates natural, iconic sound streams. Even if the display cannot modify the acoustic properties of a sound stream to support segregation, (i.e., the properties encode information), spatialization alone serves as an effective cue, at least for distinguishing sounds (McGookin and Brewster, 2004b).

Properties of a given stream must remain fixed to ensure fusion of the stream over time. Since sound frequency and amplitude within a stream must vary to convey information (e.g., phonemes in a spoken sentence), the display can only hold parameters such as position, timbre, voice gender, pitch baseline, and average loudness constant. Keeping streams at fixed locations, in particular, helps a user form mappings from spatial positions to types of reports (Shinn-Cunningham et al., 2004; Baldis, 2001; Kobayashi and Schmandt, 1997). Similarly, co-locating streams that work together to create a view establishes their relationship. For instance, if concurrent speech and sounds describing inactive tasks always originates from a spatial position to the far left of the user, he or she will learn to focus on that location for such peripheral information.

Although a user can quickly direct internal attention to a certain stream, some external mechanism for directing input to a particular view must also exist. Assume, for example, that the display supports a review of the last words spoken. If the display has multiple concurrent speech streams, it must provide a means for the user to indicate which speech stream he or she wishes to replay (Table 3.4-13).

Primary view

The primary view is most often the focus of user attention as it provides details about the current information of interest. To highlight its prominence, the display might position its streams to the front of the user and make them slightly louder than the streams in the other views. The primary view should rely on speech to convey its detailed content, to spell words to overcome pronunciation problems, and to disambiguate *homophones*, words spelled differently that sound the same.

To indicate special properties of text, the view could support one of two solutions. On one hand, the view could introduce a second, non-speech sound stream and use it to play sounds concurrently with its speech. For example, to indicate that a portion of a sentence is also a hyperlink, the view might play a *whooshing* sound while it speaks the hyperlink text (Goose and Möller, 1999; Asakawa and Itoh, 1998). With this solution, the view must ensure all sounds are quieter than the primary speech stream and played with a baseline pitch very different than the primary voice to avoid masking.

Alternatively, the primary view could rely on changes to voice parameters to convey properties of text. For instance, the display could indicate the same hyperlink by shifting the baseline pitch while speaking the words in the link. This second solution is not ideal, however, since the user could confuse changes in vocal properties with fluctuations due to *prosody*, natural changes in speech. Moreover, changing speech stream properties could lead to undesirable stream segregation over time. That is, the user might perceive the pitch shifted words in the hyperlink as an entirely separate stream of speech, not part of the sentence as a whole.

Both non-speech sounds and changes in vocal properties are insufficient for indicating some kinds of information. A fast typist, for example, may wish to hear both the letters entered and the words they form without having the former continuously interrupting the latter. The primary view might employ a second speech stream to report this additional information. The view could position this second stream close to the primary speech stream to indicate their relationship, but must offset it enough to support segregation and rapid attention switching between the streams (e.g., $\pm 10°$ azimuth where spatial resolution is high). The second speech stream might use a voice that differs greatly in gender, timbre, and baseline pitch from the first speech stream to further aid segregation and selection. The stream could also speak at a significantly lower volume to indicate its lesser importance and avoid masking the more important information in the first, primary view speech stream.

Overview

To provide constant reminders to the user about his or her current context (Table 3.4-11), the display must make the overview omnipresent. Such an overview might consist of one or more sounds representing one or more of the tasks on the active focus space stack (see Figure 3.1). The sounds could be continuous, but faint, to create a background ambiance which the user can interpret as a representation of his or her "location" in the task space. Furthermore, the mere presence of such a sound environment might aid users in retaining information and recalling it at a later time, even if the ambiance is absent (Tan et al., 2001).

All continuous overview sounds must be free of sudden changes in pitch and loudness. The user might mistake these fluctuations as carrying non-existent information.

Similarly, task sounds must be void of noticeable patterns that might become tedious and annoying to the user over extended periods of time. For these reasons, the overview should prefer the use of natural, ecological sounds to synthetic, structured, musical sounds or speech.

To create an ambiance, the display might make the overview stream omnidirectional, free of particular spatial location. Such a source would encompass the listener, just like sound in most real world environments. For example, the overview might immerse a user in a sound environment like a harbor with waves sloshing and gulls cawing, or a forest with leaves rustling and bugs buzzing, to represent the active working context.

The overview must provide a clear demarcation when the user navigates among tasks to indicate the transition. Simply changing the continuous, environmental sound may not work well, even if periods of silence are believed to reset the primitive process. A user who has learned to "tune out" the ambiance may not notice transitions to and from silence. To account for this potential problem, the overview could play some sound drastically different from the slowly changing, continuous background to indicate the activation of another task. The overview might sound a reasonably loud earcon at such a time to provide information about the type of task change (e.g., starting versus ending a task), pull user attention to the newly activated task, and force a recalculation of streams by the primitive process. For example, suppose the overview plays the ambiance of a newspaper office while the user composes an email. If the user decides to switch to his or her Web browser, the overview might sound a brief earcon indicating navigation to another task. After that sound completes, the overview might begin playing a different ambiance, perhaps the sound of a freeway, representing the Web browsing task.

If the user is interacting with the display in a situation where sounds of the real-world environment is critical to the user's work or safety, the system should allow the user to control the overview streams. Muting, lowering the volume of, or changing the selected display ambiance might all serve to avoid conflicts with important information from the external environment.

Preview

The preview should provide succinct summaries of content in or with the potential to enter the primary view (Table 3.4-10). To make plain this relationship between views, the display might locate the preview streams close to the primary view, but far enough away so that the user can listen to them independently (e.g., ±10 to ±30° azimuth where spatial resolution is high). From this distance, preview sounds are unlikely to conflict with spoken reports from the primary view, if they are held to a relatively low volume. On the other hand, the preview must take care when using speech to avoid masking output from the primary view streams, regardless of the relative volume.

To report nominal or ordinal information, the preview might prefer the use of non-speech sounds to speech. Such sounds can quickly convey information without conflicting with other speech streams. For example, the preview could summarize the length of an email using three earcons with different tempos meaning the email body is short, medium, or long. Regardless of the sounds used, the preview must play them at a volume softer than the one used by the primary view for sounds.

The ability of a listener to make absolute, or even relative, judgments of sound properties is limited (Miller, 1956). Therefore, the preview should report interval or absolute information using synthesized speech. The preview could announce the number of words in an email, for example, as a more exact indicator of its length. Such a spoken preview should have a lower volume than and start asynchronously from speech streams in the primary view. The voice used by the preview must also be dissimilar from the ones speaking in the primary view.

Both spoken and non-speech previews are unlikely to conflict with sounds played by the overview streams. Their acoustics are quite different from the slowly varying environmental sounds. In addition, previews play before or after a task change, not during a transition when the loud overview demarcation sound is present.

Peripheral View

The peripheral view should alert the user to events happening outside the primary view. This view must inform the user about the indirect consequences of his or her actions on inactive tasks (Table 3.4-5). In addition, the peripheral view must report events occurring in tasks beyond the primary view in case the user is interested in them (Table 3.4-7). For instance, say the user initiates a download in a Web browser and then switches to composing a text document. The peripheral view might announce the completion of the download in case the user is waiting for this information before continuing some inactive task.

The position of the peripheral view streams should reflect their namesake and exist in the spatial periphery (e.g., ±60°to ±90° azimuth). Locating the views to the sides of the listener emphasizes their lesser importance when compared with the views located ahead of the user. Distancing the peripheral streams from the other views also helps avoid accidental fusion when they play concurrently. Finally, the peripheral view might separate streams reporting on different segments of the information space on either side of the head. For instance, a stream to the left of the user can report on tasks related to the active task while a stream to the right announces unrelated events.

The peripheral view should tailor its announcement of events around the activities of the primary view, overview, and preview. If the primary view and preview are speaking, the peripheral view might wait for one or both to finish before speaking or playing a sound. If just the primary view is speaking, the peripheral view might play a concurrent sound to indicate an event occurred. If the primary and preview streams are silent, the peripheral view could safely speak a detailed message. Under any condition, the peripheral streams should speak or play a sound at a volume less than that of the other streams.

To keep announcements from the peripheral view relevant, the display should place an upper bound on the time the peripheral view is silent while the others are active. Once this time has elapsed, the stream might play a brief sound to request the user's attention, regardless of whether other streams are still playing.

Review

A review must repeat information presented in the past by other views. The transience of auditory information forces a listener to either hold important content in working memory or encode it for long term use (Lai and Yankelovich, 2003). The limited permanence and capacity of working memory causes the loss of information if the user does not refresh the information in the memory buffer or file it away in long-term memory (Ware, 2000; Sweller et al., 1998). The review should support this refresh by allowing the user to navigate through past speech and sounds in the primary, preview, and peripheral views in temporal order. For instance, the review could report the last chunk of content spoken by the primary view and the last summary given by the preview on user request (Table 3.4-9).

Retrieving information from long-term memory requires effective working memory cues (Card, 1983). The review should supply these cues by presenting exact replicas of past sound events. Say, for example, a peripheral view plays a sound and speaks to notify the user that a Web page finished loading. At a much later time, a user reviewing the peripheral stream hears the same sound and speech coming from the same position. Such an exact reproduction should aid user recall of the original occurrence of the event and any associated long-term memories. The review is not limited to reporting only past information, however. It can include additional details about past events after replicating the original announcements (Table 3.4-12).

In addition to supporting the automatic recording of streams, the display might give the user the ability to flag particular segments of text so that they may be easily recalled at a later time. For instance, assume a user listening to an email message encounters a username, password, and filename he or she will need later to complete a task. The display could provide an option for remembering these items, so that the user might quickly request them when reaching the Web site. Again, the review should make the replay identical to the original to assist user recall of the original information and any associated context (Table 3.4-8).

The review could report past information on user demand, after preempting all output from the primary, preview, and peripheral streams. This design accounts for the possibility that the user requests a review to resolve masking or confusion among concurrent streams. For instance, when the primary and preview streams both speak simultaneously, and the user has trouble disambiguating their content, an interruption followed by a serialized review allows the user to hear one report and then the other without interference.

4.2.2 Tasks as Stream Content

Mapping tasks to streams is a conceivable alternative to a view-stream design. The display could render every task as an audible stream with acoustic properties distinguishing it from all other tasks. The problem with such as design is that the possible number of running tasks is unbounded. Ensuring an unlimited number of streams are uniquely identifiable by their positions, loudness, pitch, timbre, and other acoustic characteristics is difficult, if not impossible. Even if an auditory display could provide the necessary distinction of streams, users would still have a hard time identifying a particular task-stream from its potentially numerous counterparts based on relative judgments of its properties (Miller, 1956).

Tasks are better considered content objects within an information space, accessible through a finite number of view-streams. In such a design, the primary view and preview could act in concert to give both detailed and summary information about the active task. The overview could provide persistent reminders about the identity of the active task. The peripheral view could report on the state of the remaining, inactive tasks. The benefit here is that the number of potential streams is fixed by the number of views while the user is free to access an unlimited number of tasks (Table 3.4-18).

The display must support navigation among the various running tasks. Commands for starting, resuming, and ending tasks could provide such navigation. When a task starts, it might immediately become the active task, the focus of the primary view, and the target of all user input (Table 3.4-19). When an inactive task resumes, the currently active task might move to the periphery while the inactive task enters the primary view. When a task ends, the display could remove it from all views and then resume the most recently visited task. Meanwhile, the overview might encode the execution of these commands using a demarcation sound separating tasks. For example, the overview could use earcons with timbre encoding the kind of task (e.g., entering information, reading a document) and melody giving the type of change (e.g., start, stop, resume) (Brewster et al., 1993).

The display must also provide a method of directing start and resume commands at a particular task. One possibility is to send all resume task commands to the last task to make an announcement. For example, if a Web browser announces that a download is complete, the next resume task command might activate the Web browser. Unfortunately, this design does not account for the case where two streams give output simultaneously for two different tasks. Continuing the same example, assume an email program announces an incoming message at the same time as the download finished report. In this case, it is unclear whether the next user resume command should activate the email client or Web browser. A slight improvement requires that the user select which concurrent stream is the target of the command before issuing it. However, the moving target problem remains: another event can occur in the stream before the user reacts to the one of interest. Enabling the use of the resume command during a review solves the timing problem, but this mechanism limits the user to picking tasks that already exist and have made recent announcements.

Presenting the user with a list of selectable named tasks mitigates all of these problems. The display could consider the act of selecting a task a main menu that the user can activate using a special command (Table 3.4-4). When the menu is active, the display might treat it like any other task in the primary, preview, and overview streams. The primary view could announce the names of the available tasks. The preview could give a summary of each task. The overview could play a continuous sound unique to the menu, and easily recognizable among all other overview sounds. For example, a musical, rhythmic sound can fit this purpose and contrast with the non-repeating, environmental sounds for other tasks. Because the user is likely to browse such a fixed menu for brief periods of time, a rhythmic, looping sound is not likely to become too distracting.

The cost of using an explicit task menu for navigation tasks is that it introduces a modal element into the display. The user must leave his or her active task to visit the menu. The design of the display must weigh the ability of this solution to provide a single, consistent, time-independent method of starting and resuming tasks against the size of the interruption it causes in the user's workflow. Still, an explicit, modal menu is likely to be more usable than forcing the listener to synchronize his or her commands with output from one of many streams.

Browsing Actions

The primary view should take responsibility for reporting text content and properties as the user browses information within a task (Table 3.4-1). Navigation by character could result in a spoken announcement of each character, and, depending on the verbosity preferences of the user, whole words as they are encountered. Navigation by words might produce spoken reports of each word in the text. Navigation by chunk could result in a report based on the task-dependent definition of a chunk. For instance, in the task of browsing emails in a mailbox, the display might reasonably define a single message as a chunk. In this case, a user might hear the sender, subject, and other header info announced in the primary view (Table 3.4-12).

The preview should give the user an idea of where he or she is browsing in the current task, and how much content remains unexplored. For instance, a preview of an email folder could report the index of the currently selected message and the total number of messages. Such information helps a user decide on the most efficient way to go about completing a task. If, for example, a mailbox contains five unread messages, the user might prefer to read them all. On the other hand, if the mailbox has five hundred new messages, the user might wish to first prioritize or filter them before traversing them.

The peripheral view should connect the content the user is browsing in the active task to relevant content. For instance, the peripheral view could summarize where a hyperlink leads when the user passes over it on a Web page. Likewise, as the user browses appointments on a calendar, the peripheral view might mention other appointments on the same day.

Searching Actions

The primary view should also take responsibility for supporting searches within a task (Table 3.4-2). The primary view could give immediate feedback when the user begins defining search terms. One primary view stream might announce the first word or chunk matching the search query every time the query is updated. A second primary view stream could confirm the terms the user has already entered. At the same time, the preview could summarize the position of the current match within the entire body content. The searching functions could respect commands for navigating forward and backward through search matches, and provide the same level of feedback after each movement. Together, these features should provide a means for filtering content so that it may be traversed by points of interest (Table 3.4-6).

Switching between searching and browsing must be transparent to the user (Table 3.4-3). At any point during browsing, the display should allow the user to initiate a search by defining search terms. Likewise, when a search reveals a match of interest, the display must allow the user to start browsing content at that location. The auditory display should make this transition seamless such that output during browsing and searching is similar, if not exactly the same. For instance, the primary view could report the name and details of a file in the same manner regardless of whether the user reached the file by browsing or searching. The preview, however, might make a verbose announcement when searching and no report at all when browsing. Searching for a file could cause the user to skip hundreds or thousands of files that do not match the search criteria. The display should inform the user of such a large jump. In contrast, browsing from one file to the next is a relatively small change. The display should not overwhelm the user with announcements about such trivial navigation.

Editing Actions

Finally, the primary view should announce changes during text editing. When entering or deleting text, the first speech stream could report words while the second should echo characters. The view streams could play sounds when the user encounters text with special states as well as when special editing events occur. An auditory icon, for example, might represent text acting as a heading in a document, a misspelled word, or emphasized text. Likewise, an earcon might succinctly indicate when the user splits an existing line in two with a line break, or joins two lines into one by removing a break.

Spoken feedback echoing user input assumes that the speech echo is not in conflict with the input method. For example, the echoing of words and characters is appropriate when input is given using the keyboard. To the contrary, when user input is spoken, shadowing everything the user says with synthesized speech is unnecessary and likely confusing. When input is spoken, auditory icons and earcons might provide effective feedback, for example, about voice recognition confidence.

4.2.3 General Behavior

In general, the display should adhere to protocols of communication such that interaction with the user is efficient and satisfying. The display should ground the interaction by confirming user actions with audible feedback and requesting more information when commands are vague or incomplete. Views should prefer short sounds as repetitive acknowledgments of user actions instead of more time consuming speech confirmations. Views might also manipulate sound parameters such as consonance to convey additional information such as whether a user action completed successful or not (Hankinson and Edwards, 1999). On the other hand, views should use speech when the user requires a more specific report of the last user action or request for further information made. The detail of the feedback given should help a user set his or her grounding criteria for the interaction. For instance, giving a large amount of spoken feedback on each user command can help the user realize that the interaction is not proceeding as smoothly as possible (Table 3.4-16).

Using ecological sounds should help a display honor the maxims of quality and manner (Table 3.4-20). Everyday listening occurs in environments with rich sound spectra, not pure tones. Mimicking real world sound sources should improve user acceptance of a display over less natural alternatives. Mixing a variety of musical, environmental, and speech sounds might also hold user interest and reduce frustration, annoyance, and fatigue over extended periods of time (Mereu and Kazman, 1997).

The decision whether to use speech or non-speech sounds relates closely to the maxims of quantity and appropriate level of detail (Table 3.4-17). When the user requires an exacting description, the display should rely on speech. The price for using speech is that it takes longer to output and is prone to content confusion caused by other concurrent speech streams. When the display must report information quickly or unobtrusively, it should use non-speech sounds. The difficulty with using sound comes from knowing how to effectively map information to the various available acoustic properties. However, some general guidelines exist for choosing how to map certain classes of information to sound.

Representing Information in Sound

When expressing boolean information, the display should use auditory icons. For instance, the primary view can play a crashing sound while reading a sentence to indicate a misspelling. The absence of the crashing sound indicates no misspelling exists. The downside to such natural sounds is that relationships between real world events and computing concepts do not always exist (Mynatt, 1997; Carroll et al., 1997). The user must take time to learn arbitrary mappings. However, this requirement is not unique to audio: the meaning of many graphical indicators must also be learned. The red underline indicating a word is mispelled is just one example.

The display should represent nominal data using earcons. Short, musical motives related by timbre, melody, rhythm, and octave can name items in a set. For example, earcon rhythm can distinguish task start, resumption, completion, and cancellation. Similarly, timbre can identify the type of task in question (Brewster et al., 1993). The drawback to using earcons is that the user must learn how their properties name concepts. In the context of the previous example, the mapping from timbre to task type is arbitrary and requires that the user discover and memorize this association.

The display should convey ordinal relationships using relative differences in basic acoustic properties of sounds. Encoding concepts such as more/less, longer/shorter, etc. in the pitch, tempo, or duration of two sounds is possible when the user needs only a rough comparison (Alty and Rigas, 1998). Presenting two sounds, for example, with the pitch of each sound mapped to the length two text documents might enable a rapid determine of which document is shorter and longer. The display must support configuration of the polarity of the mapping, though, because sighted and visually impaired users tend to expect opposing relationships (Walker and Lane, 2001).

Using non-speech sounds to convey interval and absolute data is difficult. Untrained listeners cannot easily make judgements about the distance between two sound parameters or assign an exact value to one sound parameter in isolation (Edwards, 1988; Miller, 1956). For data in these categories, a display should speak a short summary of the information whenever speech synthesis is available.

4.3 Summary

This chapter developed the groundwork for supporting office computing tasks via an auditory display. The guidelines were based both on the abilities of the user as a listener as well as the behaviors of the user as a worker. The following chapter describes an implementation of a software auditory display built from these specifications and targeted at office productivity applications.

Chapter 5

The Clique Auditory Display

This chapter describes the synthesis of the Clique software auditory display from the concepts developed in the previous three chapters. The chapter starts by approaching the design of Clique from the perspective of the user. The first few sections describe the auditory scene the user hears, how streams in that scene convey information about the user's work, and how the user issues commands to manipulate that scene, and thus advance his or her work. A section follows explaining how the user experience satisfies the user requirements stated earlier in this work.

The middle of this chapter explains Clique in terms of the software architecture that enables the desired user experience. The corresponding sections describe the model-view-controller (MVC) paradigm used to separate application business logic from user interface, the input/output pipeline concept employed to organize output from concurrent tasks, and the scripting framework designed to support adaptation of existing programs to the Clique auditory display as well as the creation of new audio applications. A section closing this discussion summarizes how the architecture resolves problems that arise when attempting to develop usable auditory displays.

The chapter concludes with a brief description of a prototype implementation of Clique in the Python programming language. The last sections explain the function of seven Clique scripts which adapt existing GUI applications for use in Clique and one which implements a simple application from scratch with no GUI counterpart. These applications serve as the basis for user evaluation in the next chapter.

5.1 User Experience

Clique divides applications into *tasks* which the user may start, pause, resume, complete, and cancel at almost any time. Each task consists of one or more *subtasks* that allow the user to browse, search, or edit content. In some cases, a task or subtask is equivalent to a window or widget in a visual display. More often than not, however, Clique tasks and subtasks support interactions that would span multiple windows and widgets if implemented graphically.

Clique positions virtual assistants around the user in a sound space. The assistants take responsibility for reporting content in the active subtask, summarizing the active subtask, indicating changes in related and unrelated tasks, and echoing important user input. The assistants use synthesized speech to read detailed information to the user and musical and natural sounds to indicate important events and task states. At times, multiple assistants speak or play sounds concurrently to give the user faster access to information that would otherwise appear later in a serial stream. Meanwhile, environmental and looping sounds play around the listener as reminders of the active *program*, one or more processes launched by a Clique script, and *task*, an arbitrary unit of work done in a program. Acting together, the assistants and environment form one implementation of the view-stream auditory display structure recommended at the end of Chapter 4.

The user issues commands to browse, search, edit, remember, and query content as well as to mediate interaction with the display. Commands such as *List my programs*, *List my tasks*, and *Start recording* are available at all times. These commands form a set of persistent services the user may invoke in the same manner regardless of what

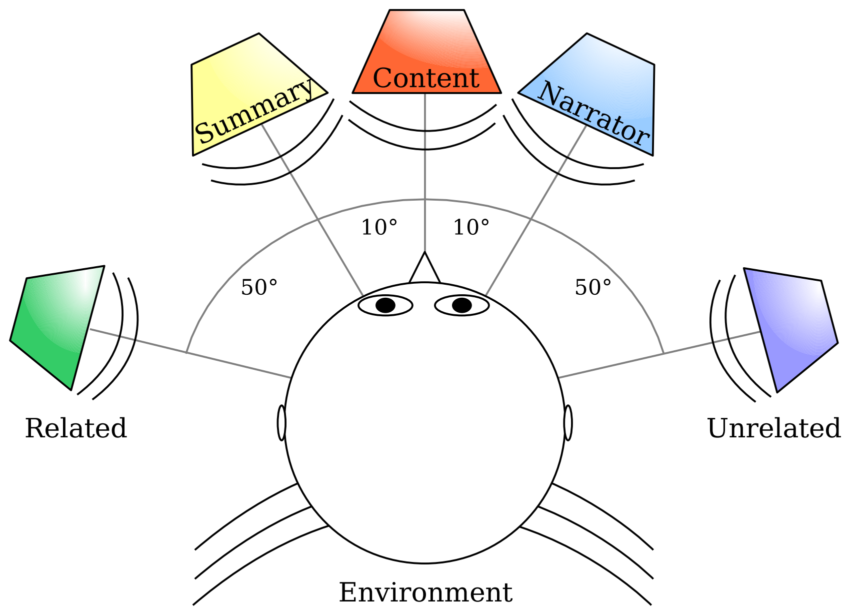


Figure 5.1: Spatial layout of the Clique virtual assistants.

program and task are active. The availability and meaning of other commands such as *Go to next item* and *Go to first item* depend on the active task and subtask. Still, the manner in which the user invokes these commands is not context sensitive.

5.1.1 Virtual Assistants

The virtual assistants in Clique describe the information space of available programs, tasks, subtasks, and their content. Clique places these virtual assistants around the user, separating them spatially and acoustically to maximize segregation and selection when they speak or play sounds concurrently. Table 5.1 states the names, responsibilities, and vocal properties of each assistant. Figure 5.1 depicts their spatial relationships to the user.

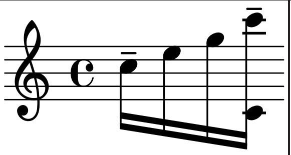
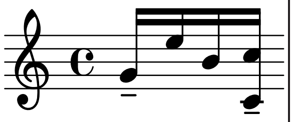
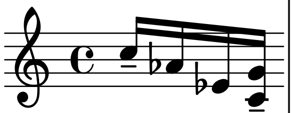
Names	*Responsibilities*	*Vocal Properties*
Content	Reports content to the user (e.g., text in an email, the names of files, appointments on a calendar). Indicates special properties, or *states*, of content reported (e.g., misspelled words, emphasized text). Part of the primary view.	American male voice, 100% speech volume, 60% sound volume, 0° azimuth
Summary	Reports orientation information to the user (e.g., index of the current email out of the total number of emails, time spent reading a document out of the total time it would take to read the document). Indicates the type, or *role*, of the current task (e.g., browsing a list, editing text). Part of the preview.	American female voice, 86% speech volume, 60% sound volume, -10° azimuth
Narrator	Confirms user actions by echoing important input (e.g., characters typed when editing and searching). Part of the primary view.	British female voice, 78% speech volume, 60% sound volume, 10° azimuth
Related	Summarizes changes in tasks and subtasks related to the active subtask when either changes (e.g., number of messages available after selecting a new mailbox, Web browser download completion notice while browsing a Web page). Part of the preview and peripheral view.	Deep American male voice, 70% speech volume, 55% sound volume, -60° azimuth
Unrelated	Reports summaries of tasks and subtasks unrelated to the active subtask when the state of the unrelated component changes (e.g., Web browser download complete notice while writing an email). Part of the peripheral view.	Hoarse American male voice, 70% speech volume, 55% sound volume, 60° azimuth
Context	Reports navigation among tasks when a task starts, completes, cancels, or resumes. Part of the overview.	No speech, 55% sound volume, not spatialized (i.e., inside the user's head).

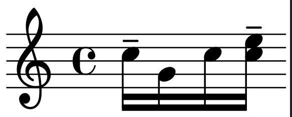
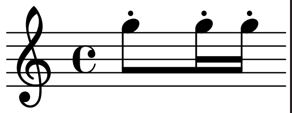
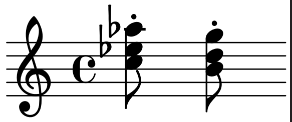
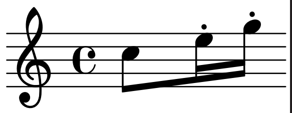
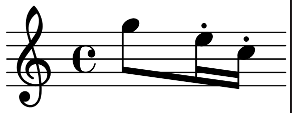
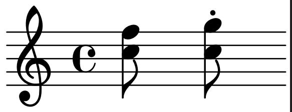
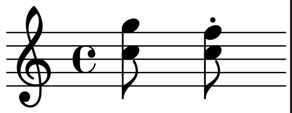
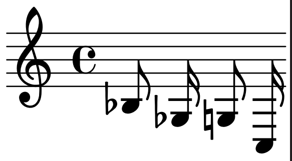
Table 5.1: Virtual assistant names, their responsibilities, and vocal properties distinguishing their speech streams. Volume percentages are given to indicate relative levels, but their unit intervals are sound hardware dependent.

Earcons

The assistants use earcons to represent *actions* taken by the user or the system. The action earcons use timbre, rhythm, pitch, consonance, and duration to convey information about events. The instrumental timbre of the earcon indicates the role of the task in which the action occurred. The rhythm names the type of the action itself. The sequence of pitches (i.e., melody) in the earcon distinguishes actions within a class of related actions. The consonance or dissonance of the earcon, according to Western Music tonality, conveys the success or failure, the positive or negative consequence, of an action. The density of notes in the earcon represents the complexity and importance of the action. The duration of all earcons is greater than 200 ms to ensure fusion of the sound stream, but not much longer than a second to keep the display responsive. The tempo of all action earcons is rapid, nearly 200 beats per minute. Chapter 2 categorizes this set of action sounds as a familial, or hierarchical, earcon grammar.

Table 5.2 names the actions indicated by earcons in Clique, shows the earcon melody on a treble clef staff, and describes their use, acoustic properties, and relationships.

Action	*Staff*	*Description*
Task started		A new task loaded and is ready for user interaction. Timbre represents the role of the task. Rhythm is the same as task completed, cancelled, and resumed.
Task completed		The user completed an existing task successfully. Timbre represents the role of the task. Rhythm is the same as task started, cancelled, and resumed.
Task cancelled		The user cancelled an existing task without completing it. Timbre represents the role of the task. Rhythm is the same as task started, completed, and resumed.

Action	*Staff*	*Description*
Task resumed		The user resumed an existing task by making it the active task. Timbre represents the role of the task. Rhythm is the same as task started, completed, and cancelled.
Wrapped		The user jumped from the last item in a collection to the first or vice versa. The collection can be a set of subtasks or a set of content items. Timbre indicates the role of the task containing a set of subtasks. Timbre is always a flute when wrapping within a subtask.
Deleted		The user deleted an item in a collection subtask. Timbre is a flute.
Moved to next line		The user browsed to the next line in an editable text document. Timbre is a flute. Rhythm is the same as moved to previous line.
Moved to previous line		The user browsed to the previous line in an editable text document. Timbre is a flute. Rhythm is the same as moved to next line.
Joined		The user joined two words or lines by deleting a space in an editable text document. Timbre is a flute. Rhythm is the same as splitting two lines or words. Harmony plays when joining lines, but not words. Rhythm is the same as split.
Split		The user split two words or lines by inserting a space in an editable text document. Timbre is a flute. Rhythm is the same as joining two lines. Harmony plays when splitting lines, but not words. Rhythm is the same as joined.
Refused		The system refused to take an action in its present state. Timbre is a baritone saxophone.

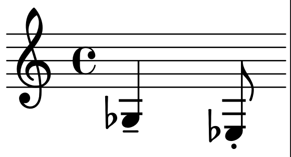
Action	*Staff*	*Description*
Warned		The system warned the user about the consequences of taking an action. Timbre is a baritone saxophone.

Table 5.2: Clique action earcons.

The assistants also use earcons to represent stateful properties of subtasks or their content. The state earcons are all percussive to distinguish them from action earcons. Beyond this simple classification, the state earcons share no other relationship with one another. According to Chapter 2, this state earcon grammar is representational.

Table 5.3 names the states indicated by earcons in Clique and describes its use, acoustic properties, and relationships to other earcons.

State	*Properties*
Missing	A subtask is currently unavailable. Timbre is a melodic drum.
Misspelled	A word is misspelled. Timbre is a tambourine.
Navigable	An item in a collection contains additional items. Timbre is a drum.
Beginning	The reading position is at the beginning of a body of text. Timbre is a timpani drum.
End	The reading position is at the end of a body of text. Timbre is a timpani drum.
Recording	The system is recording or no longer recording speech output. Timbre is guitar fret noise.
Searching	The system is accepting or no longer accepting user input for a keyword search. Timbre is a sandblock.

Table 5.3: Clique state earcons.

Auditory Icons

The assistants use auditory icons to identify the roles of subtasks and content in Clique. The role of a subtask informs the user of what interactions it supports. Therefore, the mapping between the auditory icons used and the subtasks represented hints at the possible actions in the current context (e.g., commands for browsing a list). Similarly, the role of a piece of content identifies its purpose in a body of content. In this case, the mapping distinguishes a segment of content from other content (e.g., a heading in a document).

Because many computer interactions cannot be directly related to the sounds of everyday, physical events, Clique uses symbolic and metaphorical auditory icons as defined in Chapter 2. The duration of each auditory icon is at least 200 ms to ensure fusion of the sound stream, but less than one second to keep up with fast paced interaction.

Table 5.4 names the subtask roles represented by auditory icons in Clique and describes the sounds themselves. The definition of each role and their differences is left for Section 5.3.3 of this chapter.

Role	*Sound*
List	Fast rippling through a deck of cards
Tree	Wind blowing tree leaves and branches
Text	Papers shuffling
Table	Plates and silverware clattering
Hyperlink	Zapping
Heading text	Marching footsteps
Editable text	Typing on a keyboard

Table 5.4: Clique role auditory icons.

5.1.2 Environment

Clique situates the virtual assistants in a sound environment where one or more continuous, omnidirectional streams are always active. These streams are intentionally quieter than the assistant voices and sounds so that they sit at the edge of user attention. When the user needs a reminder of the active program or active task, however, paying attention to the background sounds can help him or her quickly identify both pieces of information.

The first time the user activates a program, Clique pairs it with the recorded ambiance of a real-world environment. Thereafter, any time a user is working in that program, even after Clique restarts, the paired ambiance plays continuously in the background of the Clique auditory scene. As the user navigates between programs, the background sound fades from one ambiance to another. All ambient sounds are of long duration and have few high frequency, high intensity variations. These characteristics avoid pulling user attention to the background sound unintentionally.

Clique also assigns each task in a program a background sound. The task sounds are taken from sources commonly found in the environment representing the program. For instance, if the program ambiance is that of a farm, Clique will assign the program tasks short sounds such as hens clucking, horses whinnying, dogs barking, and so forth. The sound assigned to the active task repeats indefinitely, but with periods of silence between each repetition. That is, task sounds play intermittently, not continuously, to avoid overwhelming the user or blending imperceptibly into the program ambiance. Clique randomly selects each break duration from a range of three to ten seconds to avoid producing predictable rhythms in playback that might draw user attention inadvertently.

Together, an ambiance and its related intermittent sounds form a *environmental sound theme*. Clique attempts to assign each program a unique theme, and each live task in that program a unique intermittent sound. The purpose of this assignment is to assist the user in quickly identifying the active program and task. Since the themes are pre-recorded waveforms, however, the number of active programs may outnumber the number of available themes. Similarly, the number of running tasks in a program may be greater than the number of intermittent sounds in a theme. In such situations, Clique reuses sounds while attempting to minimize duplication for all active programs and tasks.

For example, assume nine task sounds are available in a theme and nine tasks are live in the program using that theme. When the user starts a tenth task, Clique will reuse one of the nine sounds. Now assume the user cancels two of the ten tasks leaving eight tasks alive. The next time the user starts a new task, Clique will assign it the ninth unused sound.

Table 5.5 shows the current list of ambient and intermittent sounds available for pairing with programs and tasks. For the purposes of the evaluations described in Chapter 6, this set of sounds is large enough to ensure unique identification of all programs and tasks. In less constrained contexts, users could add or customize themes by placing sound files in specific directories on disk.

Environment	*Sound Event*
Airplane	Baby crying Cabin announcement Passenger yawning Landing gear
Farm	Chicken clucking Cow mooing Horse whinnying Rooster crowing Sheep bleating
Harbor	Dolphin squeaking Foghorn blowing Gulls cawing Outboard motor running Waves lapping
Rain	Footsteps sloshing Falling splash Thunder rumbling Wind howling

Table 5.5: Clique environmental sound themes.

Clique includes three *system menus* accessible from any program or task. These menus are not part of the programs themselves, but are services provided by Clique to support navigation of tasks and content. To uniquely identify these special menus, Clique assigns them three drum beats which play continuously whenever they are active.

Table 5.6 names the three menus and describes their associated looping drum beats. The following sections describe the purpose of these menus in greater detail.

Menu	*Properties*
Program Menu	Steady, syncopated hip-hop beat with snare and woodblock
Task Menu	Fast, jungle beat with a lead congo drum
Memory Menu	Slow, marching beat driven by bass drum triplets

Table 5.6: Clique menu drum beats.

5.1.3 Program and Task Navigation

The user navigates among programs and tasks using two menus of the system. When Clique starts, the *program menu* greets the user by starting its unique drum loop. At the same time, the content voice announces the first program name and the summary voice says the number of available programs. Only programs adapted for using Clique appear in this list. The user may browse and search this menu, and then select a program to activate. The user may return to the program menu at any time to select another program to activate using the *List my programs* command.

When the user activates a program, its selected ambiance fades in. If the user is visiting this program for the first time, Clique activates its *task menu*. The unique task menu drum loop starts playing, while the content voice announces the name of the first available task and the summary voice states the initial number of available tasks. The user may browse and search the list of tasks in the same manner as the program menu, and then select a task to activate. The user may return to the task menu at any time when a program is active using the *List my tasks* command.

Unlike the program menu, the choices in the task menu are mutable and depend on the context in which the user activates it. For example, when the user initially activates the email program, its task menu lists options for browsing existing emails and writing new email. If the user activates the task menu again while reading an email, the task menu contains additional tasks such as replying to the message, forwarding the message, and moving the message to another mailbox. If the user switches to the calendar program, its task menu lists a wholly different set of choices for browsing the calendar, adding appointments, and so on.

The task menu orders its items according to a design that supports learning and fast repetition. Global tasks that are available in every context appear at the beginning of the list. A user can always rely on a given set of input commands to consistently activate one of these options. Tasks that the user may start in the current context appear next. Within that context, the user can again rely on a given set of input commands to consistently activate one of these options. Finally, any inactive tasks appear at the end of the list. Since the task menu, and all Clique lists, wrap from beginning to end and end to beginning, the user can quickly reach other running tasks using the *Go to previous item* command starting from the first menu item.

When the user is finished working on a task, he or she may choose to complete using the *Complete this* command, cancel it with the *Cancel this* command, or simply navigate away from it, leaving it inactive. If the user navigates away from a program while a task is running and then returns to the that program later, the last active task in that program resumes immediately.

As for programs, the user may only activate or deactivate them. There is no concept of opening or closing a program. Clique manages whether a program is loaded or not based on what tasks the user has left running.

5.1.4 Subtask Navigation

The user activates a subtask explicitly by navigating to it using the *Go to previous subtask* and *Go to next subtask* commands, or implicitly by starting, resuming, or cancelling a task. In all cases, Clique announces the active subtask according to the template depicted in Figure 5.2.

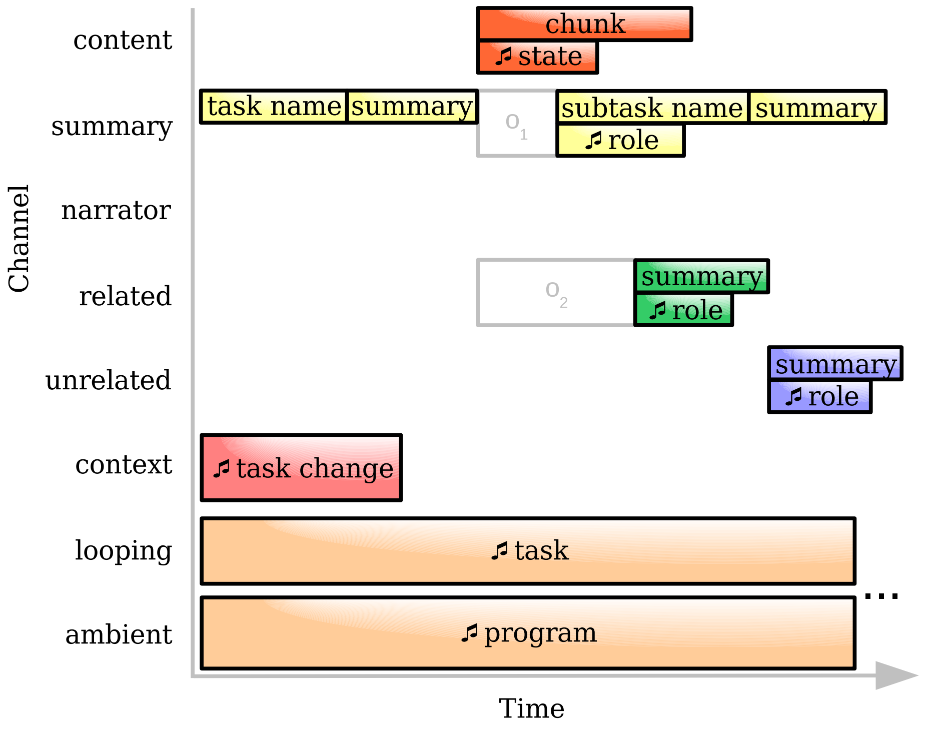


Figure 5.2: Task and subtask activation response streams. The five assistants and two environmental streams are shown on the vertical axis. Time flows from left to right on the horizontal axis. Dark horizontal bars indicate what content is reported during a time period. Musical notes within a bar depict the use of earcons and auditory icons instead of speech. Light horizontal bars represent forced delays.

If the user switches to another task, an action that implicitly activates a different subtask, the summary assistant first announces the name of the new task and a summary of its subtasks. Simultaneously, the context assistant plays the sound associated with the action taken by the user: task started, completed, cancelled, or resumed. The content assistant then announces the first *item* of content in the subtask, where the definition of an item depends on the type of subtask. At the same time, the content assistant may play an earcon indicating a special state of that item. While the content assistant is speaking, the summary assistant reports the name of the subtask and summarizes its contents. This report is delayed by 200 ms from the start of the content report as recommended in Chapter 4.

In some situations, the related assistant may play an auditory icon and speak a summary of a related subtask affected by the activation of the current subtask. If this assistant makes an announcement, its report is further delayed by 200 ms from the start of the summary stream. If a change in an unrelated task happens to occur during the activation announcement, the unrelated assistant waits to make its report until all related announcements are complete.

During all of these reports, the program ambiance is continually playing in the background. In addition, the unique intermittent task sound begins its endless loop after the task activates.

To exemplify this template, consider what happens when a user resumes the task of writing an email in which he or she last edited the subject. The summary assistant first speaks the name of the task, *Continue reply to doe@xyz.com*, and then a summary of its subtasks, *five fields*. At the same time, the context assistant plays the earcon for resuming a *Form Fill* task (defined in Section 5.3.3). Next, the summary assistant says, *Subject*, while the content assistant reads the text already in the subject, namely *Want to go to lpnch?* [sic]. While saying the misspelled word *lunch*, the content assistant plays the misspelling state earcon. All the while, the farm ambiance associated with the email program plays continuously in the background, and the sound of a horse whinnying plays intermittently to represent the task.

5.1.5 Information Seeking and Editing Commands

Within a subtask, a user may perform actions to browse, search, and edit its content. A uniform set of commands enables these behaviors across different types of subtasks.

Category	*Command*
Browsing	Go to next item
	Go to previous item
	Go to next word
	Go to previous word
	Go to next line
	Go to previous line
	Go to first item
	Go to last item
	Go to parent
	Go to child
	Go to root
	Sort in next order
	Sort in previous order
	Read this
Searching	Seek to character
	Start searching
	Stop searching
	Add character to search
	Remove character from search
	Go to next search match
	Go to previous search match
Editing	Insert character
	Insert new line
	Delete item

Table 5.7: Clique subtask commands.

Table 5.7 lists all of the information seeking and content creation commands supported by Clique.

The exact behavior of each command depends on the subtask active at the time it is issued. For example, the next item command navigates to the next menu item in the program or task menu, the next list item in a collection, and the next character in a document. The active subtask also dictates what commands are available depending on the interactions it supports. The program menu, for example, does not support text entry while an editable document does. Section 5.3.3 later in this chapter discusses the connection between supported commands and subtasks in more detail. For now, it suffices to say that the availability and purpose subtask commands change with the user's context.

Regardless of how the function of a command varies across subtask, the manner in which Clique responds to any command follows a fixed set of patterns for browsing, searching, and editing content. In all cases, the virtual assistants report information according to their assigned roles, if such a report is appropriate for the given command. For instance, when the user searches through a large body of text, the content assistant reports the text of the matching item, the summary assistant reports where in the content the match is located, and the narrator echoes the search string. On the other hand, when the user enters a single new character into a body of text, only the narrator responds by announcing the character typed.

Figure 5.3, Figure 5.4, and Figure 5.5 depict the general responses of Clique to browsing, searching, and editing commands. The axes and notation are the same as in Figure 5.2. In these diagrams, one or more of the assistants may not sound for a particular command. For example, when the user navigates to the next word, a state sound may not play if that word has no special state. In other words, the figures depict the maximum possible output from Clique in response to a single command.

5.1.6 Service Commands

Clique provides a set of commands which are accessible at all times and operate in a uniform manner in all contexts. These commands address user requirements when interacting with an auditory display such as the need to repeat information, to inquire about the active context, to ask for more details, and to remember information for later use in other tasks.

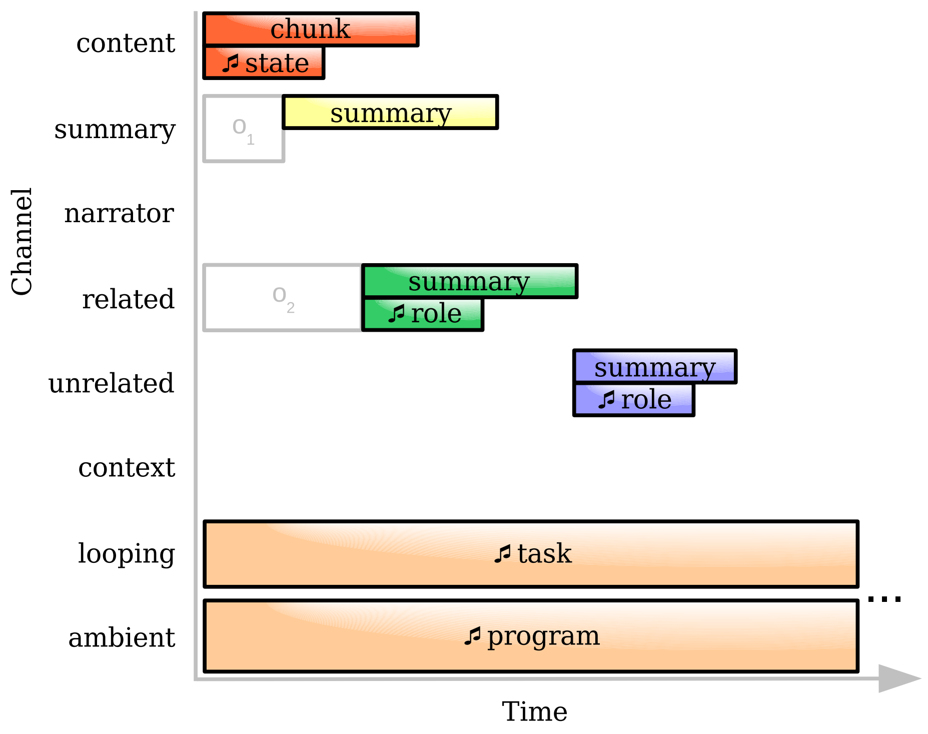


Figure 5.3: Browsing response streams. In comparison with Figure 5.2, the content and summary assistants differ. The content assistant speaks the text at the point of regard and plays an earcon representing the attributes of that content (e.g., link). The summary assistant speaks a summary of the point of regard (e.g., item one of ten).

Stop

Any time the user issues a command, its output preempts any ongoing output from the Clique assistants. Sometimes, however, the user may want to stop all output from the assistants without triggering any additional report. The *Stop* command accomplishes this goal. It immediately silences all output from the content, summary, narrator, related, and unrelated virtual assistants. This command does not affect environmental sounds.

A system- or user-driven event may produce output immediately after the user issues the *Stop* command. *Stop* only silences output active at the time it is given.

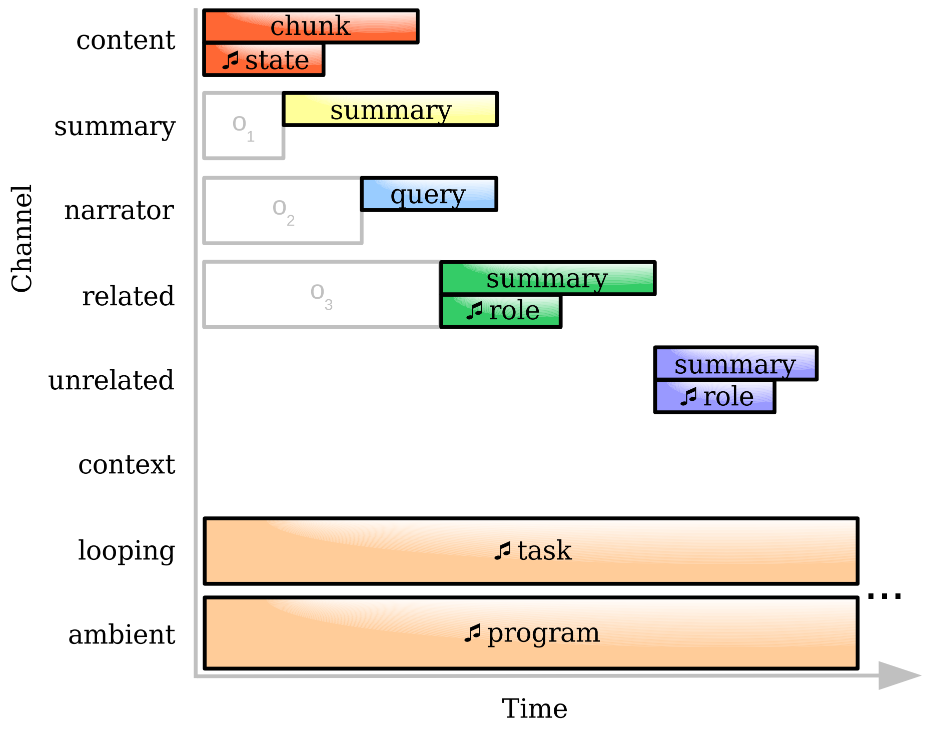


Figure 5.4: Searching response streams. The assistants report the same information as in Figure 5.3, with one addition. The narrator assistant echoes the characters entered by the user to perform the search.

Where Am I?

The ambient and intermittent streams provide a constant reminder of the current task and program. If the user forgets what these sounds represent, however, the streams are useless. Furthermore, the streams do not indicate the active subtask. To account for these problems, the user may issue a *Where am I?* command. The summary assistant responds by speaking the name of the active subtask while playing its role auditory icon, then speaking the name of the active task while playing its paired intermittent sound, and finally speaking the name of the active program.

What Was That?

The related and unrelated assistants are intentionally quiet and in the periphery of the auditory space because their messages are often less important than those from the

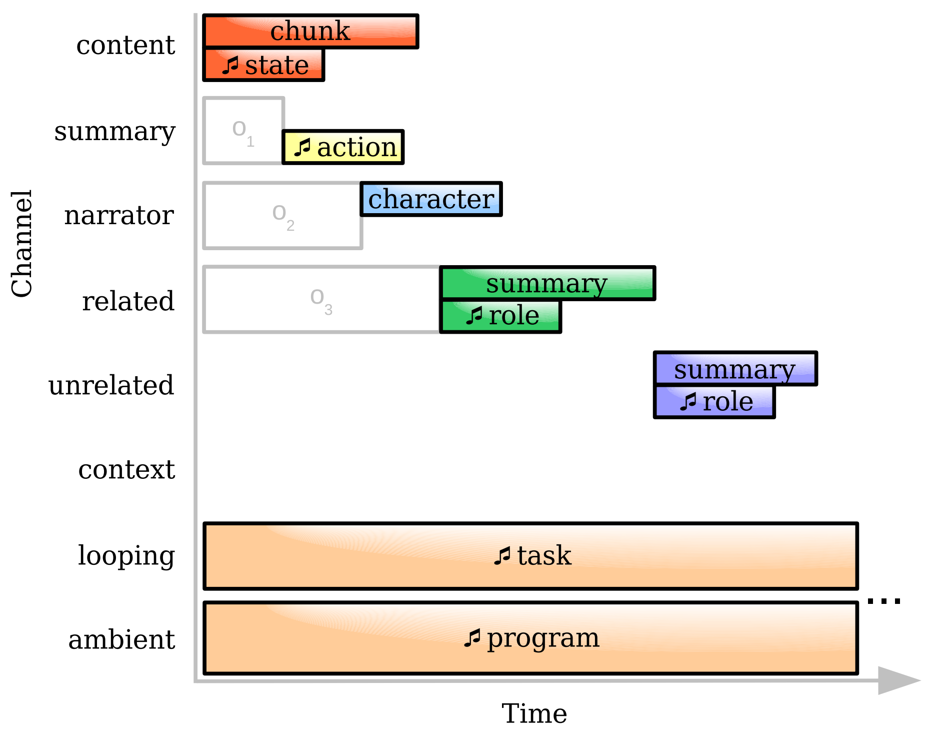


Figure 5.5: Editing response streams. In comparison with Figure 5.4, the content of summary and narrator streams differs. The summary stream plays an earcon to indicate an action taken when editing (e.g., adding a line, joining two words). The narrator stream echoes individual characters as the user types.

active subtask assistants. This design can cause problems when the user does want to hear a peripheral message while the content, summary, and narrator assistants are speaking. In this situation, the user may give a *What was that?* command. The related and unrelated assistants respond by speaking all of the messages they have previously reported in reverse chronological order. Interleaved with these messages are reports of how long ago the announcements were made in one minute blocks to help the user recognize stale information.

Tell Me About This

The content assistant often reports a subset of the available information about the current content in the active subtask. Additional details are omitted in the initial report in the interest of brevity. For example, as the user is browsing a body of text by words, the content voice only speaks the word at the text caret. The user accesses other information such as the spelling of the word and the position of the caret in the document using the *Tell me about this* command.

This command serves a second purpose: asking the active subtask assistants to repeat information they have already announced. The repetition of basic information is useful when the user has missed one or more messages spoken concurrently.

Record This

As reported in Chapter 3, the user often has trouble remembering information gleaned from one context long enough to use it in another. For instance, a user can read an email containing a set of appointments he or she needs to modify in a calendar program. If the length of the list of dates exceeds the number of items a user may hold in short term memory, he or she will need to repeatedly visit the email program to complete the calendar scheduling task.

Clique supports a *Start recording* command which reduces the memory burden on the user. When the user issues this command, Clique begins recording everything spoken by the content assistant. Recording continues until the user gives a *Stop recording* command. For each start and stop pair, Clique keeps a copy of the text spoken for later recall in the memory system menu.

The current implementation of Clique treats the word being spoken when user issues the *Start recording* command as the first word in the memorized text. A more configurable implementation could allow the user to adjust the number of words prior to the start command Clique includes in the recording, either manually or based on past performance. Such a feature could help account for variance in user reaction times and Clique speech rates.

Go to Menu

The user may visit the Clique system menus by giving the *List my programs*, *List my tasks*, and *List my memory* commands. Unlike the other service commands, the task menu is unavailable in one situation: when the user is in the program menu. The user cannot access a task in this context because no program is active.

In the memory menu, the user may browse and search all previously recorded segments of content. All of the browsing and searching commands available in other menus are available here. In addition, the user may use the *Delete item* command to remove recorded segments from memory and the *Complete this* command to insert the current segment into the last task active before the memory menu. Recursive use of the start and stop recording commands is also allowed so that the user may split existing recordings into smaller segments.

Audio Controls

Four other commands exist to give the user very coarse, but very fast control over the output from Clique. *Increase volume* and *Decrease volume* raise and lower the master volume respectively. *Increase rate* and *Decrease rate* raise and lower the speech rate of all voices respectively. For more detailed control, the user must access the volume program as described in Section 5.5.1 of this chapter.

5.1.7 Keyboard Map

As stated in Chapter 1, my primary concern in this research is the design of the auditory display output, and the functions used to control and explore it. How the commands map to a physical input device is a secondary concern, and only of importance since it is a requirement for evaluating the display with users. To produce a simple, familiar, easy to learn implementation for evaluation, I decided to map the Clique commands to a standard keyboard.

Table 5.8 lists all of the Clique commands along with the keys that trigger them. Figure 5.6 shows the spatial layout of the Clique keyboard map. The user activates most of the commands by simply pressing the required keys. Other keys must be prefixed with a modifier key. Note that the majority of the command keys are located on the numeric keypad. Also note that the left, right, up, and down arrow keys may be used in place of Keypad 4, 6, 9, and 2.

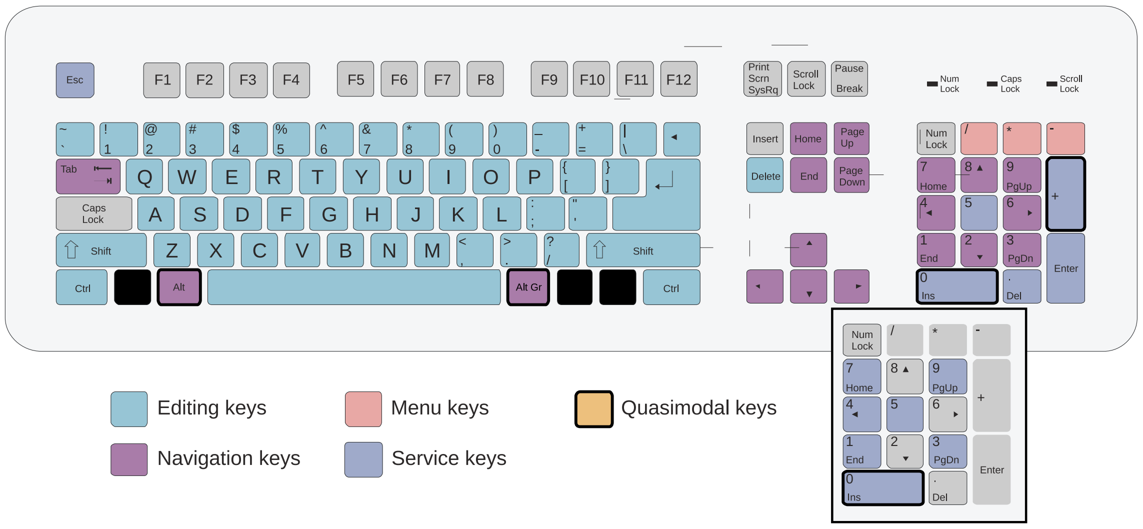


Figure 5.6: Clique keyboard map. The inset shows command keys available while Keypad 0 is held.

Quasimodal Keys

To implement all of the Clique commands on a keyboard, some of the keys must serve more than one function. For example, the system must distinguish use of the character keys for text editing from use of the characters for entering search queries. This problem could be solved by introducing explicit modes into the interaction with a subtask. Clique could, for instance, require the user to visit the task menu, select a *Search this text* task (i.e., the search mode), enter search terms in that task, complete the task, and then return to his or her initial task (i.e., the original mode) in order to hear the first search match. Thereafter, additional shortcut keys could allow the user to navigate to the previous and next match while returning to the *Search this text* task could allow the user to modify the search.

The design of Clique could certainly support this modal solution, but it is less than ideal. The addition of modes in each subtask or new tasks representing the modes requires that the user switch contexts each time he or she needs to invoke the command. Even with persistent reminders and announcements when changing modes, the mode places an extra memory burden on the user and slows down interaction. The user must check what mode he or she is in by listening to the environmental sounds, and Clique must take some time to announce a mode change. On the other hand, if such reminders and announcements are eliminated in the interest of speed, then the user must recall the active mode from memory alone.

To avoid these problems, Clique supports the concept of *quasimodal commands*. A user activates a quasimode by pressing and holding a key and leaves the mode by releasing it. The tactile and proprioceptive sensation of pressing a key serves as an effective reminder of a special mode over an extended period of time (Raskin, 2000). GUI keyboard accelerators (e.g., *Alt-F* to access the File menu) are primitive implementations of quasimodal commands. The user must press *Alt* before pressing *F* to trigger the opening of the file menu instead of the insertion of the letter *F* in text. Capitalization of text by holding *Shift* represents a true quasimode. Holding *Shift* while pressing one or more alphabetic keys produces multiple capitalized letters.

Clique has three quasimodes. First, pressing and holding either of the two *Alt* keys puts Clique into the searching quasimode by issuing the *Start searching* command. While *Alt* is held, all character key presses append to the keyword search string (*Add character to search*). Pressing *Backspace* while holding *Alt* removes the last character in the search (*Remove character from search*). Using the left and right arrow keys while holding *Alt* skips to the previous and next match for the entered text using the *Go to previous search match* and *Go to next search match* commands. Releasing the *Alt* key issues the *Stop searching command*, leaving the last announced match as the active item.

Second, pressing and holding the *Keypad +* key puts Clique into recording mode by giving the *Start recording* command. While the user holds this key, Clique marks all text spoken by the content assistant for later recall in the memory menu. When the user releases this key, he or she issues the *Stop recording* command and Clique stops remembering text. The use of this key does not interrupt output from other commands, nor does it preclude the user from giving other commands while recording. For instance, the user may visit the program menu, hold the recording key, press right arrow three times, and then release the recording key. This operation will cause Clique to place the three program names announced into one chunk in the memory menu.

Third, pressing and holding the *Keypad 0* key puts Clique into system mode. Pressing keys on the numeric keypad with zero held results in the execution of system commands like *Increase volume* and *Decrease rate*. Compared to the search and memory modes, the system mode is simpler and more reminiscent of basic accelerator keys in GUIs.

To support quasimodes using these keys, Clique must prevent certain keystrokes from reaching the underlying GUI applications. Section 5.3.1 describes this Clique function.

5.2 Satisfied User Requirements

Table 3.4 at the conclusion of Chapter 3 summarizes the requirements of supporting a working user. Table 5.9 revisits these requirements and notes how the interaction design described in the preceding sections satisfies each one.

	Guideline	*Support*
1	Allow casual and directed browsing	Continuous reading, explicit browsing
2	Support searching given partial information	Keyword searching
3	Make switching between browsing and searching effortless	Quasimodal searching
4	Provide a “home” for starting the seeking process	Program and task menus
5	Elucidate connections between content objects	Information from the related virtual assistant
6	Enable filtering of content objects	Navigation over keyword matches
7	Notify users about relevant content changes	Related and unrelated assistants
8	Support the transfer of content objects to new contexts	Recording command and memory menu

	Guideline	*Support*
9	Allow for reviews of past content	Memory menu and commands for repeating information
10	Summarize content objects with previews	Summary and related assistants
11	Establish context with overviews	Environmental sounds, introduction of active task and subtask
12	Provide details on demand	Detail commands
13	Let queries and commands be directed at particular views	Active task commands versus commands for related and unrelated speakers
14	Support rapid transitions among views	Attention switching among simultaneous streams, interruptible commands for active assistants versus peripheral assistants
15	Allow navigation through text by character, word, and chunk	Editing commands
16	Participate in grounding	Confirm all commands, interruptions when important, waiting when busy
17	Contribute information at an appropriate level of detail	Brief automatic announcements, commands for more detail
18	Support multitasking with interruptions and task switches	Active versus inactive tasks, task menu navigation, peripheral reports
19	Let queries and commands be directed at particular tasks	Active versus inactive tasks
20	Respect maxims of quality and manner	Consistent output format, most relevant information first
21	Explain limitations of the interface	Report when commands cannot execute
22	Enable repair	Task cancellation

Table 5.9: Working behaviors supported by Clique

5.3 Software Architecture

The architecture of Clique is designed to provide the user experience described in the preceding sections while making adaptation of GUI applications tenable for developers.

At the highest level, the Clique architecture follows the model-view-controller and chain of responsibility design patterns (Gamma et al., 1995). The input manager captures user key presses and converts them into input command messages. The input manager dispatches the messages on an *input pipeline* flowing through the output manager [1], active program, active task, and active subtask controllers. Any component in this chain may act on the command. Typically, program, task, and subtask controllers invoke methods on their corresponding models to query for information or change the state of the model. Views of a program, task, and subtask receive notification of the changes and may query both models and controllers for relevant details. Usually, views act on changes by producing output messages. The output messages flow through the *output pipeline* running from the subtask, task, program views and ending at the output manager. The output manager interprets these messages and performs the auditory equivalent of a visual compositing operation to produce the final auditory scene.

Clique provides pre-defined controllers and views for *interaction patterns* common across applications. Scripts created for specific applications specialize and instantiate these patterned components to define the available tasks and subtasks in a program, structure the navigation among tasks and subtasks, and define conditions for starting, completing, and cancelling tasks. Clique also defines a set of model adapters that wrap GUI widgets in order to provide interfaces understood by the controllers and views. Scripts may use the adapters to wrap widgets in existing GUI applications for use with the Clique interaction patterns. When necessary, scripts may introduce new models, views, and controllers to account for interactions not covered by the predefined components.

For reference, Figure 5.7 reproduces the MVC diagram of Clique as an auditory agent first shown in Chapter 2. The Clique model of an unaware application is based on data extracted from its GUI, the functions supported by its GUI, and the structuring of

[1]The output manager is part of the input pipeline. See Section 5.3.1 for further explanation.

the data and functions into tasks defined by a third-party script. The Clique auditory display presents the available tasks in the manner described in the first part of this chapter. The user issues commands to Clique alone to advance his or her tasks, while Clique carries out whatever actions are necessary on the underlying application GUI to fulfill those commands.

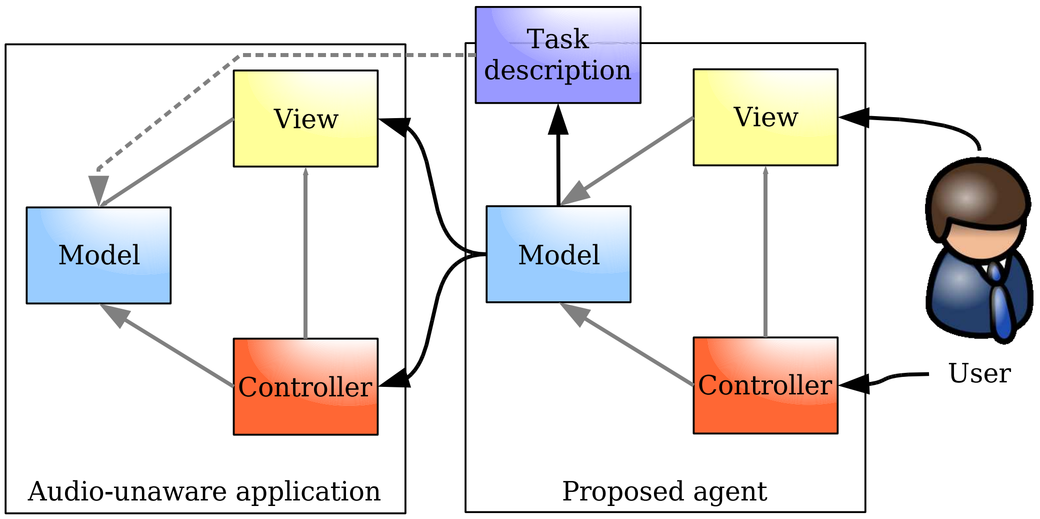


Figure 5.7: Clique model-view-controller diagram.

5.3.1 I/O Pipeline

The following sections delve into the architecture of Clique following the flow of control originating from user input and ending with audio output. This particular flow is the most prevalent in Clique, and is referred to as the *input/output pipeline*. Other, less common control flows exist and are described in later sections of this chapter. Figure 5.8 depicts the pipeline architecture of Clique described in depth in the following sections.

Input Manager

The input manager translates keyboard gestures into Clique commands. It does so by capturing system-wide key presses and releases, preventing them from reaching any other

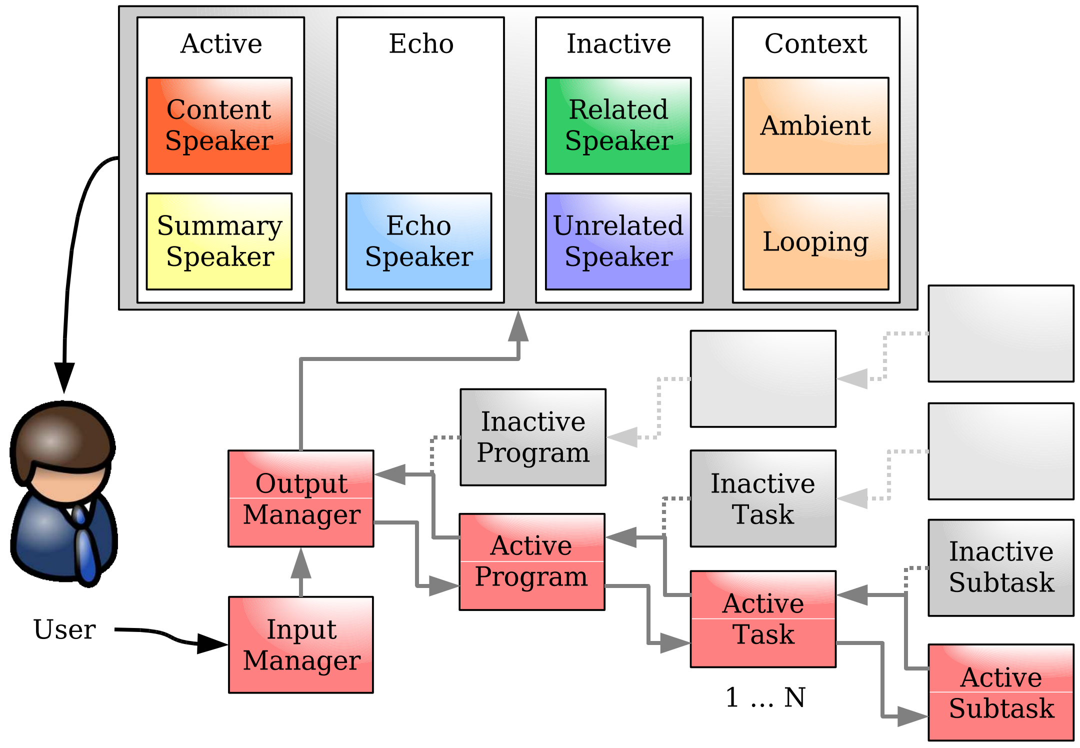


Figure 5.8: Clique input/output message pipeline architecture. Input messages originate from the user and flow from the input manager through the output manager, active program, active task(s), and active subtask. Output messages flow from any live subtask, task, or program itself. Components closer to the output manager are responsible for routing output messages from farther components to the appropriate output groups and speakers. The groups and speakers are responsible for synthesizing the speech and sounds the user ultimately hears in the auditory scene.

application on the system. This behavior allows Clique to map commands to arbitrary key combinations without interference from other applications.

When the input manager notices a particular gesture on the keyboard, it dispatches a message for the corresponding command through the input pipeline. The *input pipeline* is composed of components providing an interface for forwarding *input messages* to the next, active component in the pipe. The forwarding protocol matches that of the W3C Event Model (Pixley, 2000) involving both a capturing and bubbling phase. In the capturing phase, messages flow away from the input manager toward the furthest active component in the pipe. During this phase, any component in the pipe may take action in response to the message, including consuming it. When the input message reaches a component with no link to another active component, the bubbling phase begins. The same input message propagates in the reverse direction through the pipe heading back to the input manager. Again, any component may act on the message during this phase and potentially consume it. This two-phase design allows all active components to respond to an input command both before and after any other active component makes changes to the internal state of the system.

The active segments of the input pipeline change as the user navigates among the available programs, tasks, and subtasks. For example, imagine the user is listening to the task menu of a Web browser program. When the user presses the key for the *Complete this* command to choose the *Enter a URL* task, the pipeline routes an input message from the input manager, to the output manager, to the object representing the Web browser program, to the task menu object for that program. After the command runs and the user is in the URL task, he or she may type characters. For each character pressed, the input pipeline routes command messages from the input manager, to the output manager, to the Web browser program object, to the *Enter a URL* task, to the editing text subtask.

Though it may seem misplaced, the output manager is always the second active component in the input pipeline after the input manager. The purpose of this design is to allow the output manager to quickly process certain commands that have no bearing or requirements on programs, tasks, or subtasks. For instance, the output manager can execute the *Stop* and *What was that?* commands successfully without any information about the user's current context. The output manager simply stops all speech output in the first case and replays the history of peripheral speech and sounds in the second.

Controllers

When a component on the input pipeline receives a message, its controller may process that message. The controller decides what action to take in response to the command depending on the role of the component to which it belongs. For example, if the user issues the *Go to next item* command when the program menu task is active, its controller updates the state of the program menu model so that the next item is now active. If, instead, a subtask for editing text is active, its controller updates the state of the subtask model so that the reading position is on the next character.

As a more complex example, assume the user issues the *Where am I?* command while entering the subject of a new email message in an email program. During the bubbling phase of the input pipeline, the subtask, task, and program controllers query their models for a description of the working context (later reported by their views). The output manager and input manager controllers do not handle the *Where am I?* command, however, and do nothing in response. Thus, any subset of controllers in the input pipeline leading from the input manager to the deepest active component may query or update their models in response to a single message.

Models

Each program, task, and subtask in Clique is based on a model that encapsulates data and provides methods of accessing and manipulating that data. A model implements one or more well-defined *interfaces* describing what controller actions the model may handle and what information a view may request from the model for display. Clique includes a pre-defined set of interfaces shown in Table 5.10 supporting many common browsing, searching, and editing interactions on structured data.

Some models implementing these interfaces are wholly defined within Clique. For example, the main program menu model supports the *Interactive*, *Seekable*, *Sortable*, and *List* interfaces to enable the activation, browsing, and searching of the program

Command	*Key*	*Notes*
List my programs	Keypad -	
List my tasks	Keypad *	
List my memory	Keypad /	
Start recording	Keypad +	While pressing
Stop recording	Keypad +	On release
Tell me about this	Keypad 5	
Stop	Keypad 0	Serves as a modifier
What was that?	Keypad 0 + Keypad 4	
Where am I?	Keypad 0 + Keypad 5	
Increase volume	Keypad 0 + Keypad 7	
Decrease volume	Keypad 0 + Keypad 1	
Increase rate	Keypad 0 + Keypad 9	
Decrease rate	Keypad 0 + Keypad 3	
Complete this	Keypad Enter	
Cancel this	Escape	
Go to next subtask	Tab	
Go to previous subtask	Shift + Tab	
Go to next item	Keypad 6	
Go to previous item	Keypad 4	
Go to next word	Ctrl + Keypad 6	
Go to previous word	Ctrl + Keypad 4	
Go to next line	Keypad 2	
Go to child	Keypad 2	
Go to previous line	Keypad 8	
Go to parent	Keypad 8	
Go to first item	Keypad 7	
Go to last item	Keypad 4	
Read this	Keypad .	Reads continuously
Sort in next order	Keypad 9	
Sort in previous order	Keypad 3	
Delete item	Delete	
Seek to character	Any character key	
Start searching	Alt	While pressing
Stop searching	Alt	On release
Add character to search	Alt + Any character	
Remove character from search	Alt + Backspace	
Go to next search match	Alt + Keypad 6	
Go to previous search match	Alt + Keypad 4	
Insert character	Any character	
Insert new line	Enter	

Table 5.8: Clique keyboard commands.

Name	*Specializes*	*Description*
Context		Provides references to other models related to or containing this one
Interactive		Supports interaction with the user
Seekable		Supports searches given a single input (e.g., first character)
Searchable		Supports searching by arbitrary keywords and navigation through a linear list of results
Sortable		Supports sorting of items by one or more criteria
Selectable		Supports operations that select or deselect all items
Deleteable		Supports deletion of items
Strideable		Supports navigation by spans larger than a single item
Infinite Collection		Supports navigation through an unbounded set of items
Finite Collection	Infinite Collection	Supports navigation through a bounded set of items
List	Finite Collection	Supports queries on a one-dimensional set of items
Table	Finite Collection	Supports queries on a two-dimensional set of items
Tree	Finite Collection	Supports queries on a list of lists
Label		Supports queries on a static chunk of text
Text	Label	Supports queries and navigation on a static chunk of text
Editable Text	Text	Supports queries, navigation, and editing on a chunk of text
Hypertext	Text, Detailable	Supports queries and navigation within a chunk of text containing other rich elements, including links

Table 5.10: Interfaces implemented by Clique models and their inheritance relationships.

menu contents. This model maintains the list of available programs and the currently selected program without reliance on any other component.

Most models defined in Clique, however, act as *adapters* for components in other programs. These models maintain very little information themselves, and instead query and control external components to satisfy requests from Clique controllers and views. Such adapter models communicate with external components using a combination of platform APIs, application specific APIs, accessibility APIs driving GUI widgets, or even keyboard and mouse events: essentially, any means necessary to manipulate other programs in order to maintain the model state.

For instance, the Clique email script for Microsoft Outlook Express has a model representing the list of messages in the user's inbox. This model provides the *Interactive*, *Seekable*, *Sortable*, and *List* interfaces, just like the main program menu described above. Unlike in the program menu, the data accessible in the messages list (i.e., sender, subject, and date received) are not stored in Clique. Instead, the list model acts as an adapter for the messages list box widget in the Outlook Express GUI.

When the Clique controller for this list handles the *Go to next item* command, it uses the list interface on this model to indicate the next item in the list should become the active item. The model, in turn, uses the public accessibility API exposed by the GUI list box widget to select the next item in the list. Later, when the controller queries the model to satisfy the *Tell me about this* command, the model may again query the GUI list box widget for the information about the selected item or use previously cached data (if the model can guarantee it is fresh, not stale). In summary, the Clique model uses the items in this list as its data and the functions of the GUI widget as a means of reading and changing the data.

These adapters and their extensibility are the key to reusing the business logic in existing GUI applications while supporting interaction through an auditory display.

Section 5.3.4 discusses this topic in more detail. In addition, Section 5.5 presents a table of adapters implemented in the Clique prototype.

Views

Once the controller has processed an input command and its model has updated state, the view queries the model and controller for relevant information to report. For example, the view may request the name of a newly selected item, its index in a collection, and the key pressed to jump to the item. The view packages this information in a set of *output messages*. Each message contains metadata describing the following key attributes:

- The text to speak
- The sound to play
- The classification of the content of the message (i.e., content, summary, narrator)
- The controller associated with the view that generated the output message (to support callbacks during output for additional actions)
- The input message that initiated the report (to indicate whether output was user-driven or system-driven)
- Other attributes of the message content that may influence its output (e.g., spelling instead of pronouncing)

The view combines multiple output messages into a *output packet*. The packet acts as an enclosure for all messages reporting on a single topic. For example, in Figure 5.2, the summary voice reports information about both a task and subtask in response to the activation of a new task. The task view creates a message classified as a task summary and populates it with a sound representing the task role and speech reporting its label.

The task view places this single message in a packet, a report about the newly activated task.

The subtask view creates two messages, one classified as content and the other as summary. The content output message contains text from the subtask and, possibly, a sound indicating its state. The summary message contains the label and overview of the task content as well as a sound represent the subtask role. The subtask view packages these two messages into one packet, a report about the subtask activated as a consequence of changing tasks.

After producing one or more packets, a view dispatches them on the *output pipeline.* The output pipeline consists of components providing an interface for forwarding output packets to the next active component in the pipe and modifying packets received. The forwarding protocol reproduces only the bubbling phase of the W3C Event Model. Message packets only flow "up" the pipe from a subtask, task, or program view to the output manager where the output packet is processed for playback. In essence, the flow of packets in the output pipeline mimics the flow during the bubbling phase of the input pipeline, but without ever reaching the input manager.

Each pipeline component that receives an output packet has the ability to modify it before it reaches the output manager. This design enables components closer to the output manager to redirect output messages to appropriate assistants. For instance, output messages originating from the active subtask, task, and program pass through the pipeline unchanged to the active assistants: the content, summary, and narrator. On the other hand, when an inactive subtask in an active task generates an output packet, the task takes responsibility for redirecting it to the related assistant. If the packet instead originates from anywhere in an inactive task, its program component redirects it to the unrelated assistant.

Program and task views are also responsible for creating and dispatching message packets containing environmental sounds representing the user's context. These packets, like all others, propagate through the pipeline toward the output manager.

Output Manager

The output manager is the final destination for all packets in the output pipeline. Upon receiving a packet, the manager checks whether the output is from an active subtask, active task, or an inactive task. It uses this information to forward the packet to one of the four *output groups* depicted in Figure 5.8. The output group opens the packet and forwards the messages inside to their final destination, a *speaker* component that will speak text or play a sound.

Each output group component follows a set of rules determining when it forwards messages to a speaker, when it should silence all speakers in the group, and when it should preempt a message from being reported. One *active group* manages output packets destined for the content and summary assistants. When this group receives a packet containing any message generated as a direct consequence of user input (i.e., it contains a reference to an input message), the group immediately interrupts its output and begins playback of the new packet speech and sound immediately. On the other hand, the active group queues all messages not created as an immediate result of user input. When the speaker components in the group have finished saying or playing all of the messages in a packet, the active group proceeds with processing the next packet. The group is responsible for introducing the short delay at the start of the summary output in accordance with Figures 5.2 through 5.5.

A second active group, the *echo group*, manages messages destined for the narrator speaker. This group follows the same rules as the first active group, but allows the narrator assistant to start and stop output independently of packets in the other active group. Figure 5.5 shows that the narrator assistant announces characters typed during text editing while the content assistant speaks words as the user completes them. If the user types another character while Clique is saying the last character entered and word completed, the narrator assistant interrupts the character announcement, but the report of the word continues unabated. The separation of the content and narrator speaker components into two groups permits this ability.

The third group, the *inactive group*, processes packets sent to the related and unrelated assistants. The group first waits for the prescribed delay to offset its output from any output by the active group. During this wait, the inactive group collects all packets it receives. When the group receives a second packet from the same source view, it discards any previous packet from that source. After the required wait is over, the group sorts the retained packets in reverse chronological order with all packets for the related speaker appearing before packets for the unrelated speaker. The inactive group then iterates over the retained packets and takes one of two possible actions on each. If the packet was originally intended for the related or unrelated assistant, the group forwards its messages to the appropriate speaker unchanged. If the packet was supposed to reach the active group, but some component in the pipeline modified its ultimate destination, the group finds the best available summary sound and speech in the packet and constructs a new message. It then forwards this synthesized message to the appropriate speaker which reports the summary of the original, more detailed information.

The fourth and final group, the *context group*, manages packets containing program and task sounds from the environmental theme for the active task and program as well as drum loops for the Clique menus. This group dispatches messages directly to the looping (menu), intermittent (task), and ambient (program) speakers without modification. The speakers themselves are responsible for playing the sounds properly. The looping speaker ensures menu drum loops continue without interruption as long as the menu is active. The intermittent speaker plays the sound representing the active task at random times within a three to ten second interval. The ambient speaker continuously plays the environmental sound representing a program after fading out the previous sound and fading in the current one.

5.3.2 Other Control Flows

Not all output from Clique originates from user actions. Other control flows exist that result in reports from the Clique assistants.

Model Events

Models that act as adapters may observe events from external applications. When an event occurs, an adapter may notify its associated view of the change. The view may then generate an output message describing the event to the user. The roles of the model and view instances handling the event in a particular application script determine how Clique reports the event, if it does at all. For example, the email script for Clique binds the messages list model to the message list box in Microsoft Outlook Express. The list model observes accessibility events about changes to the number of messages in the list box widget. The model notifies its corresponding view when a change occurs. The view, in turn, sends an output message reporting the number of messages in the current mailbox. The components in the output pipeline direct the message packet to the appropriate output group and speakers depending on whether the email program, browse mail task, or message list subtask are active or not.

Controller Events

In some instances, knowledge is available at design time about how changes in one task or subtasks the affect another. Clique controllers can use this knowledge to send input messages to other controllers in order to force them to update their models and report their changes peripherally. For example, in the Day by Day Professional calendar program, selecting a date in the calendar always causes a list box widget to update to show the appointments on that date. The Clique script for Day by Day takes advantage of this association to provide a peripheral summary of the notes on a date when selecting a date. When the controller for the date subtask receives a navigation input message (e.g., *Go to next item*), it notifies the controller of the notes list subtask of the change. The notes list controller instructs its model to update which in turn refreshes the list view. The list view queries for the total number of appointments on the newly selected date, and dispatches an output message containing this information.

It is important to note that this cross task and subtask communication could be implemented between models instead of controllers. Messaging between controllers is supported in the current design because input messages already provide this communication channel.

Output Events

Views that create output messages and packets may register to receive notifications of when a speaker reaches a certain point in a speech utterance, when a speaker finishes outputting an entire message, or when a group of speakers finishes playing an entire output packet. If a view is observing output events, the speaker or group reporting the events creates input messages as notifications. The speaker or group then dispatches these messages through the input manager to the listener. Unlike standard input messages, these messages are dispatched directly to their observers and do not pass through other components in the input pipeline. For instance, when the user issues the *Read this* command within a large body of text, the view for the subtask breaks the text into quickly renderable chunks and dispatches output packets for the first two. When the first completes, the view buffers the next chunk to ensure there is no unnecessary pause in output. This chunking method keeps the system responsive during speech synthesis. As another example, the output manager observes speech progress from the content assistant. Between the *Start recording* and *Stop recording* commands, the output manager stores all words spoken for later recall in the memory menu.

5.3.3 Interaction Patterns

The models, views, and controllers are the atoms of the Clique user experience. They enable user control over auditory programs, tasks, and subtasks, regardless of whether the application logic is implemented directly in a Clique script or adapted from a preexisting program. What they lack, though, is advice for how they should be combined to create useful and usable function. *Interaction patterns*, solutions for recurring interface design problems, give this necessary guidance. Furthermore, in the context of this work, design pattern components act as templates for implementing the recommended solutions in Clique scripts.

Interaction patterns are a form of *design patterns*, repeatable solutions to problems in a domain of knowledge. First applied to the field of architecture by Christopher Alexander (Alexander, 1979), design patterns initially found use in computer science as a method of documenting best-practices for software architecture (Gamma et al., 1995) and later interface design for desktop applications (van Welie and van der Veer, 2003), Web sites (Graham, 2002), Web applications (`http://developer.yahoo.com/ypatterns/`), and even auditory user interfaces (Frauenberger et al., 2004; Frauenberger et al., 2005; Frauenberger et al., 2007). In each of these disciplines, the set of related design patterns forms a *pattern language* documenting common problems, potential solutions, principles on which solutions are based, example implementations of solutions, and relationships between patterns. The intent of the pattern language is to act as a reference for a practitioner seeking to solve a problem in his or her work without "reinventing the wheel."

In Clique, the auditory interaction pattern language is *componentized* (Arnout, 2004) such that the patterns exist as prototypes that a script writer specializes to satisfy task or subtask interactions in a particular application. Each pattern component encompasses a controller and view providing the input and output support necessary to support an application task or subtask. A Clique script instantiates these patterns, relates them to one another such that the user may navigate within and across them, and binds them to models. The pattern components become part of the input and output pipeline when activated, exactly like any other built-in Clique component.

Since the interaction pattern components consist of a controller and a view, but are not tightly coupled to any one model, the patterns may be used with any model instance that provides the necessary interfaces. For instance, the *List* pattern is successfully bound to models adapting the messages list box widget in Microsoft Outlook Express, the file list box widget in WinZip, the file type drop down list in the Windows file dialog, and even a set of radio buttons in one of the Clique questionnaire programs. In all cases, the supported input commands and the structure of the output is the same as dictated by the Clique list subtask. This ability to apply patterns in diverse contexts is the source of consistent interaction and code reuse in Clique.

Table 5.11 briefly names and describes the primary subtask and task interaction patterns provided by the current Clique prototype. For further clarification, the two sections following the table present more detailed examples of two Clique interaction patterns.

Example: Tree

The tree subtask interaction pattern supports the browsing and searching of hierarchically structured data. The *Go to previous item* and *Go to next item* support navigation within a level of the tree while the *Go to parent* and *Go to child* support navigation to higher and lower levels of containment. Pressing a character key invokes the *Seek to character* within the current level of the tree just as using quasimodal search operates within the current level. The tree pattern reports its role using an auditory icon and its

Pattern	*Level*	*Description*
List	Subtask	Use to support selection of one or more items in a flat set of items.
Tree	Subtask	Use to support selection of one or more items in a hierarchical set of items.
Table	Subtask	Use to support selection of one or more items in a two-dimensional grid.
Text	Subtask	Use to support the reading of plain text.
Editable Text	Subtask	Use to support the editing of plain text.
Hypertext	Subtask	Use to support the reading and navigation of rich text.
Form Fill	Task	Use to support the interaction with and navigation over independent subtasks.
Folder Browsing	Task	Use to support the navigation over and selection of one directory.
File Opening	Task	Use to support the navigation over and selection of one file.
File Saving	Task	Use to support the navigation over directories and entry or selection of a file name.
Run and Report	Task	Use to report a change of state after running a simple operation.
Run, Wait, and Report	Task	Use to report progress during a long running operation and a change of state at its completion.
Linked Browsing	Task	Use to support the interaction with and navigation over dependent subtasks where a change in one subtask affects one or more of the others.
Wizard	Task	Use to support the completion of a set of arbitrarily complex tasks in which one task cannot be completed until its predecessor is complete.
Program	Program	Use to support navigation across tasks in a single application using the task menu.

Table 5.11: Clique interaction patterns.

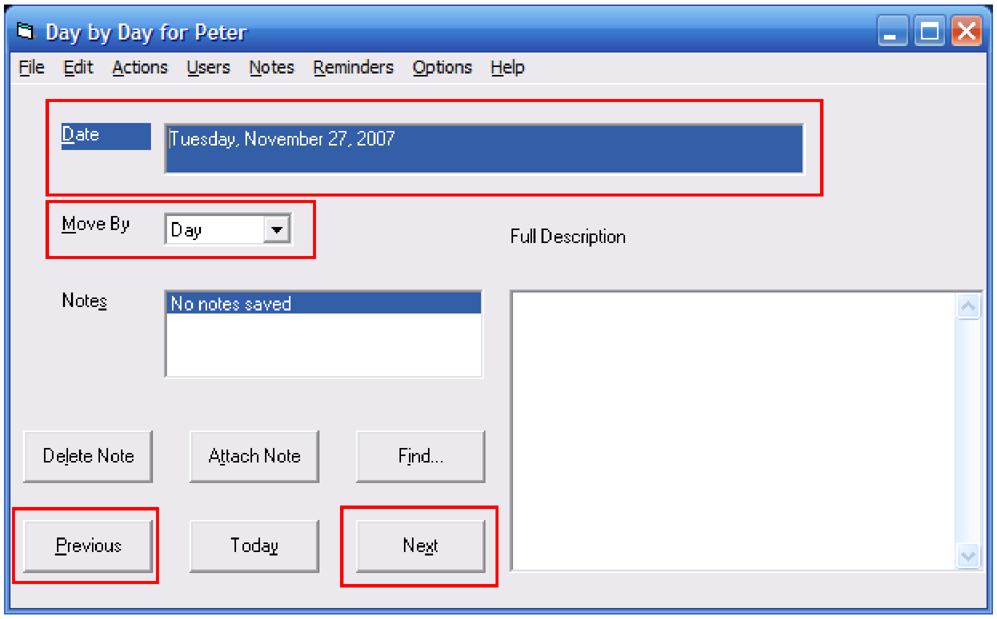
label with speech upon activation. It indicates an item in the tree is navigable (i.e., has children) with an auditory icon, and confirms navigation across tree levels with earcons.

The email program script for Clique uses this pattern to support selection of a mailbox in Microsoft Outlook Express. The script binds the *Tree* interaction pattern controller and view to a model adapting the GUI tree widget to the *Interactive*, *Seekable*, *Searchable*, *Strideable*, and *Tree* interfaces. As the user issues the commands noted above, the model drives the selection in the mailboxes tree widget.

The calendar script for Clique also uses the tree pattern to support navigation of dates by day, week, month, and year. Again, the calendar script binds the tree pattern control and view to a model. But in this case, the model adapts numerous GUI widgets to support all of the interfaces required by the pattern. The model adapts the GUI *Next* and *Previous* buttons to the *Go to next item* and *Go to previous item* commands. It binds navigation of the *Move by* drop-down list to the navigation commands *Go to parent* and *Go to child* to support browsing the calendar by different strides (i.e., day, week, month, year). Finally, it responds to queries about the currently selected item by inspecting, formatting, and reporting the date text in the read-only text box widget.

Figure 5.9 shows these visual widgets in the calendar GUI interface. Despite their visual separation, they all contribute to the *Tree* interaction pattern and the subtask of navigating dates in Clique.

Both the email and calendar scripts use the same *Tree* interaction pattern component to support navigation over mailboxes and dates respectively: the code for the *Tree* pattern is reused. On the other hand, the model adapters relating the *Tree* patterns to the email mailbox GUI widget and the calendar buttons, drop-down list, and text field GUI widgets differ across scripts. Each adapter applies specifically to one type of widget.



Tuesday, November 27, 2007

Day

No notes saved

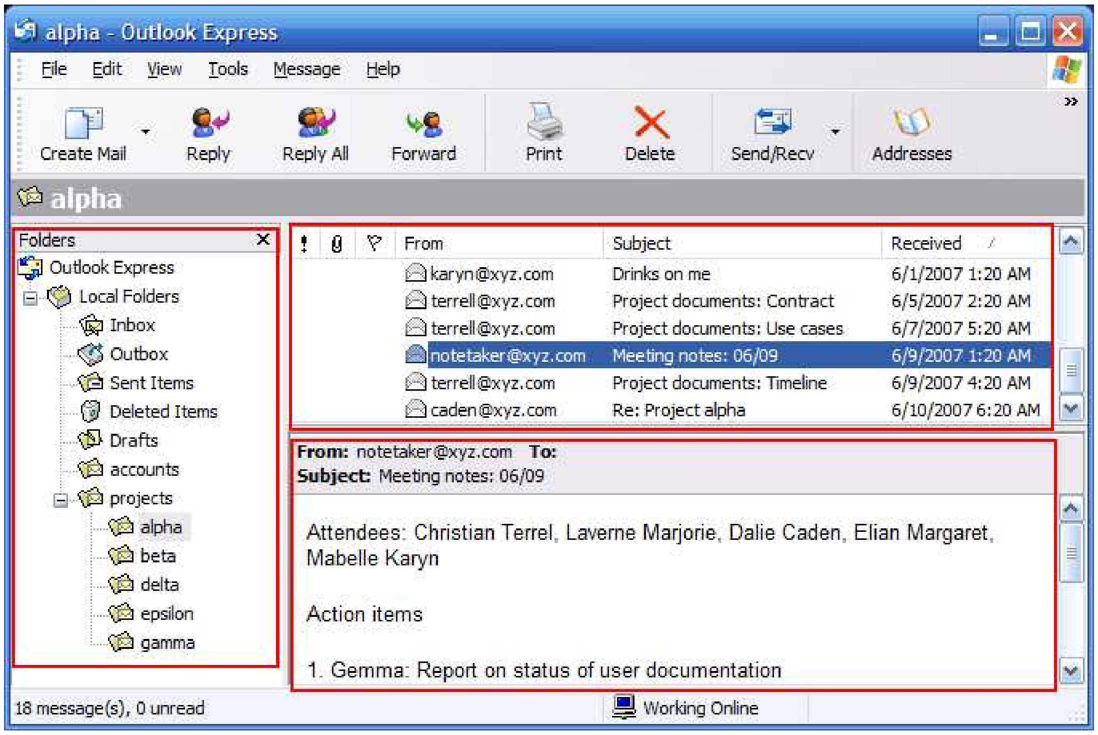
Figure 5.9: Tree pattern in Day by Day Professional. The boxed *Date*, *Move By*, *Previous*, and *Next* widgets contribute to the pattern.

Example: Linked Browsing

The *Linked Browsing* pattern connects subtasks within a task that have an effect on one another. The pattern supports navigation over the tasks via the *Go to next subtask* and *Go to previous subtask* commands. More importantly, it supports notifications among subtasks when a change in one subtask affects the state of another.

The *Browse Mail* task in the email program script uses the *Linked Browsing* pattern to provide peripheral notifications. This task relates the *Tree* subtask for browsing mailboxes to the *List* subtask for browsing messages such that as the user navigates the mailboxes, the message list subtask announces how many emails are in the currently selected mailbox. Likewise, the task relates the messages list to the *Text Reading* subtask for the message body. Navigation over the message headers causes message length announcements from the message reading subtask.

The calendar script also uses the linked browsing pattern in its *Browse Calendar* task. This task relates the *Tree* subtask for navigating dates to the *List* subtask for browsing appointments on a selected date. As the user visits different dates in the calendar, this



From	Subject	Received
karyn@xyz.com	Drinks on me	6/1/2007 1:20 AM
terrell@xyz.com	Project documents: Contract	6/5/2007 2:20 AM
terrell@xyz.com	Project documents: Use cases	6/7/2007 5:20 AM
notetaker@xyz.com	Meeting notes: 06/09	6/9/2007 1:20 AM
terrell@xyz.com	Project documents: Timeline	6/9/2007 4:20 AM
caden@xyz.com	Re: Project alpha	6/10/2007 6:20 AM

From: notetaker@xyz.com **To:**
Subject: Meeting notes: 06/09

Attendees: Christian Terrel, Laverne Marjorie, Dalie Caden, Elian Margaret, Mabelle Karyn

Action items

1. Gemma: Report on status of user documentation

Figure 5.10: Linked browsing pattern in Microsoft Outlook Express. The boxed *Folders*, *Messages*, and *Message Body* widgets contribute to the pattern.

relationship causes the appointments list to report the number of appointments on that date. In the same manner, the *Browse Calendar* task relates the appointments list to the *Text Reading* subtask for browsing details about a particular appointment. Again, as the user browses the appointments on a particular date, the details subtask summarizes the currently selected appointment.

Figures 5.10 and 5.11 highlight the visual widgets that the email and calendar scripts adapt to the linked browsing task. These examples demonstrate the application of one interaction pattern to tasks in two applications with very different graphical interfaces and intents.

Both the email and calendar scripts use the same *Linked Browsing* interaction pattern component to relate subtasks to one another. Once again, the code for the *Linked Browsing* pattern is reused. The subtasks linked in each script differ, but the pattern applies unchanged to both situations.

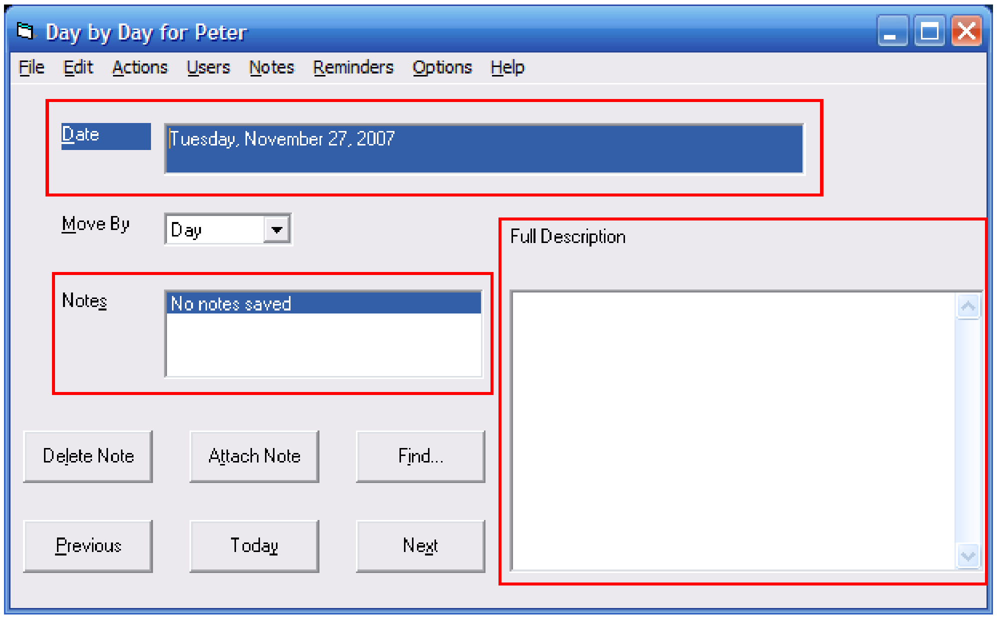


Figure 5.11: Linked browsing pattern in Day by Day Professional. The boxed *Date*, *Notes*, and *Full Description* widgets contribute to the pattern.

5.3.4 Scripts

Task and Subtask Definition

Clique scripts are components that define the tasks and subtasks available during interaction with an application. A script specializes one or more task interaction patterns to fit the purpose of the program. For instance, the WinZip archive manager supports the creation of a new zip file archive through a series of GUI dialogs. The Clique script for WinZip has a *New Archive* task definition, a specialization of the *Wizard* interaction pattern, which guides the user through the major steps of naming the archive, selecting the files, and specifying archive options.

The definitions of the tasks indicate how and when tasks start, if tasks are singletons or not, if tasks can be completed or canceled, the order of subtasks within tasks, the relationships among the tasks and subtasks, and what *successor* task, if any, should run when a task completes. A script binds the task interaction patterns to appropriate models, either by defining a new model inside the script or by using one of the predefined adapters for external GUI widgets. A script performs the same binding of subtask interaction patterns to internal models or adapters for external widgets.

Bindings to Widgets

Scripts that adapt GUI widgets for use as models in their tasks and subtasks require a method of binding the adapters to the external widgets. Such a binding has two prerequisites: a method of interprocess communication with an application and a method of uniquely identifying a widget in the application UI. A platform accessibility API supporting the querying and control of GUI widgets satisfies the first requirement. Unfortunately, most accessibility APIs lack a method of uniquely identifying all widgets without assistance from the original program author.

To overcome this problem, Clique relies on pseudo-unique identifiers for widgets. Such an identifier traces a path through the widget containment hierarchy (revisit Figure 2.2) starting at the root of a GUI window and ending at a particular widget. The identifier includes the index of the branch to follow at each level plus the role of the widget at that index as a sanity check. Clique scripts serialize these descriptions in an XPath-like notation (Clark and DeRose, 1999). For example, the path */client[3]/window[0]/editable text[3]* refers to the editable text area in the Windows XP Notepad program.

Pseudo-unique identifiers are brittle with respect to changes to the GUI structure at runtime and version differences. The former is solved by dynamically building identifiers at runtime in response to events. For example, if the Notepad program adds a toolbar above the editable text area at runtime, the index in the last segment of the path to the editable text area must be modified. A script can do this by noting the appearance of the toolbar and modifying the path appropriately. The second problem is not so easily solved, and requires heuristics determining the version of the GUI application so that the script can construct proper paths.

Thankfully, recent developments in accessibility APIs add support for better widget identification. Some APIs allow program authors to assign unique identifiers to widgets which may stay fixed within and across versions. Others assign unique identifiers to widgets automatically without the help of program authors. Neither solution is perfect, but both are an improvement over the existing path-based identifiers. The design of Clique allows for scripts to take advantage of either implementation, if and when their use becomes more widespread.

Bindings to Other Components

Clique scripts are not limited to binding model adapters to external widgets alone. The scripts may also define models which use public application APIs not tied to their user interfaces. For example, many of the functions of Microsoft Word may be automated using its Component Object Model (COM) interface. A Clique word processor script could mix models driving GUI widgets with models directly accessing the functionality of Word through COM.

Clique scripts may also define all of their business logic in-process. The Clique volume control application described in Section 5.5.1, for instance, has no GUI nor does it rely on an external program. The model for accessing the Clique sound mixer is coded directly into the script and makes the appropriate method calls to adjust the sound volume. A script writer could use the same approach to develop audio-only applications directly within Clique if desired.

Macros

Since the purpose of Clique is to adapt existing programs, its scripting framework provides support for executing *macros*, sequences of GUI widget manipulations that put the widgets into desired states. Many actions performed via an accessibility API are asynchronous, requiring the caller to listen for events to determine the result of an action.

Macros simplify this process by allowing scripts to perform actions on GUI widgets, define conditions for continuation of the macro sequence, and properly return control to the Clique input/output pipeline while that condition is not met.

As an example, consider the task of writing a new email in Microsoft Outlook Express. When the user selects this task from the Clique task menu, the Clique script executes a macro which triggers the GUI *New Message* menu item under the *File* menu, *New* submenu. The macro then returns control to Clique while waiting for the new message window to appear. During this waiting period, the task plays the waiting earcon and allows the user to navigate to other running tasks. When the window does finally appear, the macro indicates it is complete and provides a reference to the new window widget as the basis for all task and subtask models in the *Write Mail* task.

Note that a user does not hear reports of all the actions taken by a macro. A macro might click a GUI button, then select a menu item in a resulting pop-up, and finally click a button in a confirmation dialog all in the name of carrying out a single *Delete item* command given by the user. The report from the view of the subtask executing this command remains the same, no matter how many actions the underlying macro performs to produce the desired result.

Manual Program and Task Start

When a user selects an application from the program menu for the first time, Clique looks for and executes a special macro named *Main*. This macro is responsible for starting any required GUI application and getting a reference to its root widget, usually a GUI window. Clique then looks for a special list named *Tasks* in the script as the source of all the initial items to report in the task menu.

Once the user selects a task to start, Clique creates an instance of the task and allows it to run its startup macro. When the startup macro completes, Clique provides the final return value of the macro, typically a reference to a GUI window or dialog, to its associated task. The task then initializes its subtask interaction patterns, binds the patterns to models, binds the models to widgets within the associated window, makes the appropriate announcement of the new task, and activates the first available subtask. At this point, the new task appears as an additional item in the task menu.

In terms of the input/output pipeline, starting a program adds a *Container* interaction pattern to the bottom of the pipeline. That pattern immediately activates a *List* subtask pattern representing the task menu. When the user selects an option from the menu, the program pattern activates that task, deactivates any previously active task, and completes the task menu subtask.

Autotasks

In addition to the special *Tasks* list, a script may define a second list of tasks named *Autotasks*. When an event occurs in the GUI program associated with the script, Clique evaluates a trigger condition defined by each task in this list. If the trigger on any task evaluates to true, Clique automatically starts the task and makes it active. The purpose of this autotasks feature is to handle application initiated tasks with execution that cannot be predicted at design time.

For example, if the sending of an email times out, Microsoft Outlook Express shows and focuses a dialog indicating an error. The Clique script for this program must be aware of this dialog since it results in a change of application state affecting the Clique models for browsing mail, writing mail, and so on. The script must dismiss the dialog and reactivate the last focused widget before the user's task can continue. At the same time, the script must alert the user to the error.

To solve this problem, the script defines an autotask with a condition that detects the appearance of the error dialog. When the dialog appears, the error task instantiates subtasks for browsing the available error messages. When the user completes the task, the macro dismisses the error dialog and the last active task resumes.

Conditional Tasks and Subtasks

Not all tasks and subtasks are available at all times. Some tasks and subtasks have preconditions which must be met before the tasks and subtask are available for user interaction. Clique scripts may register preconditions for any defined task or subtask. The Clique framework evaluates these expressions before listing conditional tasks in the task menu or allowing navigation to conditional subtasks.

The email script, for instance, registers six conditional tasks dependent on various conditions in the *Browse Mail* task: *Reply to Sender*, *Forward Message*, *Move Message to Mailbox*, *Save One Attachment*, *Save All Attachments*, and *Resend Message*. The availability of the first three tasks depends on the user choosing a message in the messages list. The availability of the next three depend on the state of the chosen message, namely if it has one or more attachments and if previous delivery of the message failed.

Unlike autotasks, conditional tasks and subtasks are never launched automatically. Triggers for autotasks determine when a task should start in response to some system-driven event. Preconditions for tasks determine when those tasks appear in the task menu for the user to select. Preconditions for subtasks also state when the *Go to next subtask* and *Go to previous subtask* commands include those subtasks during navigation.

Task Completion and Cancellation

Tasks defined in a script state whether a user may complete or cancel them. In either case, the script may execute a macro finalizing the task in an external application. For instance, the completion macro for the new mail task clicks the send button in the new message window while the cancellation macro simply closes the window without saving the message.

Once a task has ended, Clique takes one of three actions. If the ending task explicitly specifies a *successor* task to activate, Clique activates that task. This feature is used extensively in the *Wizard* task pattern. If no inactive task remains, Clique returns control to the task menu in the current program. If inactive tasks exist, Clique reactivates the most recently active task, or the top-most focus space of the auxiliary stack in terms introduced in Chapter 3.

5.4 Satisfied Developer Requirements

In Chapter 2, I proposed methods I wished to further develop in order to improve the process of designing and implementing auditory displays. Table 5.12 summarizes the developer problems I planned to tackle with explanations of how the Clique design intends to solve them.

5.5 Implementation

I implemented a prototype of Clique in the Python (`http://www.python.org/`) programming language. Four Python packages contain the interaction pattern views and controllers, the model adapters, the output pipeline support, and the input pipeline support. Numerous classes comprise the major components of each of these packages including the output manager, the input manager, classes for each of the task patterns, and so forth. The prototype uses PyProtocols (`http://peak.telecommunity.com/PyProtocols.html`) to define interfaces, declare the interfaces provided by model classes, and to query interfaces on objects.

The application scripts are implemented as Python modules or packages discovered and loaded dynamically by a script manager component at run time. These modules contain classes inheriting from the predefined models, views, and controllers. Script modules also contain macro classes which perform actions on underlying applications via various platform APIs as required by the tasks defined for that application.

The model adapters and macros primarily rely on the Microsoft Active Accessibility (MSAA) API (`http://msdn2.microsoft.com/en-us/library/ms697707.aspx`) to

Problem	*Support*
Developers have difficulty building applications with support for auditory display from scratch because of the overwhelming demand for GUI development and limited visible use cases for auditory equivalents.	Developers may use Clique scripts to create bolt-on auditory displays for existing GUI applications.
Developers lack guidance for building usable bolt-on auditory displays. It is easier to build auditory displays that mimic their graphical counterparts.	Developers design their Clique scripts in terms of programs, tasks, and subtasks by specializing predefined interaction pattern components, designed to ensure consistency and usability across applications and tasks. Developers may extend any interaction pattern for use in new tasks and applications while retaining their basic usability characteristics. The Clique output pipeline constrains speech and sounds produced by scripts so that the resulting auditory scene is never unintelligible, and, without customizations, suitable for the listener.
Developers who wish to design applications with built-in support for audio have primitive tools for speech and sound output, and few options for supporting frameworks.	Developers may access high-level, Clique APIs for producing spatialized speech, auditory icon, earcon, environmental sound, and concurrent sound streams. Developers may build entire applications with built-in support for auditory displays in the same Clique script framework as their bolt-on solutions.

Table 5.12: Developer requirements satisfied by Clique.

read and manipulate GUI widgets. The Python wrapper for MSAA, called pyAA (`http://www.cs.unc.edu/Research/assist/doc/pyaa/`), allows models and macros to read labels, press buttons, select list items, and so on in other applications. This library is the main conduit Clique uses to automate actions on graphical interfaces while maintaining a user-facing auditory display composed of tasks. Note that while MSAA is used in the prototype, model adapters could be written for other platform APIs such as AT-SPI for GNOME (`http://live.gnome.org/GAP/PythonATSPI`), the Accessibility Protocol for Macintosh OS X (`http://developer.apple.com/documentation/Accessibility/Conceptual/AccessibilityMacOSX/OSXAXModel/chapter_4_section_1.html`), or User Interface Automation for Microsoft Windows Vista (`http://msdn2.microsoft.com/en-us/library/ms747327.aspx`). Table 5.13 lists the adapters implemented in the Clique prototype.

The pyHook library (`http://www.cs.unc.edu/Research/assist/doc/pyhook/`), a Python wrapper around the Windows hooking API (`http://msdn2.microsoft.com/en-us/library/ms632589(VS.85).aspx`), provides support for capturing all keyboard input for use by the interaction pattern controllers. The input manager alone is responsible for interfacing with pyHook.

The pyTTS library (`http://www.cs.unc.edu/Research/assist/doc/pytts/`), a wrapper around the Microsoft Speech API (`http://msdn2.microsoft.com/en-us/library/ms723627.aspx`), enables speech synthesis. The output manager component pipes all output from pyTTS through pySonic (`http://pysonic.sourceforge.net/`), a Python wrapper for the FMOD sound library (`http://www.fmod.org/`), to enable mixed, 3D sound and speech spatialization. pySonic is also responsible for playing non-speech earcons, auditory icons, and looping sound streams. The output manager, sound groups, and virtual assistant speakers are responsible for controlling pyTTS and pySonic.

I decided to use the concatenative AT&T Natural Voices engine for speech synthesis for my evaluations. This engine offers extremely natural sounding voice output at mod-

Adapter Name	*GUI Widget(s)*	*Interfaces Provided*
List	One column list box	Interactive, seekable, selectable, list
Editable list	One column list box	Interactive, seekable, selectable, list, deletable
Column list	Multicolumn list box	Interactive, seekable, sortable, selectable, list, detailable
Editable column list	Multicolumn list box	Interactive, seekable, sortable, selectable, list, detailable
Drop down list	Drop down list box	Interactive, seekable, searchable,list
Button list	Set of buttons or radio buttons	Interactive, seekable, searchable, list
Checkbox	Single checkbox widget	Interactive, seekable, list
Tree	Tree control	Interactive, seekable, searchable, strideable, tree
Text label	Static text label	Label
Text box	Uneditable text field	Interactive, searchable, text
Editable text box	Editable text field	Interactive, searchable, deletable, editable text
Simple document	Uneditable rich document	Interactive, searchable, text
Editable document	Editable rich document	Interactive, searchable, deletable, editable text
Hypertext document	Uneditable rich document with hyperlinks	Interactive, searchable, seekable, hypertext

Table 5.13: Clique adapters for common GUI widgets. The table includes the adapter name, the widget it adapts for use as a Clique model, and the interfaces provided.

erate speech rates, making it ideal for use in evaluations with both sighted and blind users. The engine can also use multiple concatenative speech databases, providing more than the number of distinct voices required by the Clique virtual assistant design.

I designed and recorded all of the earcons, all of the menu loops, and the majority of the auditory icons myself. Some of the auditory icons and all of the environmental sounds I downloaded from the Freesound repository (`http://freesound.iua.upf.edu/index.php`) under various Creative Commons licenses.

For more details about the implementation of the Python prototype, refer to Appendix B. Attribution for the Creative Commons licensed sounds is given in that Appendix as well as hyperlinks to the complete source code.

5.5.1 Example Applications

To prepare Clique for user evaluation, I implemented eight Clique scripts: five adapting GUI desktop applications of varying complexity, two adapting homegrown GUI questionnaire screens required by the evaluations, and one implementing an application entirely in Clique with no GUI counterpart. The following sections describe the tasks supported by these scripts. Again, refer to Appendix B for example source code from one of the scripts.

Notepad

The notepad script (Table 5.14) adapts the simple Windows Notepad editor to a task-based Clique experience. The initial task menu lists *Open Document*, to load a file from disk, and *New Document*, to start a blank document, as the only available tasks. After the user chooses a file in the *Open Document* task, the *Edit Document* task immediately starts. Similarly, choosing *New Document* from the task menu also starts the *Edit Document* task since the former inherits from the latter.

While editing a document, returning to the task menu reveals additional tasks including *Save File As*, *Save File* (once the file has been named), and the names of any other documents under editing in inactive tasks. The *Confirm Overwrite* task starts automatically if the user selects an existing file name when saving and needs to confirm his or her choice.

Task (Pattern)	*Subtasks (Patterns)*	*Special Properties*
Open Document (File Opening)	Files (Tree)	Successor is Edit Document
Edit Document (Form Fill)	Document (Text Editing)	No complete
New Document (Inherited)	Inherited	Subclass of Edit Document
Save File As (File Saving)	Folders (Tree), Filename (Text Editing)	
Save File (Run and Report)		
Confirm Overwrite (Form Fill)	Are You Sure? (List)	No cancel

Table 5.14: Tasks, subtasks, interaction patterns, and their properties in the notepad script for Windows Notepad. Tasks with *No complete* or *No cancel* properties cannot be completed or cancelled by the user respectively.

Web Browser

The Web browser script (Table 5.15) adapts some basic features of Mozilla Firefox for use in Clique. The task menu lists *Browse Bookmarks* and *Go to Address* upon program start. After choosing a bookmark or entering a URL, the *Browse Document* task becomes active. Within a document, the user may also elect to *Go Forward* or *Go Back* a page, depending on whether pages are available in either direction.

Navigation of the page elements proceeds in document order, with sounds indicating the roles of elements (e.g., list, table, heading, etc.). The Clique search commands function as expected and consider all text on the page. The Clique seeking commands change behavior slightly. Instead of navigating to the first letter of a particular word, pressing a single character key jumps to the next or previous document element having a role associated with that key. For instance, pressing *L* causes the reading position to jump to the next link. The user activates a link or interacts with an embedded subtask (e.g., text entry field) by giving the *Complete this* command. All images are replaced by their alternative text descriptions (i.e., HTML *alt* attribute values) when possible.

All of these Web browsing behaviors are common in existing screen readers and other auditory displays. Since these functions are not tied to the visual presentation of the document, but rather its semantic model, no changes are required to adapt them to the Clique user experience.

Task (Pattern)	*Subtasks (Patterns)*	*Special Properties*
Browse Bookmarks (Form Fill)	Bookmarks (List)	Successor is Browse Document
Go to Address (Form Fill)	URL (Text Entry)	Successor is Browse Document
Browse Document (Form Fill)	Page (Document Reading)	
Go Back (Run, Wait, and Report)		
Go Forward (Run, Wait, and Report)		

Table 5.15: Tasks, subtasks, interaction patterns, and their properties in the Web browser script for Mozilla Firefox.

Calendar

The calendar script (Table 5.16) adapts the Day By Day Professional (DbD) calendar program. The *Browse Calendar* task starts immediately when the program starts since it is the only task available initially. While browsing the calendar, the user can navigate dates by days, weeks, months, and years; browse summaries of the notes on a date; and read the full text of a note on a specific date. Activating the task menu while browsing the calendar lists the tasks *Create Note*, *Go to Today*, *Go to a Specific Date*, and *Search*

Calendar. The listing also includes the task *Edit Note* if the user has selected a note on a date.

The *Search Calendar* task allows the user to enter a set of keywords and then browse all dates with matching notes. This search feature is not tied to the Clique quasimodal search because Day by Day Professional limits the amount of information exposed in its GUI to notes appearing on a specific date. Therefore, the script must adapt the built-in DbD search feature to support searching across dates. It is possible that that calendar script could drive the DbD GUI in response to Clique quasimodal searches, but the responsiveness would be poor. For each letter typed, the script would have to close a previous search dialog, open the search dialog, execute the search, and then walk over those results.

The *No Results* autotask handles the appearance of a dialog informing the user that there are no search results. The task announces this information to the user and then immediately activates the previously active task without further (useless) confirmation.

Email

The email script (Table 5.17) adapts Microsoft Outlook Express for use in Clique. The task menu lists the *Browse Mail* task and the *Write Mail* task when the program first starts. The mail browsing task allows the user to browse mailboxes, browse messages in a mailbox, and read messages. Activating the task menu after selecting a message in a mailbox reveals additional tasks, namely *Reply to Sender*, *Forward Message*, and *Move Message to Mailbox*. If the message has one attachment, the task menu also includes an option for the *Save All Attachments*. If the message has multiple attachments, the menu lists both *Save All Attachments* and *Save One Attachment*. If Outlook Express previously failed to send the message, the menu also lists the *Resend Message* task as an option.

Task (Pattern)	*Subtasks (Patterns)*	*Special Properties*
Browse Calendar (Linked Browsing)	Date (Tree), Notes (List), Note Description (Text Reading)	No complete, no cancel
Create Note (Form Fill)	Note Description (Text Entry)	Modal
Edit Note (Inherited)	Inherited	Subclass of Create Note
Go to Today (Run and Report)		
Go to Date (Form Fill)	Date (Text Entry)	Modal
Search Calendar (Wizard)	Search String, Search Results	
Search String (Form Fill)	Keywords (Text Entry)	Modal
Search Results (Linked Browsing)	Dates (List), Notes (List)	Modal
No Results (Run, Wait, and Report)		Modal, no complete

Table 5.16: Tasks, subtasks, interaction patterns, and their properties in the calendar script for Day By Day Professional.

The message composition tasks support the entry of recipient email addresses, carbon copy addresses, subject, and message body. When any of these tasks are active, the task menu also lists *Attach Files* as an option. Once attachments exist, the user may browse and delete them in a subtask, just as he or she may edit the recipient, carbon copy, subject, and message body in subtasks.

Various autotasks handle the occurrence of warnings and errors. The *Confirm Delete From Trash* and *Confirm No Subject* tasks allow the user to continue or abort operations that will delete messages permanently or send messages without subjects. The *Send Error* task allows the user to browse all error messages about unset mail.

Archive Manager

The archive manager script (Table 5.18) makes the functions of WinZip available to Clique users. The task menu initially contains *New Archive* and *Open Archive* as options.

The *New Archive* task allows the user to name the new archive before starting the *Browse Archive* task. The *Open Archive* task, on the other hand, allows the user to select an existing archive on disk before activating the *Browse Archive* task. The browsing task allows the user to manipulate the contents of the archive file by adding, extracting, or deleting contained files.

Returning to the task menu while browsing an archive reveals the contextual *Add File*, *Add Folder*, *Extract File*, and *Extract All Files* tasks. Each of these tasks follows a wizard-like workflow. The user first completes a task selecting the origin of the file to add or destination of the file to extract. Then the user selects options for the process such as whether to extract files with the original paths intact, whether to encrypt files, whether to include hidden files when adding, and so forth. When any of the wizard tasks complete, they immediately start the *Wait While Processing* task which announces the start and completion of the operation.

One autotask handles the appearance of a special dialog indicating that an archive file already exists when attempting to create a new one with the same name. In this situation, WinZip opens the existing archive for editing. The *Open Dialog* task informs the user of this deviation.

Questionnaires

The questionnaire scripts adapt GUI versions of the Perceived Usefulness and Ease of Use (PUEU) (Davis, 1989) and NASA Task Load Index (NASA-TLX) (Hart and Staveland, 1988) surveys described in Chapter 6 and Appendix A. Both scripts have *Form Fill* task classes containing instructions for completing the survey, statements or questions that make up the body of the survey, and choices for responding to each item in the body. Within each task, the user first encounters a *Text Reading* subtask where he or she can browse the survey instructions. Following the instructions, *List* subtasks speak the statements or questions. The user either assigns a rating to the statement or chooses an answer to a question from the available choices in each list.

The questionnaire scripts exist solely for the purposes of the studies described in the next chapter. However, they provide further examples of GUI programs that may be retrofitted to fit the task-oriented model in Clique.

Volume Control

The volume control script provides a task for adjusting the volumes of all Clique output streams. The task has one *List* subtask for each assistant, looping, intermittent, and ambient sound stream. Each subtask lists percentages in steps of ten for adjusting the volume of its associated stream. For speech streams, the subtask speaks the volume percentage at the configured volume level. For sound streams, the subtask plays an appropriate sound at the chosen level and speaks the volume level for clarity.

The volume control script has no GUI counterpart. The business logic is implemented entirely in the script. A set of models manipulate the system volume API while providing the interfaces required by the *List* interaction pattern to enable user control and auditory display of the volume settings. Though simple, this script is an example of how new, audio-aware applications can be written from scratch and fit into the Clique user experience.

5.6 Perspective: Clique and Screen Readers

Having described the full user experience and architecture of Clique, it is worthwhile to put the design in perspective with respect to screen reading before continuing to the formal evaluations. The user experience in a generic screen reader is based on a narrator describing a single point of regard at a time on the visual display. The user interacts with a desktop application using its widgets and supported keyboard commands while the narrator reports the events occurring in the GUI (e.g., inserting characters, focusing on a different widget). The narration usually includes spoken details of graphical widgets such as their roles (e.g., check box, drop down), states (e.g., disabled), hotkeys (e.g., Ctrl-S), associated labels, and content, though the user may sometimes substitute sounds representing these features. The user also gives commands to the screen reader to move the point of regard independent of the application focus (e.g., spatial, hierarchical, or document review), perform mouse actions (e.g., right click), repeat or detail information about the point of regard (e.g., colors, text styles), and perform a host of other commands specific to the active program (e.g., read first unread email subject). A screen reader can enable interaction with most simple applications without a script, but usable access to complex applications (e.g., Web browsers) requires scripting or custom code in the screen reader core.

The user experience in Clique is based on a set of assistants that report information about concurrent tasks using both serial and simultaneous, speech and sound. The user issues universal commands to Clique in order to browse, search, or edit content in tasks; start, pause, resume, complete, or cancel tasks; repeat, detail, or memorize information; and adjust the volume and rate of output from assistants. Scripts binding reusable interaction patterns to adapters for GUI widgets define the tasks supported in any given application, in a manner that enables consistent interaction across applications. Only scripted applications are available for use with Clique, but as a limitation of the current prototype, not the design. (The possibility of using scripts when available and falling back on reading the screen for unscripted applications is an idea discussed more thoroughly in Chapter 7.)

5.7 Summary

The first half of this chapter described the Clique user experience. Six assistants positioned around the user in a virtual sound space use speech and sound to describe running tasks and subtasks. The assistants respond to various commands for task navigation, browsing, searching, editing, recording, and so on. Meanwhile, ambient and intermittent environmental sounds serve as reminders of the current working context. Together, these streams and inputs follow the view-stream display design described in the previous chapter.

The second half of the chapter explained the architecture of Clique. The input-output pipeline directs input messages to active tasks and subtasks, and dispatches output messages to appropriate groups and speakers. Interaction patterns define the commands supported by tasks and subtasks as well as the speech and sound they produce in response. Model adapters wrap existing GUI elements to provide interfaces required by common interaction patterns. This separation of models from views and controllers enables consistent interaction across adapted applications and reuse of components.

Two concluding sections of the chapter briefly stated the implementation details of the Clique prototype and eight application scripts. The last section put the design of Clique in perspective by describing its user experience and architecture alongside that of a typical screen reader. The next chapter evaluates the design and implementation of Clique using mathematical models, usability heuristics, and evaluations of the scripted applications by sighted and visually impaired users.

Task (Pattern)	*Subtasks (Patterns)*	*Special Properties*
Browse Mail (Linked Browsing)	Mailboxes (Tree), Messages (List), Email Body (Text Reading)	No complete, No cancel
Write Mail (Form Fill)	Recipient (Text Entry), Carbon Copies (Text Entry), Subject (Text Entry), Attachments (List), Message Body (Text Entry)	
Reply To Sender (Write Mail)	Inherited	Subclass of Write Mail
Forward Message (Write Mail)	Inherited	Subclass of Write Mail
Move Message to Mailbox (Form Fill)	Mailboxes (Tree)	Modal
Attach Files (File Opening)	Files (Tree)	Modal
Resend Message (Run and Report)		
Save One Attachment (Wizard)	Choose Attachment, Choose Folder	Modal
Choose Attachment (Form Fill)	File (List)	
Choose Folder (Folder Browsing)	Folders (Tree)	
Save All Attachments (Folder Browsing)	Folders (Tree)	Modal
Confirm No Subject (Form Fill)	Yes/No (List), Never Ask Again (List)	Modal, No cancel
Confirm Delete From Trash (Form Fill)	Yes/No (List)	Modal, No cancel
Send Error (Form Fill)	Errors (List)	Modal, No cancel

Table 5.17: Tasks, subtasks, interaction patterns, and their properties in the email script for Microsoft Outlook Express. Tasks marked *modal* prevent navigation to other tasks until the user completes or cancels it. The modal behavior is necessary because of limitations in the underlying Outlook Express software.

Task (Pattern)	*Subtasks (Patterns)*	*Special Properties*
New Archive (File Saving)	Folders (Tree), Filename (Text Editing)	Successor is Browse Archive
Open Archive (File Opening)	Files (Tree)	Successor is Browse Archive
Browse Archive (Form Fill)	Files (List)	No complete
Add File (Wizard)	Choose File to Add, Set File Options	Modal, Successor is Wait While Processing
Choose File to Add (File Opening)	Files (Tree)	Modal
Set File Options (Form Fill)	Store Full Path (List), Encrypt (List)	Modal
Add Folder (Wizard)	Choose Folder to Add, Set Folder Options	Modal, Successor is Wait While Processing
Choose Folder to Add (File Opening)	Files (Tree)	Modal
Set Folder Options (Inherited)	Inherited, Include Subfolders (List), Include Hidden (List)	Subclass of Set File Options
Extract File (Wizard)	Choose Extraction Folder, Set Extraction Options	Modal, Successor is Wait While Processing
Choose Extraction Folder (Folder Browsing)	Folders (Tree)	Modal
Set Extraction Options (Form Fill)	Overwrite Existing (List), Skip Older (List), Use Folder Names (List)	
Extract All Files (Inherited)	Inherited	Subclass of Extract File
Wait While Processing (Run, Wait, and Report)		
Open Dialog (Run, Wait, and Report)		Modal, no cancel

Table 5.18: Tasks, subtasks, interaction patterns, and their properties in the archive manager script for WinZip.

Chapter 6

Evaluating Clique

This chapter describes five evaluations of various aspects of the Clique software. The first section compares the concurrent active task streams in Clique with a single stream of speech commonly used by screen readers. The second section reports the results of a heuristic evaluation peformed on the peripheral streams in Clique. The third describes the methods, analysis, and results of a summative Clique evaluation by experienced users of the JAWS screen reader. The fourth describes another summative study testing the benefits of email access via Clique for sighted users temporarily removed from the graphical desktop environment. The final section analyzes the feasibility of adapting arbitrary GUI applications to audio using Clique.

6.1 Concurrent Streams Analysis

In Chapter 2, I explained that my primary reason for using concurrent streams in Clique was to offer faster access to utterances than is possible with a serial stream. As a first evaluation of Clique, I developed a mathematical analysis of how quickly, from the onset of the first utterance, a particular utterance plays to completion, an utterance repeats to completion, all utterances play to completion, and all utterance repeat to completion. I explored variants on the multi-channel design including cases where the number of channels equals the number of utterances, the number of channels is independent of the number of utterances, the number of channels and utterances matches the implementation of Clique, the ability to skip utterances is unavailable, and the ability to skip utterances is available.

When analyzing Clique as implemented, I developed exact times to hear certain utterances based on the properties of the active task streams (i.e., content voice and sound, summary voice and sound, narrator voice and sound) and variable utterance durations. For general analyses, I worked through proofs showing whether hearing one or more target utterances occurs sooner in a single-stream or a multi-stream design.

6.1.1 Global Assumptions

Throughout these calculations, I define an utterance as a speech or sound message conveying meaningful information when heard in isolation. My main assumption is that a user is able to listen to one and only one utterance among all those playing simultaneously across streams. The basis for this assumption comes from the theory of auditory scene analysis discussed in Chapter 4, that a listener may select and attend to only one perceptual stream at a time. The validity of this assumption only has bearing on my calculations if it turns out that the user is incapable of hearing even a single target. If the listener is able to hear more than one target among many concurrent utterances, these calculations come to represent the worst-case scenarios. See Section 6.3 for a further discussion of this assumption.

My other assumptions are less volatile and concern the design of the streams and utterances themselves. As recommended in Chapter 4 and implemented in Chapter 5, the onset of each stream in the multi-channel condition is delayed from the previous stream onset by at least 200ms. In my calculations, this delay appears as a constant term k. The offset of each stream from the start of the first utterance grows linearly according to $o_i = i \cdot k$ where $i = 0, 1, \ldots, n$ and n is the total number of concurrent utterances. Note that this definition indicates the offset increases with utterance, not channel, such that if one or more channels is silent, they do not contribute a delay to the next channel to play an utterance.

I also assume that the next utterance always starts before the previous one ends (i.e., successive utterances overlap somewhat in time). If d_i represents the duration of utterance i, then this implies $k < d_i$ for all $i = 0, 1, \ldots, n$. This assumption is valid for the current implementation of Clique since the 200 ms delay constant is less than nearly all utterances, except, perhaps, single letters at very high speech rates.

Finally, I assume that the order of utterance onsets is the same in both the single- and multi-stream designs. In other words, utterance u_{i-1} starts immediately before u_i in both designs for all $i = 0, 1, \ldots, n$.

6.1.2 To Play One Utterance

When a user wishes to hear only one utterance among many, a multi-stream design may reduce the delay a listener experiences waiting for a single-stream display to report the target utterance. The following sections state proofs of this benefit by showing a target utterance in various multi-stream designs always plays sooner than the equivalent utterance in a serial stream. Whether the user actually hears the desired utterance is answered later in this chapter (see Section 6.3).

N-utterances, N-channels

In this first model, I assume the number of utterances n is equivalent to the number of available channels that can sound simultaneously. What follows is a proof that any utterance u_i in this concurrent design will play to completion sooner than the same utterance in a serial stream, even when forced delays $o_i = k \cdot i$ exist in the multi-stream design.

It is important to note that the N-utterance by N-channel design assumed in this proof is *not* the one found in Clique. This proof is solely a precursor to the one described in the next section concerning the blocked-concurrency design used in Clique. While some readers may find the outcome of this N by N proof obvious by intuition, I include it for completeness.

For the multi-stream condition, let o_i be the offset from the start of all streams for utterance i with $o_0 = 0$, and let m_i be the time measured from the onset of the first utterance to play utterance i in its entirety. For the single-stream condition, let s_i be the time measured from the onset of the first utterance to play utterance i in its entirety. For both conditions, let d_i be the duration of utterance i.

Figures 6.1 and 6.2 provide visual representations of these definitions. Time appears on the horizontal axis. In the multi-stream representation, channels appear on the vertical axis.

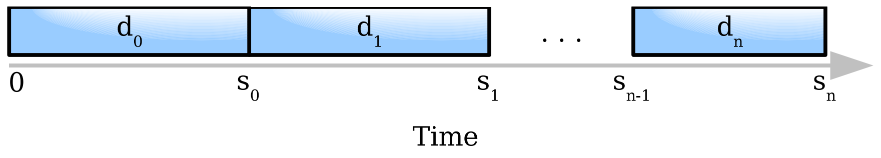


Figure 6.1: Depiction of a single stream of n utterances arranged serially.

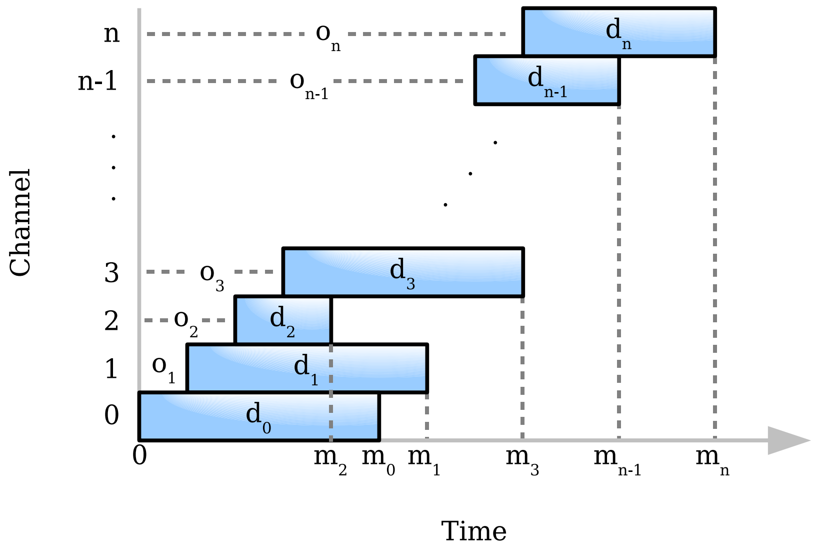


Figure 6.2: Depiction of n channels reporting n utterances arranged concurently and offset by a constant factor k.

Under these terms, the goal is to prove or disprove that $m_i < s_i$ for all $i = 1, \ldots, n$. By definition,

$$m_i = o_i + d_i \qquad \text{and} \qquad s_i = s_{i-1} + d_i$$

Solving the recurrence relation,

$$s_i = d_0 + d_1 + d_2 + \ldots + d_{i-1} + d_i$$

Subtracting the term m_i from both sides,

$$\begin{aligned} s_i - m_i &= d_0 + d_1 + d_2 + \ldots + d_{i-1} + d_i - m_i \\ s_i - m_i &= (d_0 + d_1 + d_2 + \ldots + d_{i-1}) + (d_i - m_i) \\ s_i - m_i &= (d_0 + d_1 + d_2 + \ldots + d_{i-1}) - o_i \qquad (6.1) \end{aligned}$$

The sign on the right hand side of the equation must be determined to complete the proof. In the global assumptions, I state that any concurrent utterance must start before the previous one in the order ends. Stated another way, the offset o_i for utterance i must be less than the ending time of the previous utterance, m_{i-1}. Written in the terms of this proof, $o_i < m_{i-1}$ for all $i = 0, 1, \ldots, n$. Using this information,

$$\begin{aligned} o_0 + o_1 + \ldots + o_{i-1} + o_i &< m_0 + m_1 + \ldots m_{i-2} + m_{i-1} \\ o_0 + o_1 + \ldots + o_{i-1} + o_i &< (d_0 + o_0) + (d_1 + o_1) + \ldots (d_{i-2} + o_{i-2}) + (d_{i-1} + o_{i-1}) \\ o_i &< d_0 + d_1 + \ldots + d_{i-2} + d_{i-1} \\ 0 &< (d_0 + d_1 + \ldots + d_{i-2} + d_{i-1}) - o_i \qquad (6.2) \end{aligned}$$

Given this result, the right hand side of Equation 6.1 must be positive. By equivalence, so must the left hand side. Therefore,

$$\begin{aligned} s_i - m_i &> 0 \\ s_i &> m_i \end{aligned}$$

This completes the proof that it takes less time to play any utterance in a multi-stream display having n utterances and channels than in the equivalent single-stream display for all utterances beyond the first.

N-utterances, C-channels in Blocks

The number of channels which may sound simultaneously without turning into an unintelligible cacophony is often much less than the number of utterances to be reported. In this second model, I assume that the number of channels c is less than the number of utterances n. Up to c utterances sound simultaneously with offsets as in the previous section. After the longest utterance in a block of c utterances completes, the process starts again for the next c utterances. Thus, the c channels report all n utterances in $\lceil n/c \rceil$ blocks. The following is a proof that any utterance in this concurrent design will play to completion sooner than the same utterance in a serial stream.

For the multi-stream condition, let $o_{i,j}$ be the offset in block i for utterance j measured from the start of the first of all utterances to play, $u_{0,0}$. Let $o_{0,0} = 0$. Let $m_{i,j}$ be the time measured from the onset of $u_{0,0}$ to play utterance j in block i in its entirety. For the single-stream condition, let $s_{i,j}$ be the time measured from the onset of the first utterance to play utterance $i \cdot c + j$ in its entirety. For both conditions, let $d_{i,j}$ be the duration of utterance j in block i, or, equivalently, the duration of utterance $(i \cdot c + j)$. Figure 6.3 is a visual depiction of the multi-stream design. The single-stream display is the same as in the previous proof.

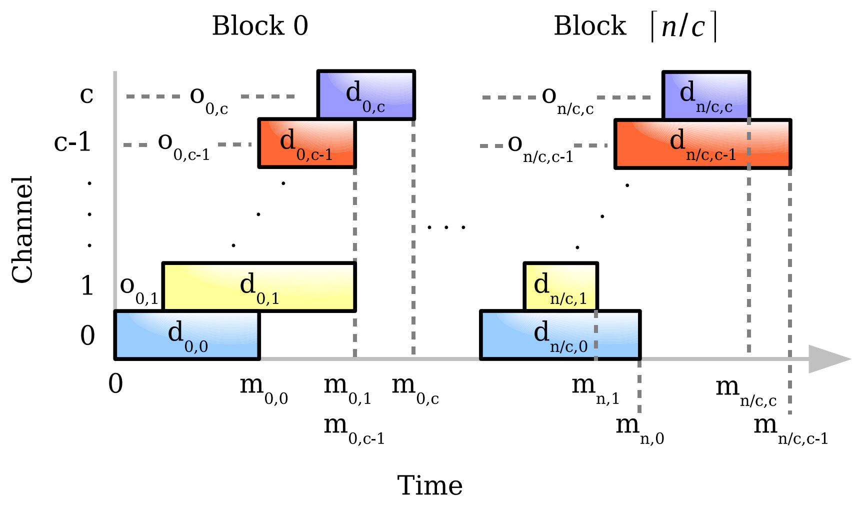


Figure 6.3: Depiction of c channels reporting n utterances arranged concurently in blocks and offset by a constant factor k.

Under these terms, the goal is to prove or disprove that $m_{i,j} < s_{i,j}$ for all utterances except the first. By definition,

$$m_{i,j} = \max((o_{i,0}+d_{i,0})+\ldots+(o_{i,j}+d_{i,j}))+m_{i-1,c} \qquad \text{and} \qquad s_{i,j} = d_{i,0}+\ldots+d_{i,j}+s_{i-1,c}$$

Solving both recurrence relations,

$$\begin{aligned} s_{i,j} &= (d_{0,0}+d_{0,1}+\ldots+d_{0,c})+(d_{1,0}+\ldots+d_{1,c})+\ldots+(d_{i,0}+\ldots+d_{i,j}) \\ m_{i,j} &= \max(o_{0,0}+d_{0,0},\ldots,o_{0,c}+d_{0,c})+ \\ &\quad \max(o_{1,0}+d_{1,0},\ldots,o_{1,c}+d_{1,c})+ \\ &\quad \ldots \\ &\quad \max(o_{i,0}+d_{i,0},\ldots,o_{i,j}+d_{i,j}) \end{aligned}$$

Subtracting the second equation above from the first,

$$s_{i,j}-m_{i,j} \;=\; d_{0,0}+\ldots+d_{i,j}-$$

$$\begin{aligned}&[\max(o_{0,0}+d_{0,0},\ldots,o_{0,c}+d_{0,c})\\&\max(o_{1,0}+d_{1,0},\ldots,o_{1,c}+d_{1,c})+\\&\ldots\\&\max(o_{i,0}+d_{i,0},\ldots,o_{i,j}+d_{i,j})]\end{aligned}\tag{6.3}$$

Again, the sign of the right hand side of the equation must be determined to complete the proof. Let $u_{i,\max}$ be the last utterance to complete in block i or any subset of block i up to and including utterance j. Let $d_{i,\max}$ be the duration of $u_{i,\max}$ and let $o_{i,\max}$ be its offset. Let $d_{i,\max-1}$ be the duration of the utterance started immediately prior to $u_{u,\max}$.

Using the relation developed in Equation 6.2,

$$\begin{aligned}o_{i,\max} &< d_{i,0}+d_{i,1}+\ldots+d_{i,\max-1}\\o_{i,\max}+d_{i,\max} &< d_{i,0}+d_{i,1}+\ldots+d_{i,\max-1}+d_{i,\max}\\o_{i,\max}+d_{i,\max} &< d_{i,0}+\ldots+d_{i,c} \qquad (6.4)\\ & \qquad (6.5)\end{aligned}$$

For ease of reference, let the left side of Equation 6.4 be p_i and the right side be q_i. Returning to Equation 6.3 and reordering terms,

$$s_{i,j}-m_{i,j}=(q_0-p_0)+(q_1-p_1)+\ldots+(q_{i,j}+p_{i,j})$$

From Equation 6.4, the right side of this equation must be positive. Therefore, the left must also be positive, yielding

$$\begin{aligned}s_{i,j}-m_{i,j} &> 0\\s_{i,j} &> m_{i,j}\end{aligned}$$

This completes the proof that it takes less time to play any utterance in a multi-stream display having c channels, n utterances, and $\lceil n/c \rceil$ blocks than in the equivalent single-stream display for all utterances beyond the first.

Exact Clique Times

The c-channel, n-utterance proof shows that the block multi-stream design in Clique guarantees utterances play sooner than in a single-stream design. Since the utterances and number of streams is known for Clique, it is possible to state exact times to play information while browsing, searching, editing, and changing task context. Referring to Figures 5.2 through 5.5 in Chapter 5, Table 6.1 shows the time to play each utterance in its entirety in terms of durations and offsets. The final row of the table shows another metric, the maximum time to utterance start (MTUS). This number states the maximum time a listener must wait for a particular utterance to become available as a target.

For comparsion, the exact time to play utterance i in a single stream design is $\sum_{j=0}^{i} d_j$. The MTUS in all cases for a single stream is $\sum_{j=0}^{n-1} d_j$.

	Browse	*Search*	*Edit*	*Change*
Action			$o_1 + d_{action}$	d_{action}
Task Name				d_{tname}
Subtask Name				$d_{tname} + d_{tsumm} + o_1 + d_{stname}$
Role				$d_{tname} + d_{tsumm} + o_1 + d_{role}$
Chunk	d_{chunk}	d_{chunk}	d_{chunk}	$d_{tname} + d_{tsumm} + d_{chunk}$
State	d_{state}	d_{state}	d_{state}	$d_{tname} + d_{tsumm} + d_{state}$
Task Summary				$d_{tname} + d_{task_summ}$
Subtask Summary	$o_1 + d_{stsumm}$	$o_1 + d_{stsumm}$		$d_{tname} + d_{tsumm} + o_1 + d_{stname} + d_{stsumm}$
Echo		$o_2 + d_{echo}$	$o_2 + d_{echo}$	
MTUS	o_1	o_2	o_2	$d_{tname} + d_{tsumm} + o_1 + d_{stname}$

Table 6.1: Exact times to play all active task utterances when browsing, searching, editing, and changing tasks and subtasks.

Revisiting An Utterance

A comparison of the time to revisit an utterance in a multi-stream versus single-stream display is also possible. I first assume that, after the initial announcement, the repetition of all utterances starts on a user key press, occurs serially for clarity, and offers no ability for the user to skip individual utterances. Let h be the time it takes the user to press the repeat key. In any multi-stream design, the total time it takes to repeat utterance i is $m_n + h + \sum_{j=0}^{i} d_j$ where m_n is the total time it takes for the multi-stream announcement to complete. With a single stream, the total time to repeat is $s_n + h + \sum_{j=0}^{i} d_j$. Since $m_n < s_n$ from the previous proofs, it takes less time to revisit an utterance in the multi-stream display than in the single-stream design.

Adding the ability to skip through utterances (e.g., hear only the beginning of an utterance before jumping to the next one) alters the calculation slightly. In any multi-stream design, the total time it takes to repeat utterance i with the ability to skip the prior i serial utterances is $m_n + \sum_{j=0}^{i} h + d_i$. In the single stream design, the time is $s_n + \sum_{j=0}^{i} h + d_i$. Once again, revisiting takes less time in the multi-stream case.

6.1.3 To Play All Utterances

The prior calculations show that when a user wants to hear one utterance out of many, a multi-stream design yields faster target access than a serial stream. But since auditory scene analysis predicts a listener may attend to no more than one stream at a time, a multi-stream design may not provide similar benefits when the user wishes to play all utterances.

In the multi-stream display, the total time to play all utterances is $m_n + h + \sum_{j=0}^{n} d_j$ where m_n is the total time it takes for the initial multi-stream announcement to complete and $\sum_{j=0}^{n} d_j$ is the time it takes for the user to play all utterances serially. With a single stream, the total time to play all utterances is s_n (i.e., no utterances overlap). Since

$s_n = \sum_{j=0}^{n} d_j$ by definition, it takes less time to play all utterances in the single-stream case than in the multi-stream case without interruptions.

To revist all utterances, an additional time of s_n is required. Since the same factor is added to both times, the single-stream report still remains shorter than the multi-stream case.

For simplicity, and to remain conservative, I assumed here that the user may attend to only one utterance during any multi-stream announcement, regardless of whether the report is chunked into blocks or not. As Section 6.3.4 shows, this assumption is probably too conservative as users can report information from concurrent streams with non-trivial accuracy. Furthermore, the comments by users with visual impairments in Section 6.3.6 indicate that most information reported by screen readers is often ignored while certain target utterances are sought.

6.2 Ambient Display Heuristics

In Chapter 2, I stated a second reason for using concurrent audio, namely to improve contextual awareness through the use of peripheral streams. To test the effectiveness of peripheral streams in Clique, I performed a heuristic evaluation designed for ambient displays. My goal in this study was to evaluate the usability of the peripheral streams alone.

6.2.1 Study Justification

I choose to perform a heuristic evaluation instead of another timing analysis or formal user study for two reasons. First, a mathematical comparison of when a user hears peripheral information in a concurrent versus serial design provides little insight into the benefits of peripheral streams. A multi-stream display can immediately announce peripheral information in a secondary stream whereas a single stream display cannot by definition (i.e., there is no periphery). A more formal comparison is moot.

Second, comparing the use of Clique versus JAWS when completing tasks requiring knowdge of peripheral events is biased in favor of Clique. As explained in Section 2.5.4 of Chapter 2, single stream screen readers must choose to ignore peripheral events, delay their announcement, or announce them immediately, with the latter two running the risk of being interrupted accidentally by the user. Any auditory display that is able to use at least one additional stream to announce peripheral events can easily overcome all of these problems by placing peripheral events in the second stream. Comparing the use of such a multi-stream display with a single stream one, again, reveals little useful information.

For this study, I assumed that the utility of peripheral streams was not in question. The fact that multiple streams can make additional information available for attention establishes their benefit over a single output stream. I chose instead to question the usability of the peripheral streams in Clique using a heuristic evaluation rather than a method of direct comparison.

6.2.2 Design

In this evaluation, I used Mankoff's heuristics for ambient information displays (Mankoff et al., 2003), an extension to Nielsen's ten original principles for user interface design (Nielsen and Mack, 1994). Using Clique, I executed browsing, searching, and editing tasks across the four program scripted for user evaluation (i.e., email, Web browser, archive manager, calendar) as well as the various menus in Clique. For each of Mankoff's criteria, I recorded problems in the peripheral auditory streams: related task speech, related task sound, unrelated task speech, unrelated task sound, ambient sound, and looping sound.

I did not recruit other people to act as evaluators. I analyzed Clique in light of the heuristics myself as a relatively quick and cheap method of spotting gross problems. Recognizing that my observations may be biased, I did not rely on the results of this study as the only test of Clique. For example, Section 6.3 describes an evaluation comparing Clique and JAWS on tasks that both systems support, not just those involving peripheral information readily available in Clique but not JAWS.

To summarize my findings, problems did exist, but none because of unexpected flaws in the design of Clique and none so major as to prevent evaluation by users. The sections below contain the details of my evaluation, labeled by the heuristics.

Sufficient Information Design

All peripheral streams provide minimal, but sufficient information to serve their purpose. The ambient, intermittent, and looping streams play sounds that identify their associated programs, tasks, and menus. The related and unrelated streams play sounds indicating the source of the event with accompanying speech summarizing the change. Clique collapses successive messages from a single source into one report announced when the user is not busy interacting with the system.

Identification of the exact source of a report from a related or unrelated task is not possible with the information given. Only a sound indicating the role of the source plays. If identification is required, a snippet of the intermittent task and ambient program sound associated with the event source should play simultaneously with the existing report. However, three sounds and speaking in the periphery is likely to mask or pull user attention away from the active task unnecessarily.

Consistent and Intuitive Mapping

Positions, volumes, voice timbres, voice rates, and sound durations of all peripheral streams are consistent across reports. Ambient and looping identifiers are consistent within and across instances of programs and menus. Intermittent sounds representing tasks are consistent within instances, but inconsistant across invocations of the same task. The inconsistency stems from difficulty determining what constitutes the same task, when information in the task varies widely across instances. For instance, a user might compose and send an email. Later, the user might open a draft of that exact same email and send it to another user. Determining that the task is one and the same is not simple for an external agent like Clique.

Users must learn the mappings between all non-speech sounds and the information they represent. A listener cannot deduce what program, menu, task, or role a sound represents without hearing it in context of the active task speech streams first. An intuitive relationship does exist between ambient program streams and intermittent task sounds, however, in the form of an environmental theme. That is, the ambient background representing a program is an environmental sound, and any intermittent task sounds for that program are commonly found in the paired environment. For instance, if the ambient sound is that of a rainstorm, intermittent sounds are thunder claps, splashing footsteps, wind blowing, and so forth.

The biggest problem facing consistency and intuition is the reuse of sound themes among programs and tasks running simultaneously. This is an unfortunate consequence of using a limited number of pre-recorded waveforms. Increasing the number of themes and task sounds within themes would reduce the occurence of this problem. Switching to sounds themes generated algorithmically would likely eliminate it altogether. For the purposes of the user evaluations, however, the five sound themes are sufficient to uniquely identify the four office applications and questionnaire programs.

Match Between System and Real World

The separation of sound sources into spatialized groups and the existence of ambient and intermittent sounds have strong ties to the real world. They are reminiscent of real-world environments where the ambiance of the locale fills the background, events cause sounds that “stand out”, and spoken interruptions come from people at the periphery. This everyday listening should be familiar to users who are responsible for learning the mappings between such common streams and information on the computer.

Visibility of State

Program identity, menu identity, and busy state identity are immediately audible because of the continuous, looping sounds representing them. Transitions among programs, menus, and busy/ready states are also noticable thanks to smooth volume fades between ambient, environmental sounds and the more jarring action sounds.

Sounds representing task identity are not audible at all times. These sounds are intermittent to prevent burdening the user with too many simultaneous streams. They also avoid blending the program and task sounds together such that the two bits of information cannot be separated. If task identity must be perceivable at all times, a scheme for layering a task-identifying ambient sound on top of the program-identifying sound should be developed.

Notifications in the related and unrelated streams are not always audible either. Messages in these streams only sound when the user is not busy giving input. Even then, successive messages in these streams from a given event source collapse into a single report. Nevertheless, any report is guaranteed to contain the most recent information, and the user may request the history of notifications at any time.

Aesthetic and Pleasing Design

All sounds come from natural environments, natural occurances, or follow Western musical structures. The ambient and intermittent sounds belong to themes, while earcons follow related rhythmic, melodic, and harmonic patterns. All voices are synthesized from high-quality, concatenative engines. The display is somewhat jarring and cacophonus when three or more of the perhipheral streams sound at once, but this is a rare occurence.

Useful and Relevant Information

The ambient stream identifies the active program at all times. The looping stream likewise identifies system menus. The related and unrelated streams report the most recent notifications from one or more changing tasks as soon as the there is a break in interaction with the user. The intermittent stream identifies the active task within a random window of time.

By their very definition, the unrelated streams may report information irrelevant to the user's current task. The user may choose to ignore or completely silence these streams if they repeatedly prove useless. The user may find the intermittent sounds useless too, or possibly even annoying, given that they repeat within a fixed time frame and are more jarring than the flowing ambient and looping streams. In this case, the user may have difficulty ignoring such sounds, and may elect to turn them off more readily.

Visibility of System Status

At no time is the state of the system invisible to the user. The presence of environmental or looping sounds informs the user that the system is ready for interaction. When the system is busy, and not ready for user input, the ambient stream plays a unique clock ticking sound. Upon hearing this sound, the user may choose to navigate away from this task and resume other work, or continue waiting. The related and unrelated streams report when a previously busy task becomes interactive again if it is no longer the active task. For example, assume the user launches a new task and then navigates away to another program while that task is busy loading. When the task finishes loading and is ready for interaction, the unrelated assistant speaks the name of the task and plays the task started earcon to inform the user of the change in the task state.

User Control and Freedom

The peripheral stream reports have no effect on user control over the active task. The user is free to browse, search, edit, and navigate tasks at any time, regardless of what the peripheral streams are currently reporting. If one or more peripheral streams are undesirable, the user has the ability to lower its volume or silence it completely. To the contrary, if a stream is more desirable, the user may raise its volume to make it more prominent. The process for adjusting volumes is difficult in the current implementation, but could be improved by providing more direct access to the volume controls.

The user lacks straightforward control over the sounds used in the periphery from within the Clique user interface. The sound themes are extensible, but require the addition of new sound files to a particular folder on disk. Making this process simpler is worthwhile, especially since user-selected sounds can reduce fatigue over longer periods of use (Barra et al., 2001).

Easy Transition to More In-Depth Information

The user may give the *Where am I?* command at any time to hear a spoken description of the current task and program. Clique plays the task and program sounds simultaneously with their spoken descriptions to reinforce the pairing. Likewise, the user may issue the *What was that?* command at any time to review the history of messages in the related and unrelated streams in reverse chronological order. Repeating this command causes Clique to jump back further in time without forcing the user to listen to an entire message in the history. A command that immediately activates the task or subtask that gave a peripheral report does not currently exist, but is a topic for future research (see Chapter 7).

Peripherality of Display

The ambient and looping streams are soft in volume and have few high-frequency changes. These properties help the peripheral streams avoid masking the active task streams and grabbing attention unnecessarily. These same properties may reduce listener sensitivity to these streams over time, however. The user may rely on the volume control to either make these streams more or less prominent depending on his or her needs.

Some of the intermittent sounds may be too jarring. Making the intermittent stream softer, or choosing sounds with less attack may correct this problem. Again, the volume controls may allow users to resolve the problem on an individual basis.

The related and unrelated streams are located in the literal periphery of the listener, to the left and right of the head. The speech from these streams may grab attention when they report, but the user may learn to ignore them. Their soft volume and offset positions support such learning.

Error Prevention

The user may issue the *Where am I?* and *What was that?* commands at any time to repeat missed, transitory reports. The system attempts to pair unique sound themes with programs, and keep those pairings for all future instances of a program. The finite number of sound sets, however, leads to redundant pairings. Since the system places a higher priority on consistency than on uniqueness between concurrent programs, it is possible for two running programs to have the same sound theme. This repeat usage of sounds can lead to errors in identification. The solution is to include more sound sets or make the user knowledgeable of this shortcoming so he or she can confirm the program name using the *Where am I?* command.

Flexibility and Efficiency of Use

The user has the ability to control the volume of all peripheral sources as well as to immediately silence the related and unrelated streams any time they are playing. The peripheral streams are otherwise rigid with respect to how, when, and where they sound. For instance, the user cannot relocate a source from the far left in spatial sound to the center nor indicate that the related stream should only sound when the message is from a particular task. Such flexibility in manipulating the peripheral streams might benefit advanced users, but use of the streams is not impeded by the current lack of configuration options.

Efficiency of the ambient and looping streams is high. The listener need only pay attention to these continuous streams to identify the active menu or possible program. Efficiency of identifying a task is relatively high, but not as effortless as identifying programs and menus. The listener must wait for an intermittent task sound to play. Finally, efficiency of hearing related and unrelated reports is high thanks to their paired speech and sound streams, and the *What was that?* command. The user may attend to these streams when expecting a report, or use the explicit command one or more times to quickly replay recent notifications.

6.3 Visually Impaired User Evaluation

The primary goal in developing Clique was to improve the usability of desktop applications for people with visual impairments over what they experience today using a standard screen reader. To evaluate whether this goal was met, I ran a user study involving the use of JAWS, the most successful commercial screen reader available today, by people with visual impairments performing routine tasks in four common applications: email client, Web browser, file manager, and calendar. My goal was to determine whether various novel features of Clique had a positive impact on user efficiency, learning (of new tasks), effectiveness, and satisfaction when compared with JAWS. Of the many possible research questions, I sought to answer the following:

1. The analysis in Section 6.1 demonstrates that access to a target segment of speech is faster when segments are presented concurrently rather than serially. But can users actually pick out a stream of interest from many presented simultaneously in Clique?

2. Does the ability to search using a set of universal commands in Clique decrease the amount of time it takes to locate information over using screen-based or application-specific search dialogs with JAWS?

3. Does structuring information in terms of tasks in Clique instead of individual widgets in JAWS improve the ability of users to extrapolate previous knowledge to unfamiliar applications?

4. How does the experience of completing email, Web browsing, scheduling, and file management tasks using Clique compare with that of using JAWS?

6.3.1 Study Justification

This study served two purposes, the first of which was to complement the mathematical models developed in Section 6.1. In that section, I showed that utterances play sooner in a multi-stream design than in a single stream display. The first task performed by users with visual impairments in this study tested whether users are able to take advantage of this speed-up by picking out an utterance of interest presented simultaneously with other non-target utterances. Furthermore, this task tested the assumption that users are able to hear and understand only one of many concurrent utterances by checking whether users hear non-target utterances as well.

The second reason for this study was to compare Clique and JAWS on equal footing: on tasks that both systems are designed to support. In Section 6.2, I established that use of multiple streams can solve the problem of presenting peripheral information in a multitasking environment better than a single stream display. If the same design is beneficial, equivalent, or detrimental to work performed without need for peripheral information remains to be seen. The second, third, and fourth tasks performed by users in this study were designed for this purpose.

6.3.2 Design

I met with each participant in two separate sessions. The first session consisted of training mixed with trials testing two specific aspects of Clique, namely concurrent speech and searching. The second session included two trials, one evaluating the use of Clique when learning about a new application and the other evaluating the use of Clique when interacting with multiple applications in concert. Throughout the sessions, participants worked with the Microsoft Outlook Express email client, the Firefox Web browser, the Winzip archive manager, and the Day by Day calendar program using both Clique and JAWS. The speech rate for both Clique and JAWS were fixed at roughly 175 words per minute, near the average for conversational American English speakers.

I measured performance using both interfaces in terms of accuracy, rate of completion, rate of recall, and so forth depending on the purpose of the trial. I used Davis' Perceived Usefulness and Ease of Use (PUEU) (Davis, 1989) questionnaire after each trial to collect subjective feedback about each participant's experience with Clique. I also administered the NASA Task Load Index (NASA-TLX) (Hart and Staveland, 1988) during the last trial to measure subjective workload for both Clique and JAWS.

To complete the PUEU and NASA-TLX questionnaires, participants used the Clique scripts I described in Chapter 5 along with the procedure and GUI software detailed in Appendix A.

Training

I taught each participant all of the Clique keyboard commands described in Chapter 5. After explaining each feature, I asked the participant to practice using it. For features requiring interaction with an application, participants interacted with sample data in Microsoft Outlook Express and Winzip. Along the way, I explained the purpose of all output from Clique including the role of the assistants, the goal of simultaneous output, and the meaning of all sounds.

The four trials described below were interspered with the training steps. After a participant received all of the prerequisite training for a trial, that trial was executed.

Trial 1: Concurrent Speech

In the first trial, I asked each participant to perform a number of listening tasks testing their ability to understand one of the four concurrent assistants active during a search in Clique: content, summary, narrator, and related (see Figure 5.4 in Chapter 5). I explained the purpose of the assistants during a search and the participant's goal with this text:

> In the following trial, you will hear a number of recordings. In the recordings, I am using Clique to search through messages in the email program. As I jump between messages, you will hear up to four voices speaking at once. Each voice tells you a particular kind of information about the current email.
>
> - The male voice directly in front of you reads the sender and subject of the message.
> - The female voice slightly left of center says the number of the current message in the mailbox and the total number of messages.
> - The female voice slightly right of center says the key I pressed to jump to the current message.

- The male voice to the far left speaks the length of the current message in seconds.

> Before I perform each search, I will ask you to listen for one of these six pieces of information. There will be a brief pause between the time I give you instructions and Clique begins speaking. Your primary goal is to listen for the information I request. After listening to the report from Clique, say the information I requested aloud. If you only heard part of the information, say so along with the part you heard. If you missed the information entirely, say that you missed it.
>
> If you manage to hear additional information from the other voices, please say it aloud next. However, do not sacrifice listening for the information I request in order to hear what the other speakers are saying.

I next played the participant sixty-three recordings of Clique speaking with two, three, or four voices after a search was performed. Each recording was prefaced with my voice asking the participant to listen for and report one of the following pieces of information: email sender, email subject, email number, total emails, search key pressed, and email length. The first twenty-one recordings were practice and participant responses were not noted. Responses to the next forty-two were logged, and covered all possible pairings of prefaces and number of active voices. The recordings were presented in random order across participants. I allowed participants to listen to each recording repeatedly during training, but only once during the measured trials.

For each trial recording, I counted the number of correctly identified, partially identified, and incorrectly identified primary targets. I noted similar counts for any secondary information heard.

The headers of the emails spoken in the recording were pieced together from random, but sensible information. The sender names were randomly created from a bank of

Anglo-Saxon first and last names. The message subjects were randomly drawn from a list of three word phrases having meaning as email titles (e.g., *Please send files*, *Update your password*, *Found lost keys*, etc.) The email index and totals were integers between one and one hundred under the constraint that index was less than or equal to the count. The search key names always matched the first letter of the message sender's first name. The email lengths were a random number between zero and fifty-nine followed by either the word minute or second. All recordings had the same continuous sound playing behind the voices to simulate the real user experience in Clique.

At the completion of this trial, I had each participant complete a copy of the PUEU questionnaire with respect to the concurrent speech features of Clique.

Trial 2: Searching

For the next trial, I asked each participant to perform a number of searches in Outlook Express and Winzip testing their efficiency in locating messages and files respectively. The participant performed the searches using Clique and JAWS given partial information about the target: message senders, message subjects, dates received, message body contents, file names, creation dates, file types, dates of modification, and cyclic redundancy check (CRC) numbers. I explained the activity to the participant as follows:

> Next I would like to see how well you can search for information using Clique and using JAWS. In the following trials, you will locate messages in an email program and files in a file manager. I will tell you something about the email or file I am seeking and you will attempt to locate it as quickly as possible. When you believe you have found the target, say "I'm done" aloud. If you have not found the correct target, I will tell you so you can continue your search.
>
> You will need to search through email headers, but you will not need to look in message bodies or other mailboxes. Likewise, you will not need to open

> and read any of the files, just search through their file names and details. You will search the emails and files separately, not at the same time. You will have five minutes to locate up to ten files and another five minutes to locate up to ten emails.

I asked the participant to first practice using one system (JAWS or Clique) to locate at least two targets in one application (Outlook Express or WinZip). After practicing, the participant next attempted to find ten additional targets with the same system and application pairing. As instructed, the participant worked for five minutes or until all ten targets were located. The participant repeated this process for all other display and application pairings using similar, but not equal, sets of practice and trial targets. The order in which the displays were tested, the pairing of message and file batches to a display, and the decision as to whether emails or files were searched first were all randomized across participants.

The corpus of one hundred emails, fifty per display, was generated as in Trial 1. The dates on the messages were randomly generated within a one year period. The set of one hundred files, fifty per display, was generated with filenames drawn from a list of common words, creation dates within the same one year period as the emails, random CRC values, and one of the following file types: text, spreadsheet, picture, sound, and movie. All searches performed could be successfully completed given some subset of these attributes.

At the completion of this trial, I had the participant complete a copy of the PUEU questionnaire with respect to the searching features of Clique.

Trial 3: Learning

At the start of the second session, I first gave the participant time to reacquaint him or herself with Clique and JAWS. I then asked each participant to explore an unfamiliar calendar application for a brief period of time and then explain how to complete three tasks using each auditory display. The instructions I gave were the following:

> Today I would like to see how well you learn to use an unfamiliar application via Clique and JAWS. In the following trials, you will learn how to perform three different tasks in a calendar application using each system in turn. The three tasks are the following:
>
> - Browsing the calendar by week
> - Creating a calendar entry
> - Deleting a calendar entry
>
> At the end of eight minutes, I will ask you to repeat, step-by-step, how you accomplish these tasks. Try to be as accurate in your description as possible, including what keys you must press or commands you give, and what you hear along the way.
>
> Here's a hint. When you are using JAWS, do not waste time browsing through the menu bar. You only need to use the controls in the program windows to complete the three tasks.

After giving these instructions, I presented the participant with an example of what I would do and what I would hear when completing a common task in an email program. I described this task both from the perspective of using JAWS and using Clique in exact detail to illustrate a perfect response.

I then gave the participant the allotted eight minutes to explore the calendar program using one of the auditory displays. At the end of eight minutes, I asked the participant how to complete each of the three tasks using that particular auditory display. After listening to the explanation, I had the participant repeat the exploration with the alternative display and report his or her findings. Both the order in which the participant used the displays and the order in which I asked about the three tasks were randomized across participants.

During the participant reports, I noted the number of correctly remembered input gestures required to complete each task. I maintained similar counts for output messages from the auditory display.

For this trial, I assumed that transfer of knowledge about the calendar application across trials with Clique and JAWS was negligible. Learning how to complete any of the three tasks with one display provided almost no insight into how to complete it using the alternative.

At the completion of this trial, I had the participant complete a copy of the PUEU questionnaire with respect to the experience learning to use a new program via Clique.

Trial 4: Workflow

As a final task, I had participant complete a number of tasks assigned by emails requiring interaction with all four applications: email client, Web browser, archive manager, and calendar.

> I would now like to see how you complete work requiring you to use multiple applications. Your goal is to read your email messages and complete the tasks they assign for fifteen minutes. Try to correctly complete as many tasks as possible during the allotted time. I do not expect you to be able to complete all seven tasks. You will use one system and then the other, and have access to all of the features of each.
>
> You must complete the tasks in the order given. Later tasks are harder then earlier ones. If you need help, tell me. But, in the interest of getting as much work done during the available time, refrain from asking questions unless you absolutely need assistance.

I gave the participant fifteen minutes to complete as much work as possible using one of the auditory displays. At the end of the first fifteen minutes, I asked the participant to complete the NASA-TLX questionnaire with respect to the auditory display he or she just finished using. I then had the participant repeat the trial using the second display and a second set of emails. Once again, I asked the participant to complete the NASA-TLX questionnaire with respect to the second display. The order in which the displays were tested and the pairing of a display to a batch of inbox emails were randomized across participants. All possible permutations were represented.

The order in which participants attempted the emails was fixed with tasks increasing in difficulty from beginning to end. Emails having the same index in the order were similar, but not the same, in both batches. For example, the fourth task in each batch asked the participant to locate information on a wiki Web page and email it to someone. The information sought and emailed differed between batches.

During the trial, I noted the total number of tasks completed using each system. I did not allow participants to proceed to the next task until they successfully completed the current one. The seven messages in the inbox folder assigned tasks following these themes:

1. delete a message
2. delete a message
3. reply with personal information
4. reply with information from an email in another mailbox
5. reply with information located on a wiki Web page
6. create a zip file from a folder and send it with a reply
7. modify calendar appointments

When using JAWS, the participant started the trial with focus on the first message in the inbox. When using Clique, he or she started on the first inbox message in the messages list subtask. In both cases, all of the other required applications were running in the background.

In addition to the inbox, the mail account contained a tree of other folders organizing previously received emails. The names of these folders stayed fixed across trials to avoid confusing the user, but their content changed to offset learning.

- inbox
- drafts
- outbox
- sent items
- deleted items
- projects
  - alpha
  - beta
  - gamma
  - delta
  - epsilon
- account requests
- meeting notes
- practice
  - one

- two
- three

The calendar application contained various appointments scheduled within a year of the evaluation. A folder named *documents* on the primary hard drive contained five folders storing batches of files. A set of local wiki Web pages listed contact information and provided navigation links to other local pages. These emails, appointments, files, and Web pages were essential to the completion of some of the email tasks.

At the completion of this entire trial, I had the participant complete a final copy of the PUEU questionnaire about their final experience using Clique.

6.3.3 Execution

Participants

I recruited eight participants with no usable vision from the Raleigh-Durham-Chapel Hill area. The ages of the participants were 13, 13, 25, 27, 30, 35, 48, 52. Six participants were male and two were female. All participants were experienced in the use of the JAWS screen reader, email, Web browsing, and file management. Seven of the participants had an onset of blindness at an early age while only one lost his vision later in life. Each participant spent four hours over two sessions completing the study. At their request, some participants completed both sessions on the same day with at least a two hour break between sessions.

Setting

Sessions with participants took place in Sitterson Hall on the UNC-Chapel Hill campus. All participants interacted with a computer running Windows XP, Clique, JAWS 7.0, Microsoft Outlook Express, Winzip, Firefox, Day by Day, and the PUEU and NASA-

TLX questionnaire programs. All participants used the same full-sized, unsplit keyboard for input and listened to 3D spatialized audio output through standard headphones.

6.3.4 Descriptive Statistics

Concurrent Speech

Figure 6.4 shows the average accuracy of all users reporting target utterances and non-target utterances. I scored completely correct reports as ones and partially correct, incorrect, and "unheard" responses as zeros. Accuracy was then computed as the sum of all scores divided by the highest possible score. For both targets and non-targets, the graph depicts the accuracy for all two, three, and four stream recordings as well as their average. Table 6.2 provides more details about these data.

In both data sets, accuracy was highest in two-stream recordings, lower in three-streams, and lowest in four. Cross-stream masking explains this result: utterances grew more difficult to discern in the presence of an increasing number of concurrent streams. The overall accuracy of reporting utterances was higher for targets than non-targets. Considering users were focused on locating one target among many non-targets, this result is logical. The number of non-targets accurately reported was far from zero, however. Users responded immediately after listening to the recordings, likely soon enough to attend to secondary streams still available in the temporary auditory image store.

Figure 6.5 shows the average accuracy of all users reporting targets and non-targets by utterance rather than stream. Table 6.3 provides more statistics.

Target accuracy was always better than non-target accuracy, consistent with the results reported by stream count. Within the target and non-target data sets, accuracy was highest when reporting the email subject and total number of emails. Both of these utterances appeared late in their respective speech streams, each after another

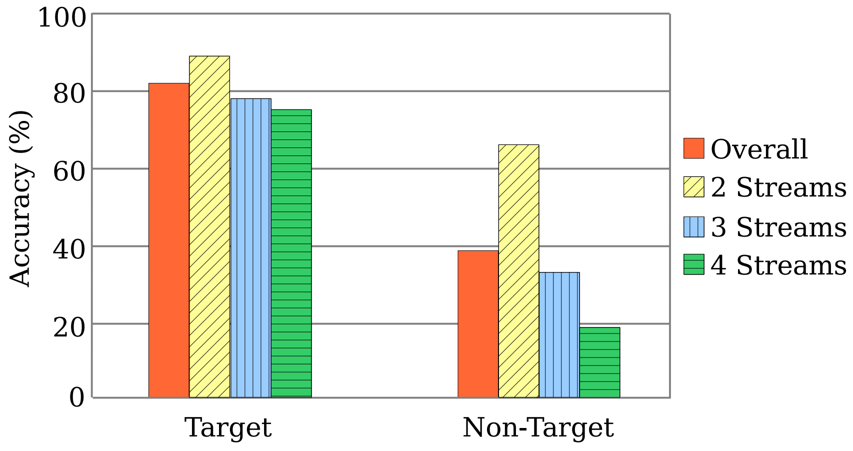


Figure 6.4: Accuracy of all users reporting a target utterance and non-target utterances with two, three, and four concurrent speech streams. The overall percentage for targets and non-targets is also reported. Higher is better.

utterance, and therefore overlapped less with other active streams. As a result, masking effects were likely lower for subject and total than for other utterances.

Users were least accurate when reporting the sender of a message as a target. Performance when listening for this utterance was poor because participants were unfamiliar with the names and the speech engine had difficulty pronouncing them well. For non-targets, participants had the least success hearing message length announcements. The related task assistant, located to the left of the user and speaking at a relatively low volume, was responsible for reading the lengths. These attributes likely contributed to the diminished length accuracy.

Searching

Figure 6.6 shows the average rate of completion of all users locating search targets by application. Items located within the given five minutes per application were scored as ones. Unlocated items were scored as zeros. The rate of completion was computed as the sum of scores divided by the maximum possible score (i.e., ten). For both the email

Target Accuracy (%)							
Streams	*Min*	*1st Q*	*Med*	*3rd Q*	*Max*	*Mean*	*Std Dev*
2 Streams	72.22%	77.78%	94.44%	95.83%	100.%	88.89%	11.11%
3 Streams	44.44%	66.67%	83.33%	90.28%	94.44%	77.78%	17.57%
4 Streams	50.00%	62.50%	75.00%	87.50%	100.%	75.00%	19.92%
Overall	61.90%	76.79%	83.33%	91.07%	95.24%	82.14%	11.38%
Non-Target Accuracy (%)							
2 Streams	31.58%	61.84%	68.42%	79.61%	81.58%	65.79%	17.40%
3 Streams	17.19%	25.00%	26.69%	41.80%	51.56%	33.40%	12.36%
4 Streams	10.00%	15.83%	16.67%	24.17%	26.67%	18.75%	6.16%
Overall	19.70%	35.42%	39.77%	45.27%	53.79%	39.39%	11.00%

Table 6.2: Descriptive statistics for target and non-target accuracy by number of concurrent streams. The columns are the number of streams, minimum, first quartile, median, third quartile, maximum, mean, and standard deviation.

and archive manager programs, the graph depicts the completion rate for both Clique and JAWS. Table 6.4 provides more details about these data.

In both applications, users fared better with Clique than with JAWS, more so in the archive manager than in the email client. The lack of a built-in search feature in the file manager and the visually-based search tools provided by JAWS account for the difference when searching files. For example, participants had to alternate between executing a search, scrolling the list of files, and switching between JAWS pointers in order to locate a desired file. In Clique, participants had to perform the relatively simpler process of entering a portion of the search string and navigating among the matches

In the email program, the rate of completion with JAWS increased dramatically. The built-in search feature provided by the email client was the source of this improvement. Participants used the application-specific search with JAWS to quickly locate emails based on subjects, senders, dates, and so forth. The Clique rate of completion remained practically the same as in the file manager. With Clique, participants used the same quasi-modal search utility in both applications, and therefore performed almost identically in both tasks. Still, the number of targets located in the email program was higher with Clique than with the built-in search feature, possibly because navigation to

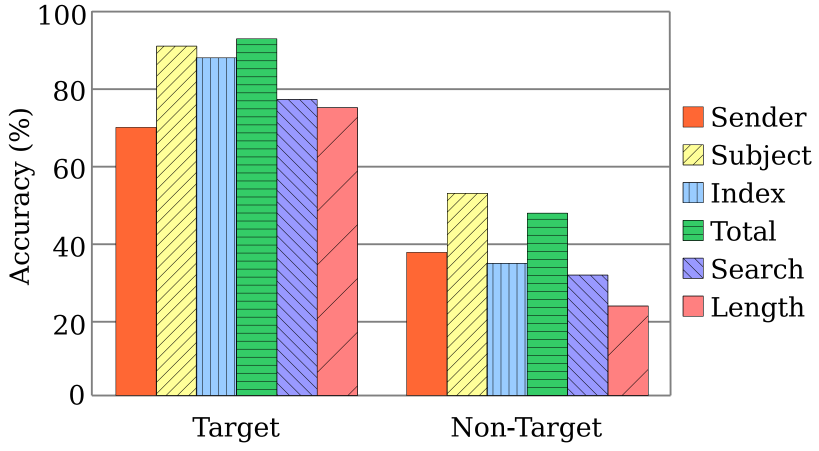


Figure 6.5: Accuracy of all users reporting a target utterance and non-target utterances. Higher is better.

and within the Microsoft Outlook Express search dialog required more time than using the quasi-modal search in Clique.

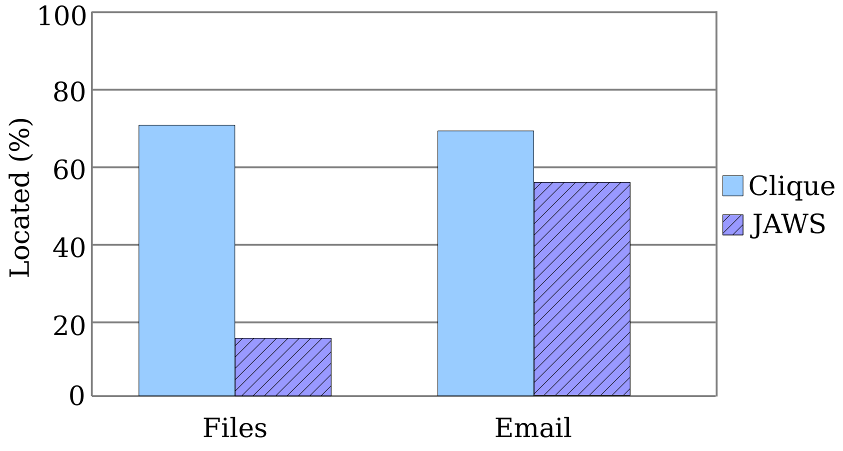


Figure 6.6: Percentage of search tasks successfully completed in the email and archive manager programs by system. Higher is better.

Learning

Figure 6.7 shows the average accuracy of all correctly recalled steps required to complete the browsing, creation, and deletion tasks in the calendar program. Input given to correctly advance a task toward completion was scored as ones while missing or incorrect

Target Accuracy (%)							
Utterance	*Min*	*1st Q*	*Med*	*3rd Q*	*Max*	*Mean*	*Std Dev*
Sender	14.29%	67.86%	71.43%	71.43%	100.%	69.64%	25.82%
Subject	71.43%	85.71%	92.86%	92.86%	100.%	91.07%	10.63%
Index	71.43%	85.71%	85.71%	85.71%	100.%	87.50%	9.16%
Total	85.71%	85.71%	92.86%	92.86%	100.%	92.86%	7.64%
Search	42.86%	64.29%	85.71%	85.71%	100.%	76.79%	22.83%
Length	28.57%	71.43%	71.43%	71.43%	100.%	75.00%	22.59%
Non-Target Accuracy (%)							
Sender	8.70%	33.70%	41.30%	47.83%	56.52%	37.50%	16.26%
Subject	30.43%	38.04%	52.17%	64.13%	82.61%	53.26%	18.56%
Index	21.74%	25.00%	32.61%	38.04%	56.52%	34.24%	12.39%
Total	30.43%	43.48%	45.65%	54.35%	65.22%	48.37%	11.01%
Search	20.00%	2.00%	35.00%	4.00%	45.00%	31.88%	10.33%
Length	5.00%	10.00%	27.50%	35.00%	40.00%	23.75%	13.56%

Table 6.3: Descriptive statistics for target and non-target accuracy by utterance. Columns are the same as in the previous table.

File Completion (%)							
Task	*Min*	*1st Q*	*Med*	*3rd Q*	*Max*	*Mean*	*Std Dev*
Clique	30.00%	35.00%	100.00%	100.00%	100.00%	71.43%	35.79%
JAWS	0.00%	5.00%	10.00%	20.00%	50.00%	15.71%	18.13%
Email Completion (%)							
Clique	20.00%	47.50%	70.00%	100.00%	100.00%	68.75%	30.44%
JAWS	20.00%	30.00%	60.00%	72.50%	100.00%	56.25%	28.25%

Table 6.4: Descriptive statistics for file and email search task completion by system and application.

input was scored as zeros. Utterances heard and reported by the user were counted as ones while incorrectly remembered or absent reports of output were noted as zeros. Input accuracy was computed as the sum of all input scores divided by the total possible inputs needed to successfully complete a task following any valid procedure. Output accuracy was calculated as the sum of all output scores divided by the total utterances made by a display when following any valid procedure. Overall accuracy was calculated as the sum of input and output scores over the total number of inputs and outputs. Table 6.5 provides additional statistics about these data.

The data show that accuracy of recalling input given was greater than accuracy of recalling output given by either system. Most users reported ignoring output not essential to completing the tasks, even though I asked them to repeat it. They also stated that such behavior was common practice during their daily interactions with a computer, not something specific to this trial.

For input, output, and total steps, accuracy in reporting was always greater for Clique than with JAWS. Better user recall of the Clique experience than the JAWS usage, fewer required inputs and given outputs in Clique than in JAWS, or a combination of both could have produced this result. Figure 6.8 suggests the second explanation is correct. Users actually reported more JAWS commands and output on average than Clique, but completing the same three tasks in JAWS required more inputs and gave more output than the same three tasks in Clique.

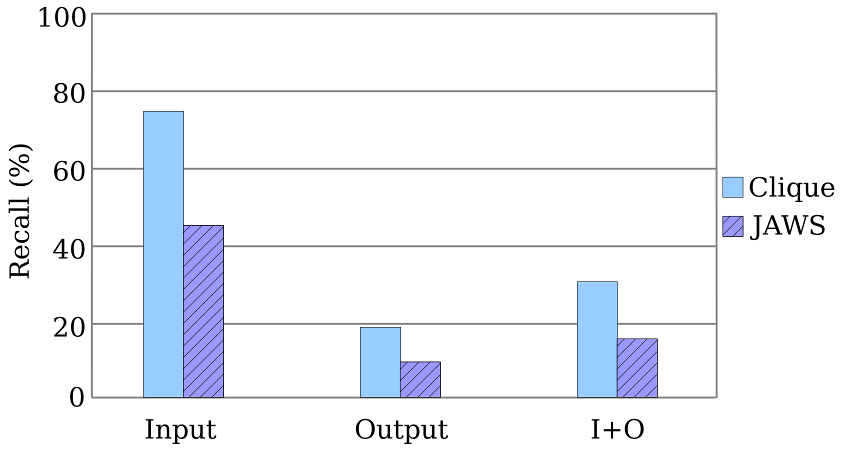


Figure 6.7: Accuracy of all users reporting input commands and system output after learning to complete three unfamiliar calendar tasks. The average accuracy of input and output is also reported. Higher is better.

Workflow

Figure 6.9 shows the average rate of task completion for all users performing tasks in the email, Web browser, archive manager, and calendar programs. Tasks wholly completed

Clique Total Accuracy (%)							
Task	*Min*	*1st Q*	*Med*	*3rd Q*	*Max*	*Mean*	*Std Dev*
Week	20.00%	40.00%	60.00%	60.00%	60.00%	50.00%	15.12%
Create	21.74%	26.09%	26.09%	27.17%	34.78%	27.17%	3.85%
Delete	0.00%	16.67%	22.22%	36.11%	55.56%	25.00%	19.47%
Clique Input Accuracy (%)							
Week	0.00%	50.00%	100.00%	100.00%	100.00%	75.00%	37.80%
Create	60.00%	80.00%	80.00%	85.00%	100.00%	82.50%	12.82%
Delete	0.00%	37.50%	100.00%	100.00%	100.00%	68.75%	45.81%
Clique Output Accuracy (%)							
Week	33.33%	33.33%	33.33%	33.33%	33.33%	33.33%	0.00%
Create	5.56%	11.11%	11.11%	12.50%	16.67%	11.81%	3.56%
Delete	0.00%	0.00%	7.14%	17.86%	42.86%	12.50%	16.09%
JAWS Total Accuracy (%)							
Week	11.11%	11.11%	15.56%	18.33%	28.89%	16.39%	6.16%
Create	10.64%	12.77%	15.96%	17.55%	21.28%	15.69%	3.59%
Delete	4.08%	13.27%	20.41%	24.49%	24.49%	17.86%	7.62%
JAWS Input Accuracy (%)							
Week	11.11%	11.11%	22.22%	44.44%	88.89%	31.94%	26.85%
Create	44.44%	55.56%	55.56%	55.56%	77.78%	56.94%	9.27%
Delete	11.11%	30.56%	44.44%	66.67%	88.89%	47.22%	25.72%
JAWS Output Accuracy (%)							
Week	8.33%	11.11%	11.11%	13.89%	19.44%	12.50%	3.32%
Create	2.63%	2.63%	5.26%	7.89%	13.16%	5.92%	3.65%
Delete	2.50%	8.75%	13.75%	15.00%	15.00%	11.25%	5.00%

Table 6.5: Descriptive statistics for input, output, and overall recall accuracy by system and task.

within the given fifteen minutes were counted as ones. Tasks partially completed or not attempted were counted as zeros. The rate of completion was computed as the sum of scores divided by the maximum possible score (i.e., seven).

Table 6.6 provides additional statistics about these data. The mean and median rates of completion were higher for JAWS than for Clique. The first and third quartiles were also higher for JAWS and Clique. The minimum, maximum, and standard deviation were exactly the same across systems.

Participants commented that years of experience with JAWS versus a few hours training with Clique helped them complete slightly more JAWS tasks in this trial. Many also

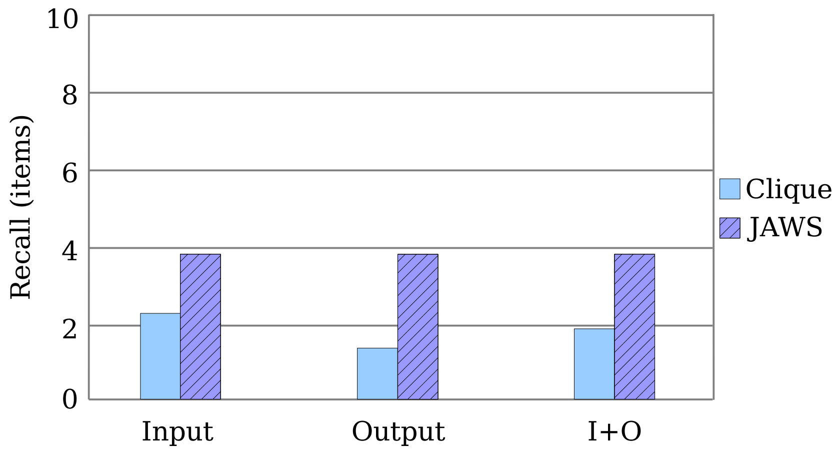


Figure 6.8: Average number of inputs and outputs recalled by users after learning to complete three unfamiliar calendar tasks using each auditory display system. The mean of input and output is also depicted. Higher is better.

said they could have done better with Clique given more training. Further observations of participant behavior and comments in this trial are found in Section 6.3.6.

Again, it is important to note that this trial tested Clique and JAWS on tasks that JAWS facilitates, namely work done in a foreground window with no important events in the periphery. Had peripheral information been important (e.g., alarms for appointments, instant messages, incoming email), there would be no fair basis for comparison: JAWS provides little or no output for such events outside the point of regard.

Task	*Min*	*1st Q*	*Med*	*3rd Q*	*Max*	*Mean*	*Std Dev*
Clique	42.86%	53.57%	57.14%	60.71%	100.00%	60.71%	18.31%
JAWS	42.86%	64.29%	71.43%	71.43%	100.00%	67.86%	18.31%

Table 6.6: Descriptive statistics for cross-application task completion.

Subjective Data

Figures 6.10 and 6.11 show the sums of the first six and last six ratings on the PUEU survey respectively for each of the four trials. The sums are depicted as percentages of total possible score where higher percentages mean better ease of use and usefulness.

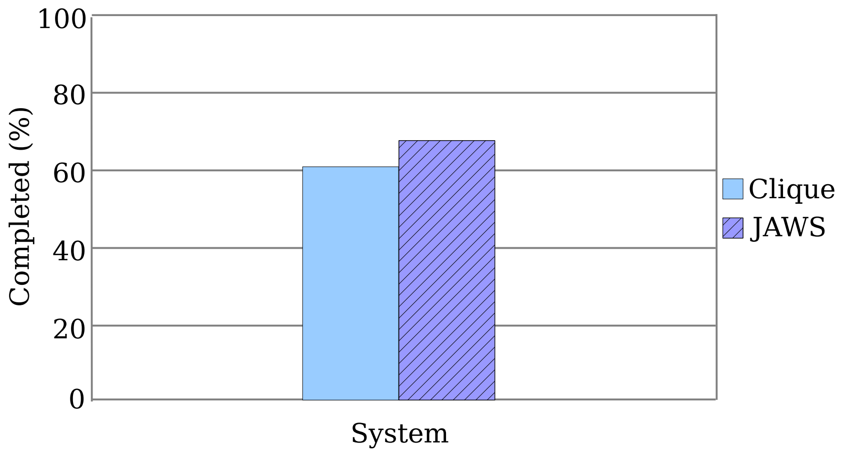


Figure 6.9: Percentage of office tasks successfully completed across applications by auditory display system.

All statements on the questionnaire were rated according to the participants experience using Clique up to and including the trial immediately proceeding the survey. The graphs show a close correspondence between ease of use and usefulness ratings across trials, with both metrics receiving their highest scores after the searching exercise and lowest after the concurrent speech trial.

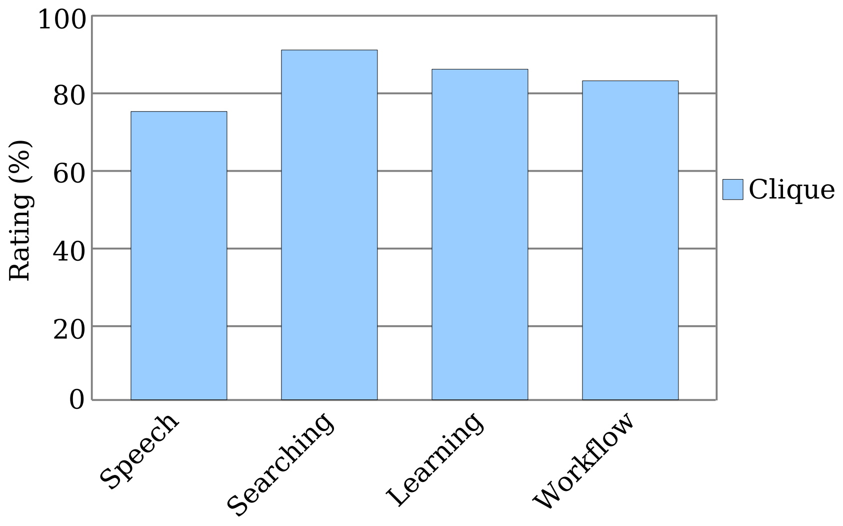


Figure 6.10: Summed usefulness ratings by trial as a percentage of total possible rating. Higher is better.

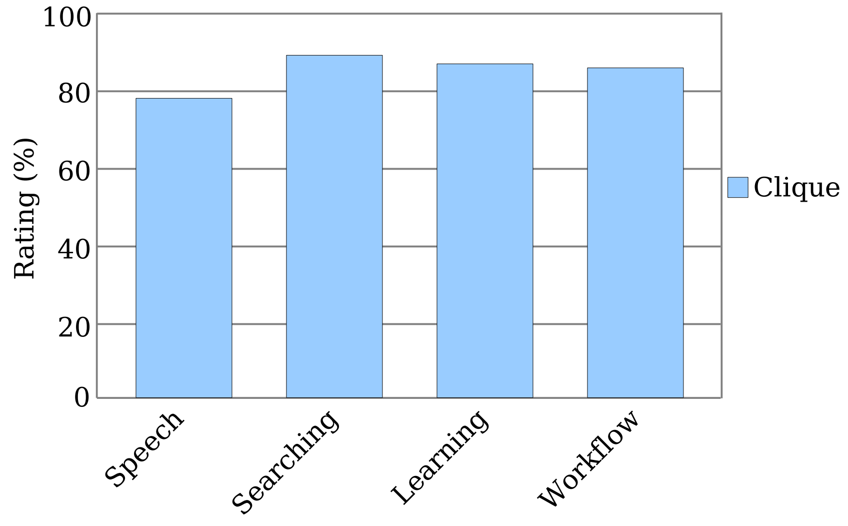


Figure 6.11: Summed ease of use ratings by trial as a percentage of total possible rating. Higher is better.

Table 6.7 gives statistics about the subjective workload scores computed using the NASA-TLX procedure for both Clique and JAWS during the workflow trial. The scores are shown as percentages of the maximum possible score where lower scores mean less workload. The first quartile, third quartile, and mean were lower for Clique. The minimum, median, and maximum were lower for JAWS.

System	*Min*	*1st Q*	*Med*	*3rd Q*	*Max*	*Mean*	*Std Dev*
Clique	15.33	38.92	49.50	64.75	94.67	51.63	24.36
JAWS	13.33	40.08	48.67	69.83	86.33	52.08	23.01

Table 6.7: Descriptive statistics for all blind user study workload scores. Lower is better.

Figure 6.12 shows the unweighted ratings of the six workload factors as percentages of total possible rating. For each factor, a lower percentage indicates less of a contribution to overall workload. Participants assigned higher workload scores to JAWS than Clique in terms of physical demand, perceived performance, effort invested, and frustration experienced. Clique received a higher mark for temporal demand. The two systems received the same rating for mental demand.

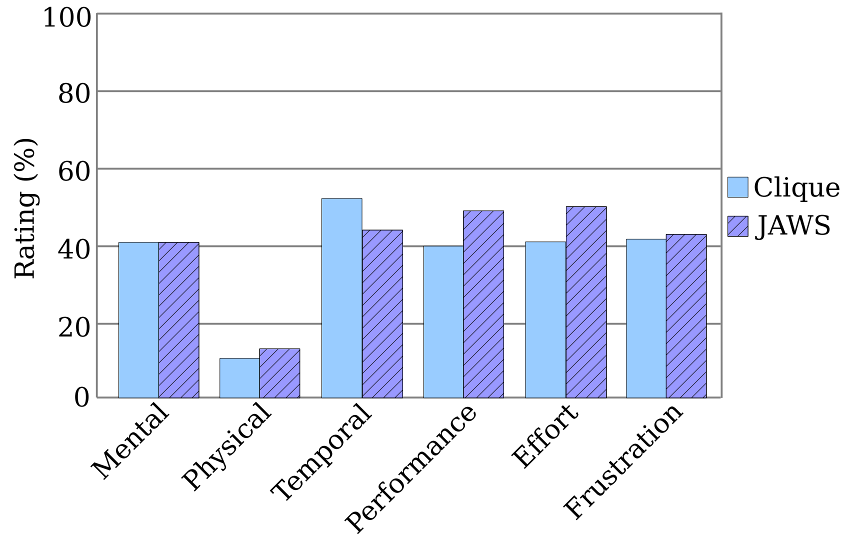


Figure 6.12: Summed workload factor ratings for Clique and JAWS as a percentage of total possible rating. Lower is better.

6.3.5 Statistical Inference

I performed hypothesis tests and computed correlations on the various data collected in this study. For all computations, I considered my results significant if they could occur by random chance in less than one in twenty samples under the null hypothesis ($\alpha = 0.05$). Because my small sample size yields low statistical power, I noted p-values close to my chosen critical value as "interesting," or potentially significant if retested with a larger sample. All paired tests were two-tailed since I had no prior assumptions about which condition would fare best.

Speech Target Accuracy

For the speech accuracy trial, I tested the null hypothesis that speech target accuracy was the same for two, three, and four stream recordings. The result of a logistic regression corrected for repeated measures shows no significant difference at my chosen level for α ($\chi^2(2) = 5.29, p = 0.0709$). Since this result was interesting, I performed contrasts among the individual conditions and obtained the results in Table 6.8.

Contrast	$\chi^2(1)$	*p-value*	*Description*
Two versus three	6.20	0.0128	Significant
Two versus four	5.24	0.0221	Not significant
Three versus four	0.09	0.767	Not significant

Table 6.8: Dpeech target accuracy p-values by stream.

A significant difference appears between accuracy in the two and three stream conditions after applying a Bonferonni correction ($\alpha/3 = 0.017$). An interesting difference shows between the two and four stream conditions, but not one that falls into my chosen critical region. These results suggest that further testing may reveal the following claims are true:

- The accuracy of identifying a target utterance is likely to be significantly greater with two concurrent streams than with three or four.
- The accuracy of identifying a target is likely the same with three or four concurrent streams.

I also tested the null hypothesis that speech target accuracy was the same for all possible utterances: sender, subject, index, total, search, and length. A logistic regression corrected for repeated measures reveals no significant difference ($\chi^2(5) = 6.83, p = 0.2334$). Since these results were far from interesting, I did not perform individual contrasts and accepted that target accuracy was not significantly affected by utterance type.

Speech Non-Target Accuracy

I performed similar tests for non-target accuracy by stream and utterance. I tested the null hypothesis that speech non-target accuracy was the same for two, three, and four stream recordings. A logistic regression corrected for repeated measures shows a significant difference for $\alpha = 0.05$ ($\chi^2(2) = 7.43, p = 0.0244$). As a follow-up I performed contrasts among the individual conditions and obtained the values in Table 6.9.

Contrast	$\chi^2(1)$	*p-value*	*Description*
Two versus three	39.95	< 0.0001	Significant
Two versus four	93.56	< 0.0001	Significant
Three versus four	170.72	< 0.0001	Significant

Table 6.9: Speech non-target accuracy p-values by stream

With a Bonferroni correction ($\alpha/3 = 0.017$), all three comparisons are significantly different. Therefore, I rejected the null hypothesis and accepted the following alternatives:

- The accuracy of identifying a non-target utterance is significantly higher with two streams than with three streams or four streams.
- The accuracy of identifying a non-target utterance is significantly higher with three streams than with four streams.

In addition, I tested the null hypothesis that speech non-target accuracy was the same across utterances. A logistic regression corrected for multiple measures reveals no significant difference ($\chi^2(5) = 6.94, p = 0.2254$). Based on this result, I accepted the hypothesis that utterance type did not affect non-target accuracy. Since the p-value was far from interesting, I did not perform individual contrasts.

Search Target Completion

For the searching trials, I tested two independent null hypotheses. First, I tested the null hypothesis that the search rate of completion for file targets in Winzip was the same for both Clique and JAWS. The result of a logistic regression corrected for multiple measures shows a significant difference at my chosen level for α ($\chi^2(1) = 37.89, p < 0.0001$). I rejected the null hypothesis and accepted the alternative that participants located significantly more files with Clique than with JAWS in a fixed amount of time.

Second, I tested the null hypothesis that search rate of completion for email targets in Microsoft Outlook Express was the same for Clique and JAWS. The result of another corrected logistic regression shows no significant difference at the $\alpha = 0.05$ level ($\chi^2(1) = 2.65, p = 0.1037$). In the case of emails, I accepted the null hypothesis that participants located a similiar number of messages using Clique and JAWS.

Learning Recall Accuracy

I performed further hypothesis tests concerning participant accuracy in recalling input commands and system output when learning to complete three tasks in the Day by Day Professional calendar program. For input, I tested the null hypothesis that the percentage of calendar inputs correctly recalled was the same when using Clique or JAWS. A logistic regression corrected for repeated measures shows a significant difference in recall accuracy ($\chi^2(1) = 14.49, p = 0.0001$). As such, I rejected the null hypothesis and accepted the alternative that users recalled a significantly higher percentage of inputs required to complete the three learned tasks when describing Clique than when describing JAWS.

For output, I checked the null hypothesis that the percentage of system utterances (e.g., labels, values, roles, key echos, etc.) correctly recalled was the same when using Clique or JAWS. Another corrected logistic regression shows a significant difference in output recall accuracy for my α ($\chi^2(1) = 9.36, p = 0.0022$). Therefore, I concluded that the null hypothesis was incorrect and favored the alternative, that users reported a significantly higher percentage of Clique utterances than JAWS utterances.

I performed one final check on the learning trial data by testing the null hypothesis that the overall percentage of correctly reported events (i.e., input plus output) was the same for both Clique and JAWS. The results of a logistic regression corrected for repeated measures shows a significant difference in overall recall accuracy ($\chi^2(1) = 33.66, p < 0.0001$). I rejected the null hypothesis and accepted the alternative that participants recalled a significantly higher percentage of events when describing their work with Clique than with JAWS.

Once again, it is important to note that these results stem from a large difference in the total number of possible input and output events participants could report in Clique versus JAWS. Revisit Section 6.3.4 for more information.

Workflow Task Completion

To compare user performance on completing cross-application tasks, I tested the null hypothesis that participants completed the same percentage of tasks when using Clique as when using JAWS. Another corrected logistic regression shows no significant difference in rate of task completion ($\chi^2(1) = 0.62, p = 0.4308$). As such, I accepted the null hypothesis that users completed practically the same number of tasks with both systems.

I emphasize again that these tasks did not require peripheral information to complete, and were purposely designed such that they could be completed with a reasonable amount of effort using Clique or JAWS. It is somewhat unsurprising that participants did equally well, statistically speaking, using both systems. However, Section 6.3.6 reveals additional information about the use of Clique and the measurement of performance in this trial for which any interpretation of this result should account.

Subjective Workload

For subjective workload, I tested the null hypothesis that the NASA-TLX weighted scores were the same for both Clique and JAWS. With my chosen α, the results of a paired t-test show no significant difference between mean workload for the two systems ($t(7) = -0.067, p = 0.949$). As a result, I accepted the null hypothesis that both systems had the same subjective workload.

Usefulness

After every trial, I asked participants to complete a copy of the PUEU questionnaire with respect to their most recent use of Clique. I computed the Pearson correlation coefficient for the summed usefulness ratings in each trial. Table 6.10 shows the values of the coefficients for each pair of trials, their p-values, and their significance at the $\alpha = 0.05$ level.

Question	ρ	*p-value*	*Description*
Speech, Search	0.723	0.043	Significant
Speech, Learn	0.500	0.207	Not significant
Speech, Workflow	0.514	0.192	Not significant
Search, Learn	0.213	0.612	Not significant
Search, Workflow	0.107	0.801	Not significant
Learn, Workflow	0.837	0.010	Significant

Table 6.10: PUEU usefulness p-values.

The data in this table identify two significant, positive correlations between ratings in the speech and search trials, and ratings in the learning and workflow trials. These results suggest that participants felt that the usefulness of the search tool in Clique was directly affected by the concurrent speech ability. Likewise, the usefulness of Clique in completing tasks across applications was related to how useful Clique was in learning to complete new tasks.

Ease of Use

Finally, I computed the Pearson correlation coefficient for the summed ease of use ratings in each trial. Table 6.11 shows the values of the coefficients for each pair of trials, their p-values, and their significance at the $\alpha = 0.05$ level. According to these results, only the search and workflow ratings showed a significant positive correlation. That is, participants felt the ease of working with multiple applications was related to how easy it was to locate information in those applications.

It is tempting to interpret this result as indicating that the search feature alone benefited participants working across programs. I do not believe the answer is quite so simple for two reasons. First, the calculations in the previous section revealed a positive

Question	ρ	*p-value*	*Description*
Speech, Search	0.480	0.229	Not significant
Speech, Learn	0.079	0.853	Not significant
Speech, Workflow	0.528	0.178	Not significant
Search, Learn	0.377	0.358	Not significant
Search, Workflow	0.792	0.019	Significant
Learn, Workflow	0.434	0.283	Not significant

Table 6.11: PUEU ease of use p-values.

correlation in subjective usefulness between the concurrent speech and search trials. The usefulness of the searching commands in various applications may be indirectly related to the design of the concurrent speech streams in Clique, not just the design of the search feature.

Second, though I have used the term *search trial* to refer to the second part of this study, nothing in the instructions for that trial forced participants to use only the search features of Clique, JAWS, or the underlying application. In fact, some tasks were completed more efficiently by simply browsing lists of files or emails, for example, after sorting them in a particular order. Participants did take advantage of these other methods, mixing browsing, searching, and ordering actions to complete tasks in the second trial. Therefore, it is better to interpret the positive correlation in ease of use as being between all methods of locating information in Clique, not the search commands alone, and the ability to work across applications.

6.3.6 Other Observations

After the concurrent speech trial, some participants commented that they would require more control over the degree of simultaneity when using Clique for real work. For instance, at least one participant had difficulty hearing the related task assistant and wished to increase its volume slightly. All users stated that they could get used to the concept of concurrent streams, though none freely projected how it would help them in their daily tasks.

A few users experienced difficulty entering numerals when searching. This unforeseen problem arose from forcing users to enter digits using the flat, upper row of the keyboard while holding the search key. The keypad, accessible entirely with one hand, remained bound to various Clique functions. While most other users reported no trouble locating proper numbers on the top row, reverting the keypad to entering digits with the search key pressed might improve the ease of searching in Clique.

In the workflow trial, much time was wasted navigating back to an inbox email to re-read forgotten instructions. For example, users read an email requesting information from a Web page, but forgot what they were seeking by the time they opened the Web browser and visited the proper page. This problem is exactly the one noted in the working memory section of Chapter 3. Unfortunately, no participant thought to use the Clique recording commands and memory menu designed specifically to overcome memory problems. When asked why they did not take advantage of the memory features, participants said they forgot about their existence. Better training on the use of this critical tool or a trial directly evaluating its benefits may have increased its use, and perhaps participant performance, during the workflow trial[1].

The data for the workflow trial did not capture partial task completions. In fact, of the five participants who completed exactly one more task with JAWS than with Clique, all five of them were nearly done with their final Clique task when time expired. When using JAWS, these same five participants had just started their next task when time expired. In other words, though the analysis shows more tasks completed with JAWS than with Clique, given an extra ten seconds or so, the number of tasks completed would be equal for all participants.

[1]While I had prepared a memory tool trial for this user study, I decided to remove it in the interest of keeping the study to two sessions and four hours per participant. Instead, I established the effectiveness of the memory tools in the sighted user study described in Section 6.4.

To aid data collection in the learning trial, I made audio recordings of the spoken descriptions participants offered for the three calendar tasks they learned to complete. Knowing that I was recording, some participants took the opportunity to make additional comments about Clique and JAWS. The following are some interesting quotes, corrected only for grammar.

> *00B - Comments about learning the calendar with JAWS*
>
> They had all of the tasks. But my only problem is remembering what does what. And that would come with working with it. ... It is better, I know I'm not explaining this well, but it's, the problem is that you've got all these keys and it's my first time at bat here.
>
> I know that sounds confusing, but, that [laughs] I mean, JAWS sucks. And that's just it. I hope you got that part.
>
> *00D - Comments about output from JAWS*
>
> I really don't pay attention to what [JAWS] is saying when I'm using an application. I just use it, figure out what I have to do to it, and be done with it.
>
> *00F - Comments about output from JAWS*
>
> I can't remember all the things JAWS says. I really can't. I kind of ignore them.
>
> *00G - Comments about output from Clique*
>
> I mean, I know the voices come in and do what they're [laughs] supposed to do, but [laughs] I'm so used to just using them as tools and remembering the keys.
>
> *00G - Comments about learning new applications with Clique versus JAWS*
>
> Just the part about not believing this is the same program from two different screen readers. [laughs]

> This just shows that when you throw a new program at someone, that it looks completely different. The experience is, more painful with JAWS which is the program I'm more familiar with. You know, it's like. OK? I mean it's telling me really what's there, but it's not telling me what I need to know about what's there. [laughs] You know?

At the end of the study, all eight participants asked if and when Clique would be available to them for personal use. Two participants expressed the opinion that the task-based nature of Clique would be best suited to specialized users and environments. One user said she felt Clique would be a good interface for inexperienced computer users, especially children with visual impairments in classroom settings. The other stated he believed blind call center operators who use only one or two applications in their work would benefit from the task-driven interaction. He also went on to say that he personally would not be able to rely on a task-based solution alone in his daily computer use. He explained that, being the computer guru in the family, he often had to assist sighted relatives in using their software and needed to know what was on the screen.

I agree with participants that the evaluated version of Clique only enables the use of a small set of applications, and then in a manner not conducive to collaboration in terms of widgets. But these are shortcomings of Clique *as implemented*, not *as designed*. A shrink-wrapped, consumer version of Clique could certainly support both task-based interaction as well as screen reading depending on the current needs of the user. My research did not focus on such a hybrid because screen reading is already well understood, whereas auditory, task-based computing is not. User comments relegating Clique to special-purpose uses must be considered in light of this intentional limitation and the prototype they tested.

6.3.7 Study Conclusions

Users appear to be adept at picking out desired information in one stream of speech among many. The number of concurrent streams directly affects their performance on this task, with accuracy significantly higher with two active streams rather than three or four. The type of utterance sought does not have a significant impact overall, though users seem to fare better with utterances that appear later in a given stream. Together with the timing analysis performed in Section 6.1, these results suggest users are able to hear information faster when presented in parallel than when presented serially.

Even while users are listening for a specific utterance, their ability to hear and report non-target utterances is non-trivial. The number of concurrent streams also has an impact on non-target accuracy making performance with two streams significantly better than with three or four, and performance with three streams significantly better than four. Again, the type of utterance sought does not have a significant impact overall, but user accuracy seems to be higher for utterances appearing later in a stream. These findings indicate concurrent streams may actually increase the output bandwidth over what is possible with a single stream.

The Clique quasi-modal search feature seems to assist users greatly in locating information in applications without built-in search dialogs (i.e., those that rely on fast visual scanning). The Clique search capabilities significantly increase the number of targets located in a fixed period of time over what is possible using the visual JAWS search tool. In applications providing their own search dialogs, the Clique search facility appears to provide similar benefits, but is not significantly better than using the application-provided search through JAWS.

Users appear to provide more accurate descriptions about how to complete tasks learned using Clique than learned using JAWS. In fact, users correctly report a significantly higher percentage of input commands to and output from Clique than they do with JAWS. This result is not necessarily due to more memorable Clique input and output, but more likely a consequence of fewer interactions required to complete tasks in Clique than in JAWS.

When using multiple applications to complete work, users perform equally well using Clique and JAWS. Likewise, subjective user ratings of workload are nearly the same for Clique and JAWS. Although Clique appears not to benefit users when working across applications, this result must be considered in light of the large difference in user experience with JAWS and Clique, the lack of use of critical Clique memory features during the workflow exercise, and the all or nothing measurement of task completion. Most users stated the belief that they could have completed more work with Clique given more training. The fact that users achieved similar results after only a few hours experience with Clique as years of experience with JAWS is noteworthy.

Subjective user ratings suggest correlations between Clique features in terms of usefulness and ease of use. A significant positive correlation between the ease of use ratings for the search and workflow trials indicates the Clique ease of using the search feature is important to the ease of completing work across programs. A significant correlation between the usefulness ratings in the speech and search trials suggests that the existence of concurrent speech affects the usefulness of the search tool. A final, significant correlation between the learning and workflow ratings shows that the usefulness of Clique in learning new applications impacts the usefulness of Clique in working across applications.

6.4 Sighted User Evaluation

In the conclusion of Chapter 2, I suggested that moving away from literal screen reading would make the auditory displays used by people with visual impairments more useful and attractive to sighted users in certain situations. To test this claim, I conducted a user study involving sighted participants temporarily removed from an environment conducive to visual computing. My goal was to determine what effect, if any, use of Clique during a brief stint away from a graphical desktop might have on a user's efficiency upon his or her return. The specific research questions I sought to answer were the following:

1. Does using an auditory display offer some benefit to worker efficiency over doing nothing at all during a brief hiatus from a computer with a graphical display?
2. What features must an auditory display offer to confer some benefit? Two conditions of interest are the following:
    (a) Minimal interaction. The user listens passively as information is read, and takes basic action, namely bookmarking important segments and deleting unimportant items.
    (b) Full interaction. The user both consumes and produces information by locating existing information and composing new content.
3. How does the subjective workload compare among the two levels of auditory display?
4. How do the subjective ease-of-use and usefulness compare among the two levels?

6.4.1 Study Justification

The research noted in Chapter 2 includes evaluations of techniques for mobile auditory display (Brewster et al., 2003) and systems supporting workers during long-term breaks in contact with desktop computers (Yankelovich et al., 1995; Ly and Schmandt, 1994). Two additional studies mentioned have explored the interaction between recording audio while mobile and making a transcription available later on a standard workstation (Hayes et al., 2004; Vemuri et al., 2004). No known research, however, has addressed questions similar to those above, and certainly not with an auditory agent like Clique.

Brief disconnects from the graphical desktop environment are common today for mobile office professionals. Computers with graphical displays are abundant, and many people have access to them, but certain temporary circumstances prohibit their use. For instance, consider the following users:

1. the IT support professional who must both check email for new support requests and walk among rooms in an office complex to service computers

2. the employee who rides cramped, light commuter rail for thirty minutes every morning on his or her way to work

3. the student hustling to his or her next class on a college campus

4. the architect and foreman who must sometimes meet to work outside in the bright sunlight

Each of these people might benefit from temporary auditory access to a computer while away from their primary workstation. This study explores how much benefit users derive from an auditory display functioning at varying levels of interaction.

6.4.2 Design

Throughout the study, participants interacted with the Microsoft Outlook Express email program via Clique and via its native GUI in order to complete common email tasks. When using Clique, the speech rate was fixed at 140 words per minute, slower than conversational American English to account for the lack of experience listening to synthesized speech.

I recorded two times as objective measures of efficiency. First, I observed the *total interaction time*, or the total time the user spent working with the computer. Second, I noted the *time of completion*, or how soon a user completed all tasks from a common starting point. I also used two surveys after each auditory trial to collect subjective feedback on the ease of use, usefulness, and workload of Clique.

Training

Before learning about Clique, each participant completed fifteen practice tasks using Outlook Express via its graphical user interface. This training familiarized participants with the layout of its GUI, the structure of the mailboxes used in the trials, and the kinds of tasks assigned. It also provided me with the chance to instruct participants on the fastest way to complete certain tasks. At the end of this training, I assumed all participants shared the same level of expertise at using the Outlook Express GUI.

Before each trial, participants received training on the Clique features required for the upcoming trial. For instance, before the trial involving minimal interaction with Clique, I instructed participants on the use of the highlighting and deletion commands, and the output they would encounter. Because of randomization, some participants received training on all features of Clique before the first trial and required no later training. By the end of the study, all users had received training equivalent to that provided in the visually impaired user evaluation discussed in Section 6.3, with two exceptions. First, I did not introduce the participants in this study to the main program menu. All tasks were completed in the email program alone.

Second, participants learned that using the Clique *Start Recording* and *Stop Recording* commands would not only record audio snippets of text, but also highlight recorded text segments in the visual display. Participants were taught to use this feature to mark important sections of email when using Clique so that they could quickly review critical information when later viewing it through the email GUI.

Trials

The trials followed a repeated measures design with all participants completing email tasks under three conditions. In the two experimental conditions, participants started their assigned work using Clique for a maximum period of ten minutes before switching to the GUI to complete it. In the control condition, participants used the GUI alone to complete all assigned tasks. The availability of Clique features served as the independent variable while time was the dependent variable. All permutations of trials were represented at least twice to counterbalance ordering effects. The three levels of audio display availability tested were the following:

1. No auditory display. The baseline condition in which the auditory display was not available. Participants completed all work using the GUI.
2. Minimal auditory display. The Clique *Start Recording*, *Stop Recording*, and *Delete This* commands were available. All output was given.
3. Full auditory display. All of the Clique commands were available. All output was given.

Participants were not required to use Clique for the full ten minutes. For instance, in the minimal condition, the participant could quit working with Clique after hearing all of the messages read aloud at least once. On the other hand, the same participant could choose to use Clique for the full ten minutes in the full condition. The measure of the total interaction time captured this decision.

Ten emails located in the inbox at the start of each trial assigned the required tasks. All tasks were variations on a few basic themes, but the exact content of the messages varied across trials to offset a learning effect among conditions. Each task required one or more of the following user actions:

- Replying to a message with personal information

- Replying with information found in emails in the inbox and other folders
- Replying after attaching files to the message
- Forwarding emails in the inbox or other folders
- Sorting messages into particular mailboxes
- Deleting messages

In addition to the inbox, the mail account contained the same tree of folders used in the visually impaired evaluation (see Section 6.3.2). Some of the emails in these other folders were referenced in the inbox messages and critical to the completion of the assigned tasks. Others were simply noise. The names of these folders stayed fixed across trials to avoid confusing the user, but their content changed to offset learning.

To motivate the trials, I read the following passage to each participant.

> Imagine you are a mobile professional. During a normal work day, you take a number of short trips away from your desktop computer. You've tried using a laptop and hand-held computer to work on-the-go, but find them troublesome in many situations. For example, the screen is too dim in the bright sunlight and too cumbersome while in transit.
>
> You recently learned about auditory displays for computers. You would like see if using one while away from the office can help you finish your work more quickly when you return to your desk. You would like to compare three different ways of completing your work:
>
> 1. Waiting to do all of your work when you return to the office.
> 2. Passively listening to, marking portions of, and selectively deleting your email before returning to the office.

3. Using all of the features of the auditory display to complete some work before returning to the office.

Before each of the three trials, I read the participant an additional passage describing the features of Clique that would be available during that trial. I then explained that the participant could ask me for help at any time about forgotten features of Clique, but not the task itself. Finally, I informed the participant that he or she should say "I'm done" aloud to signal completion. If the participant had indeed completed all of the tasks, the trial ended. If not, I told the participant to continue working and the trial continued. The postcondition of having all work correctly completed ensured that time was the only explicit dependent variable.

After each trial involving audio, the participant completed one copy of the the Perceived Usefulness and Ease of Use (PUEU) survey (Davis, 1989) and one of the NASA Task Load Index (NASA-TLX) (Hart and Staveland, 1988). The participant used the same questionnaire software as a user in the blind study (see Section 6.3), but this time through a graphical interface, not via Clique. I instructed the participant to respond to each segment with respect to their use of Clique in the most recent trial. For the perceived usefulness and ease of use survey, I also reminded the participant to consider a hypothetical situation in which they would be using Clique (e.g., in transit), not the test environment (e.g., sitting at a computer with its monitor turned off).

6.4.3 Execution

Participants

I recruited seventeen sighted participants from University of North Carolina at Chapel Hill campus and surrounding community. All participants were experienced in the use of at least one graphical desktop email client. Furthermore, all participants shared an interest in mobile audio as expressed by regular use of a portable music player such as an Apple iPod.

I used the results from the first four participants to refine the study procedure, particularly the amount of time necessary for training. The data from these sessions were not included in my analysis.

Setting

Sessions with participants took place in Sitterson Hall on the UNC-Chapel Hill campus. All participants interacted with a computer running Windows XP, Clique, Microsoft Outlook Express, and the PUEU and NASA-TLX questionnaire programs. All participants either used a full-sized, unsplit keyboard or a slightly smaller laptop keyboard plus external numeric keypad for input. All participants listened to 3D spatialized audio output from Clique through standard headphones.

6.4.4 Descriptive Statistics

For my analysis, I considered the total interaction time as the time spent using Clique plus the time spent using the GUI in each trial. I defined the time of completion as the time spent using the GUI alone[2]. Under this definition, the total interaction time and the time of completion were the same in the control condition with no Clique interaction.

Table 6.12 shows various decriptive statistics about the time of completion and total interaction time for each condition. Figures 6.13 and 6.14 depict some of these same values graphically using box and whisker plots.

According to these data, users completed work approximately 2.0 minutes sooner on average when using the full auditory display versus the GUI alone. Likewise, users completed work 1.5 minutes sooner with the minimal display. On the other hand, users

[2]The maximum linear offset of ten minutes spent working with Clique or doing nothing in the GUI-only trial has no effect on the analysis.

Time to Completion (seconds)								
Condition	*Min*	*1st Q*	*Med*	*3rd Q*	*Max*	*Mean*	*Std Dev*	*95% CI*
No audio	356.0	399.0	473.0	638.0	842.0	533.7	159.1	(437.5, 629.9)
Minimal	295.0	350.0	377.0	563.5	754.0	435.2	138.1	(351.8, 518.7)
Full	259.0	328.5	391.0	489.0	707.0	411.5	120.1	(338.9, 484.0)
Total Interaction Time (seconds)								
No audio	356.0	399.0	473.0	638.0	842.0	533.7	159.1	(437.5, 629.9)
Minimal	775.0	788.0	829.0	1047	1333	911.2	167.5	(810.0, 1012)
Full	859.0	914.0	991.0	1099	1307	1009	124.2	(934.2, 1084)

Table 6.12: Descriptive statistics for all sighted user study time measurements. The columns are the auditory display condition, minimum, first quartile, median, third quartile, maximum, mean, standard deviation, and 95% confidence interval of the mean.

interacted with a computer for roughly 7.9 minutes longer when using the full auditory display and 6.3 minutes longer with the minimal display before the GUI.

Figures 6.15, 6.16, and 6.17 depict the summed ratings for all ease of use questions, usefulness questions, and workload factors in both the minimal and full auditory display conditions. The graphs show the sums as percentages of the maximum possible rating. For ease of use and usefulness, higher percentages are better. For workload, lower percentages are better.

According to the usefulness and workload graphs, the summed ratings for minimal access to Clique were consistently lower than those for full access. The ratings for ease of use in the minimal condition were higher than those for the full condition as shown in the ease of use graph, except for the question about flexibility.

Table 6.13 shows statistics about the subjective workload scores computed using the NASA-TLX procedure for both the minimal and full auditory display conditions. The mean, median, minimum, and first and third quartiles were lower for the minimal display condition. The maximum scores were tied.

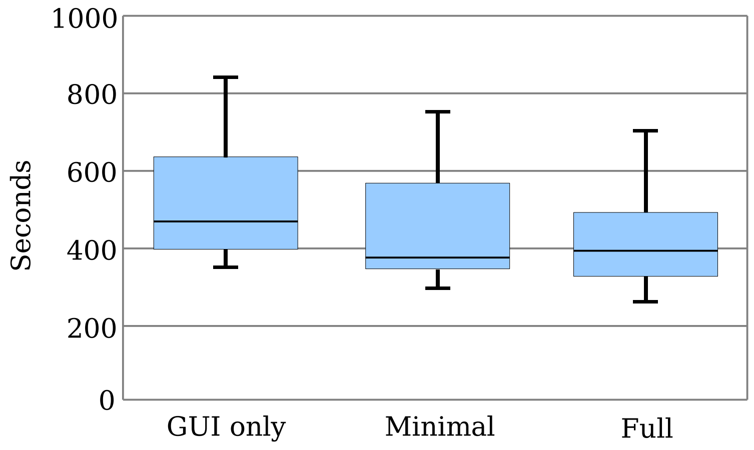


Figure 6.13: Box and whisker plot of the time of completion under the no audio, minimal audio, and full audio conditions. The values depicted are the minimum, first quartile, median, third quartile, and maximum. Lower is better.

Condition	*Min*	*1st Q*	*Med*	*3rd Q*	*Max*	*Mean*	*Std Dev*
Minimal	5.000	13.50	20.00	47.67	72.33	29.13	20.89
Full	20.33	33.67	42.33	62.33	72.33	46.08	16.90

Table 6.13: Descriptive statistics for all sighted user study workload scores. The columns are the auditory display condition, minimum, first quartile, median, third quartile, maximum, mean, and standard deviation.

6.4.5 Statistical Inference

I conducted hypothesis tests comparing objective participant performance and subjective participant ratings across conditions. For all tests, I considered my results significant if they could occur by random chance in less than one in twenty samples under the null hypothesis ($\alpha = 0.05$). Furthermore, all paired tests were two-tailed since I had no prior assumptions about which condition would fare best.

Time of Completion

I first tested the null hypothesis that the time spent working in the GUI, the time of completion, was the same under all conditions. Mauchly's test indicated that the

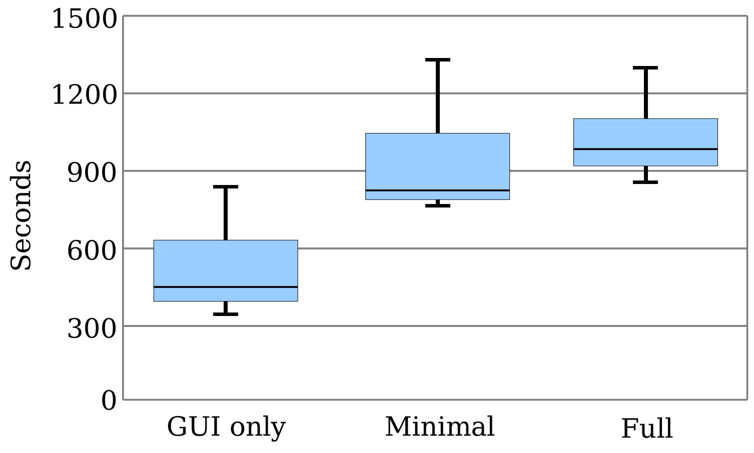


Figure 6.14: Box and whisker plot of the total time spent working under the no audio, minimal audio, and full audio conditions. The values depicted are the minimum, first quartile, median, third quartile, and maximum. Lower is better.

assumption of sphericity was not violated ($\chi^2(2) = 0.317, p = 0.853$). Therefore, I used uncorrected degrees of freedom in a repeated measures ANOVA. The results of this test show that there was a significant difference among the three conditions ($F(2, 12) = 6.965, p = 0.004$). After obtaining this result, I applied Dunnett's test comparing the minimal- and the full-audio experimental conditions to the GUI-only control condition. The results of this post-hoc test show that there was a significant difference in the time of completion between between the full and control conditions ($q = 3.520, p < 0.01$) and the minimal and control conditions ($q = 2.835, p < 0.05$). The 95% confidence interval for the difference between the control and full condition means was 40.64 to 203.8 fewer seconds. The 95% confidence interval for the difference between the control and minimal means was 16.87 to 180.1 fewer seconds.

With my chosen α, I rejected the null hypothesis and accepted the following alternatives:

- The time taken to complete all tasks using the GUI differed significantly among the three conditions.

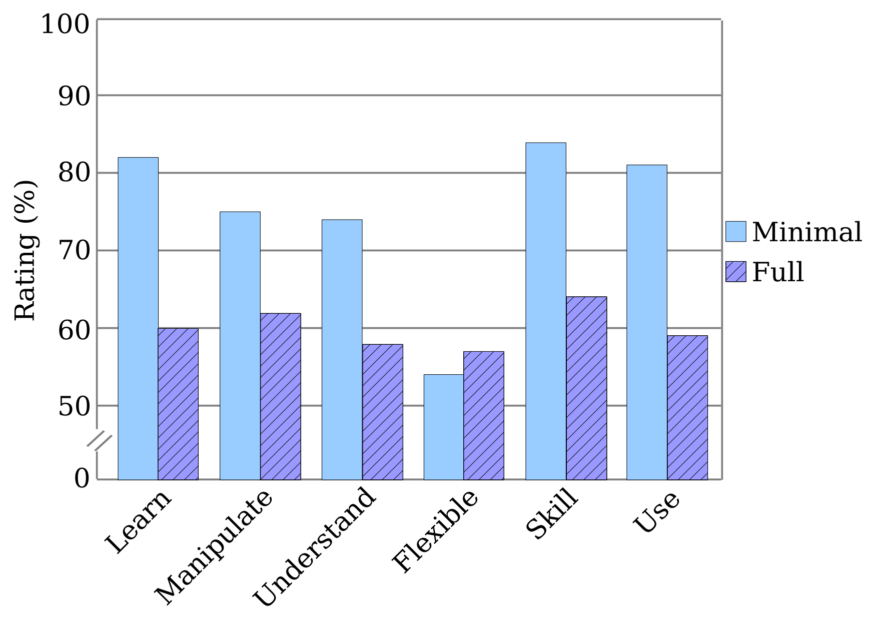


Figure 6.15: Summed ease of use ratings by question for the minimal and full audio conditions as a percentage of total possible rating. Higher is better.

- The time spent working in the GUI after having full auditory access was significantly less than the time spent working in the GUI alone. With a 5% chance of error, I can state that users complete work between 41 seconds and 3 minutes, 40 seconds faster after having full access to Clique. Users complete work approximately 2 minutes sooner on average.

- The time spent working in the GUI after having prior minimal auditory access is significantly less than the time spent working in the GUI alone. With a 5% chance of error, I can state that users complete work between 17 seconds and 3 minutes faster after having full access to Clique. Users complete work approximately 1.5 minutes sooner on average.

Total Interaction Time

I next tested the null hypothesis that the total interaction time was the same under all conditions. Mauchly's test indicated that the assumption of sphericity was not violated

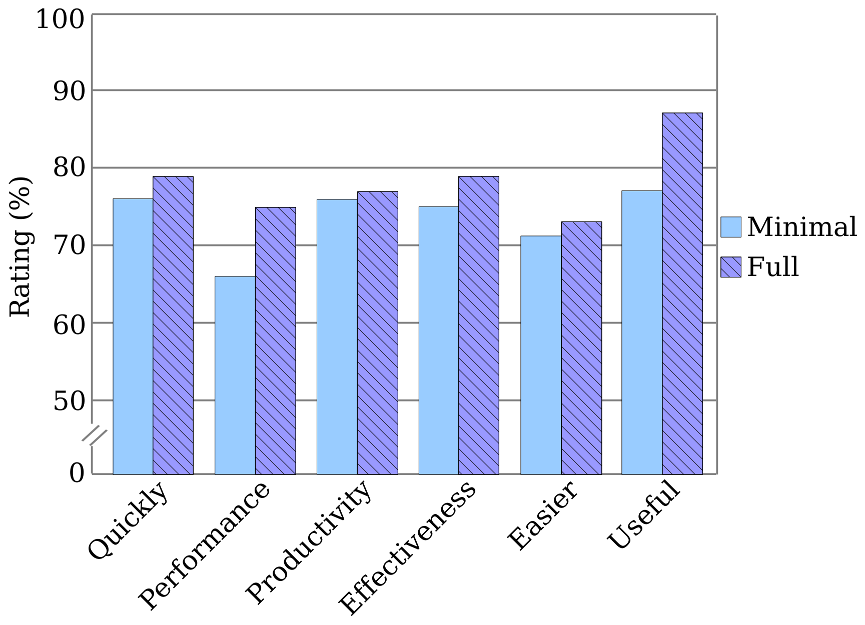


Figure 6.16: Summed usefulness ratings by question for the minimal and full audio conditions as a percentage of total possible rating. Higher is better.

($\chi^2(2) = 0.192, p = 0.909$), so I once again used uncorrected degrees of freedom in a repeated measures ANOVA. The results of this test show that there was a significant difference among the three conditions ($F(2, 12) = 94.15, p < 0.001$). I applied Dunnett's test again comparing the minimal- and the full-audio experimental conditions to the GUI-only control condition. The results of these post-hoc tests show that there was a significant increase in time spent working in the full condition over the control ($q = 12.99, p < 0.001$) and a significant increase in time spent working in the minimal condition over the control ($q = 10.32, p < 0.001$). The 95% confidence interval for the difference between the control and full condition means was 389.6 to 561.6 more seconds. The 95% confidence interval for the difference between the control and minimal means was 291.5 to 463.5 more seconds.

With my chosen α, I rejected the null hypothesis and accepted the following alternatives:

- The total time spent working on all tasks differs significantly among the three conditions.

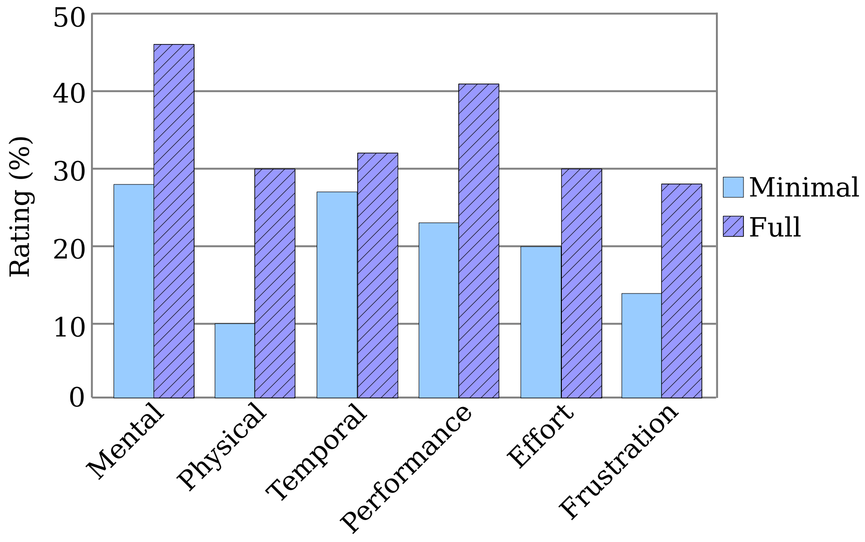


Figure 6.17: Summed workload factor ratings for the minimal and full audio conditions as a percentage of total possible rating. Lower is better.

- The total time spent working across the full auditory display and the GUI is significantly greater than the time spent doing all work using the GUI alone. With a 5% chance of error, I can state that users interact with a computer between 6.5 minutes and 9.4 minutes longer when using all features of Clique plus a GUI.

- The total time spent working across the minimal auditory display and the GUI is significantly greater than the time spent doing all work using the GUI alone. With a 5% chance of error, I can state that users interact with a computer between 4.9 minutes and 7.7 minutes longer when using a minimal portion of Clique plus a GUI.

Subjective Workload

For subjective workload, I tested the null hypothesis that the NASA-TLX weighted scores were the same in both the minimal and full auditory display conditions. The results of a paired t-test show that the workload scores associated with the minimal condition were significantly lower than that in the full auditory access condition with

Question	*p-value*	*Description*
Quickly	0.550	Not significant
Performance	0.046	Significant, full > minimal
Productivity	1.000	Not significant
Effectiveness	0.206	Not significant
Easier	0.776	Not significant
Useful	0.071	Not significant

Table 6.14: PUEU usefulness p-values.

my chosen α ($t(12) = 3.039, p = 0.010$). I rejected the null hypothesis and accepted the alternative that the subjective workload differed in favor of the minimal display.

Usefulness

I also tested the null hypothesis that overall perceived usefulness, as Findicated by the summed scores in the first half of the PUEU survey, was the same in both the minimal and full conditions. The results of another paired t-test show that the usefulness scores were not significantly different between the minimal and full conditions ($t(12) = 1.159, p = 0.269$). Therefore, I accepted the null hypothesis that the overall perceived usefulness of both displays was the same.

For a finer grained answer, I tested the null hypotheses that the ratings for each of the six usefulness questions on the PUEU survey were the same in the minimal and full display conditions. The results of six independent Wilcoxon Signed Ranks tests show a significant difference in only one question with my chosen α. Table 6.14 summarizes the questions, p-values, and direction of the difference.

Ease of Use

Finally, I tested the null hypothesis that overall perceived ease of use, as indicated by the summed ratings of the second half of the PUEU survey, was the same in both the minimal and full conditions. The results of a final paired t-test show that the ease of use scores were significantly lower in the minimal condition than in the full condition ($t(2) = 3.677, p = 0.003$). Therefore, I rejected the null hypothesis and accepted the alternative that the perceived ease of use differed in favor of the minimal display.

I also tested the null hypotheses that the individual ratings for each of the six ease of use questions on the PUEU survey were the same for the minimal and full display conditions. The results of six independent Wilcoxon Signed Ranks tests show significant differences in four questions with my α. Table 6.15 summarizes the questions, p-values, and direction of the differences.

Question	*p-value*	*Description*
Learn	0.006	Significant, minimal > full
Manipulate	0.093	Not significant
Understand	0.016	Significant, minimal > full
Flexible	0.889	Not significant
Skill	0.007	Significant, minimal > full
Use	0.006	Significant, minimal > full

Table 6.15: PUEU ease of use p-values.

6.4.6 Other Observations

All users asked for assistance remembering Clique keyboard commands at least once when interacting with the full Clique display. Most of the questions asked concerned navigation across and within messages, namely what keys to press. No user asked for assistance with the few commands available in the minimal Clique interaction trial.

The working behavior of participants in both audio trials was very similar. In the minimal condition, all users made frequent use of highlighting to mark messages requiring further action in the GUI and deletion to filter messages requiring no further attention. Some participants attempted to highlight near critical information in a message while others simply highlighted any segment to flag the message for re-reading in the GUI.

In the full interaction condition, participants consistently acted in the same manner by manually navigating among the messages, highlighting messages for later action, and filtering those of no importance. Most users attempted and successfully performed only one additional task when given full control: replying to an email requesting simple, personal information. Only one user attempted an additional task requiring searching a folder outside the inbox for a certain message. He did not use the memory features in Clique to record his search target. After spending a few minutes searching the correct folder, he forgot the search target, gave up, and returned to reading and highlighting inbox messages.

During training, all users asked about the background sounds and overlapping speech. I explained the purpose of each, and indicated their usefulness when multitasking across multiple applications, something not required in this study. Some users commented on both concepts after completing the study. The reaction to the ambient sound was mixed, with some participants saying the intermittent task sounds were annoying while others indicated they were easy to ignore and "faded into the background." Likewise, opinions about the concurrent speech varied. Some users said they had a very hard time focusing on one voice and others indicated they had no problem paying attention to a voice of interest. A few users reported that they learned to listen to the content assistent alone and tuned out all of the other voices.

6.4.7 Study Conclusions

Users appear to benefit from brief auditory access to email when use of a visual interface is not possible. Ten or fewer minutes of interaction with Clique saves users roughly a minute to two minutes of time completing common email tasks in a GUI. How efficiency scales with longer sessions with Clique is unknown, but worth further study.

The time spent working, however, increases when using Clique before using a GUI. A trade-off exists between getting email tasks done as soon as possible versus getting them done with the least amount of effort. Clique aids the first goal, but its current implementation is severely detrimental to the second. Continued practice, an increased speech rate, a more domain-specific input device, and an email script tailored for the tasks sighted mobile users typically complete could all drive down the total time spent working. Reducing the total interaction time by improving Clique is another topic for additional study.

Users suggest that having full control over Clique increases their workload over a design that sacrifices features in favor of some automation. Similarly, they indicate that Clique is easier to learn, understand, and use when only a subset of its commands are available. In light of the observation that the methods used by participants in the full and minimal auditory conditions were similar, this suggestion points to difficulty navigating as the primary source of the difference. Nevertheless, users believe that their performance is better when they have full access to Clique commands rather than a subset.

Additional training might reduce the perceived workload and increase the perceived ease of use while keeping all of the features in Clique. If so, this training might also retain the perceived performance benefits of the full auditory display seen in this study. Alternatively, a better blend of the full and minimal conditions might result in low workload, high perceived performance, and ease of use without requiring significant additional training. For example, adding the task-specific commands *Next Message* and *Previous Message* to the current minimal access design might serve this purpose. A few participants suggested this exact idea or some form of it (e.g., a single *Skip Message* command).

6.5 Adaptation Feasibility

As a final evaluation of Clique, I studied the development effort invested in creating the seven Clique application scripts used in these studies. Table 6.16 shows some measurements on the code in these scripts. In this table, *system subclasses* are direct descendants of interaction pattern, adapter, and macro classes not tied to a single application. For instance, *CreateNote* is a subclass of the reusable *FormFill* interaction pattern class. *Custom subclasses* are descendants of system subclasses tied to the application adapted by the script. Continuing the same example, *EditNote* is a subclass of *CreateNote* which is specialized for the calendar application. *New classes* are patterns and adapters implemented from scratch to support some task or subtask in one and only one application. Again, in the calendar script, the new *CalendarModel* class adapts multiple, disparate widgets unique to the Day by Day Professional application to the reusable *Tree* interaction pattern.

I distinguished these three kinds of classes to assist in my evaluation of code reuse. System subclasses indicate the reuse of code across application scripts. Custom subclasses indicate the reuse of code within an application script. New classes indicate the need for code supporting a one-off interaction in a single application.

According to these data, the seven scripts specialize reusable, system classes (83% of all classes) and subclass their own, application-specific classes (17% of all classes) most often. New classes account for only 1% of the total classes. Figure 6.18 depicts this comparison graphically.

The lines of code (LOC) range from 67 to 362. These endpoints are indicative of the extreme regularity in some applications and the extreme customization in others. The PUEU script has relatively few lines of code because its GUI consists of a repeating set of common widgets as shown in Appendix A, Figure A.1. On the other hand, the calendar program shown in Chapter 5, Figure 5.11 has a very irregular GUI for browsing dates and requires more adaptation to bring it into line with standard Clique interaction

Script	*Tasks*	*LOC*	*System Sub.*	*Custom Sub.*	*New Cl.*	*Total Cl.*
Email	10	280	16	3	0	19
Notepad	3	111	10	1	0	11
Archives	10	280	24	3	0	27
Calendar	7	362	14	1	1	16
PUEU	5	67	3	5	0	8
TLX	4	91	5	4	0	9
Web Browser	4	205	12	0	1	13

Table 6.16: Measurements on application scripts. The columns state the name of a script, the number of tasks it supports, the number of lines of code (omitting whitespace and comments), the number of system subclasses, the number of indirect system subclasses, the number of new classes, and the total number of classes in the script. The rows are sorted according to the order in which I implemented the scripts. Tasks within a *Wizard* pattern count as separate tasks.

patterns. Even with this special code, however, the calendar adapter is still less than six times longer than the shortest script.

As another basis for comparison, the PUEU and NASA-TLX GUI programs, also written in Python, have 517 and 710 lines of code respectively, 96 lines of which is shared between the two. The Clique scripts for both applications provide access to all functions of the original programs in 87% less code in both cases. In the case of the Web browser script, the difference is greater. The Firefox code supporting the Windows accessibility API alone is thousands of lines long. Even considering that it is written in C++ instead of Python and that the Clique script does not support all features of the browser, the working Clique script is staggeringly smaller. Comparisons with the other applications are not possible because they are closed source.

Table 6.17 derives metrics about the script code. Over the seven scripts, support for a single task requires between roughly 13.4 and 51.7 lines of code, and 1.6 and 3.3 classes. At best, supporting a single task requires one class specializing an interaction pattern. These numbers show that the scripts require more than one class per task in practice, however. The macros required to properly drive the underlying GUIs account

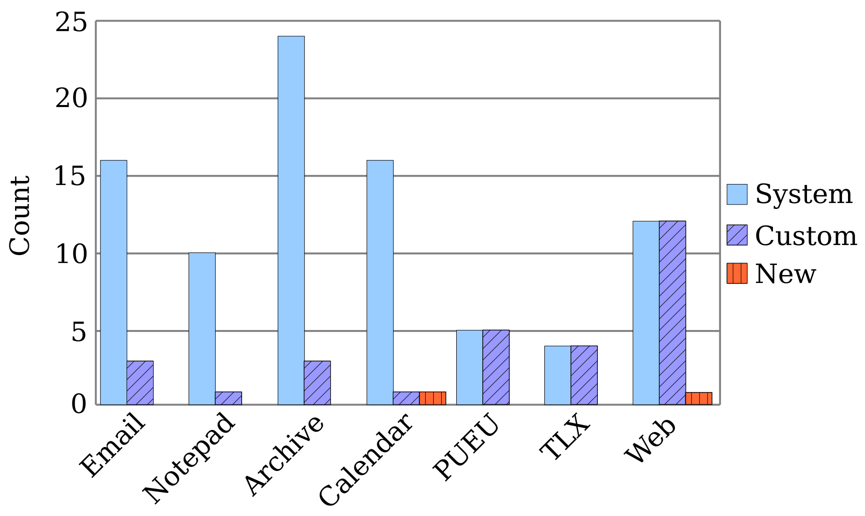


Figure 6.18: Class reuse in application scripts. The counts indicate the number of system-provided components subclassed, the number of script components subclassed, and number of components created from scratch by the scripts.

for the overhead in most cases. Further specializations of subtask patterns and adapters contribute the remaining excess.

Script	*LOC per Task*	*Classes per Task*
Email	28.0	1.9
Notepad	37.0	3.7
Archives	28.0	2.7
Calendar	51.7	2.3
PUEU	13.4	1.6
TLX	22.8	2.3
Web Browser	51.3	3.3

Table 6.17: Metrics on application scripts. The columns name the scripts, the number of lines of code per task, and the number of classes per task derived from the raw measurements.

I implemented the seven application scripts one after another without any overlap in time. While writing each script, I logged the new interaction pattern and adapter classes I had to develop to support all of the target tasks in the corresponding application.

Figure 6.19 depicts the number of new interaction pattern and adapter classes added for each script.

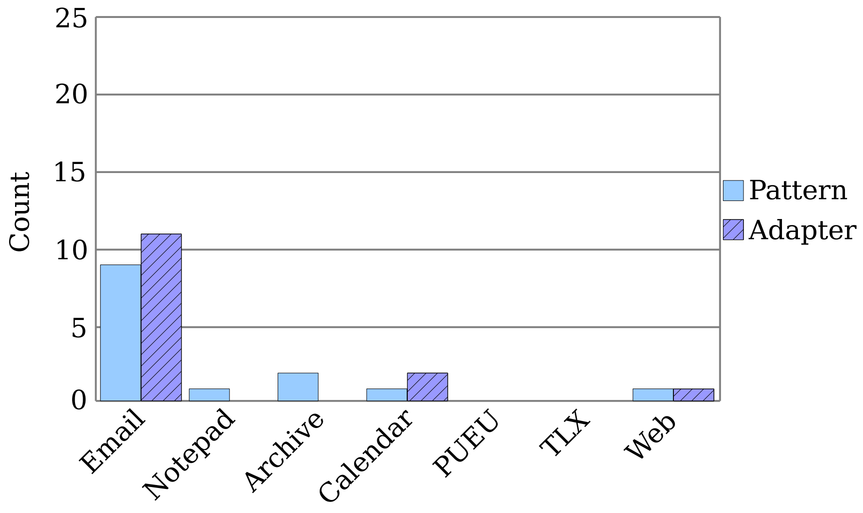


Figure 6.19: Class additions during prototype development. The counts represent the number of interaction and adapter classes added to the core of Clique for each script. Scripts are sorted from left to right in order of implementation.

The first script implemented, the one for Microsoft Outlook Express, accounts for 71% of all of the interaction pattern and adapter classes required by all seven scripts, 64% of all interaction pattern classes and 78% of all adapter classes. The following seven scripts require only 8 additional classes, or 19% of the total classes, to support their tasks.

Figure 6.20 shows the same class additions, but in a worst case order. The script with the fewest total classes appears first and the one with the most appears last. There is no clear trend here as in the previous graph, most likely because the first few scripts support very few tasks. It is worth noting, however, that the script with the greatest number of classes included no new classes beyond those provided in the previous scripts.

Figures 6.21 and 6.22 show the frequency of use of all implemented interaction pattern and adapter classes across the seven scripts. According to the interaction patterns histogram, there are few very common interaction patterns such as *List*, *Form Fill*, and

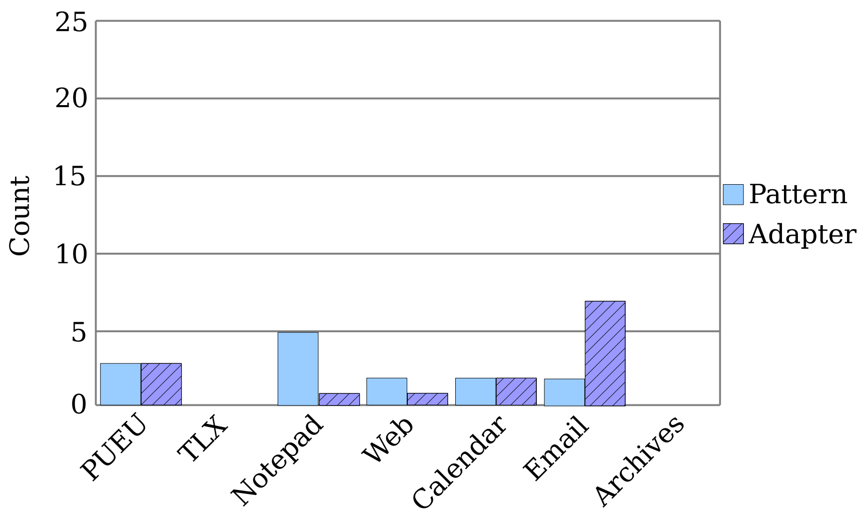


Figure 6.20: Class additions ordered by script size in classes. The counts represent the number of interaction and adapter classes added to the core of Clique for each script. Scripts are sorted from left to right in ascending order of total number of classes in the script.

Text Entry. At the same time, there are many patterns used infrequently as indicated by the long tail with low values. While three patterns are used only once in these seven scripts, at least two of them, *File saving* and *Document Reading* are applicable to interactions in applications beyond the sample of seven. The remaining pattern, *Strided List*, is potentially not a pattern at all, but rather a one-off solution required by the calendar script alone.

The histogram of adapter use paints a different picture. The values of the most frequently used adapters in this graph are noticeably lower than those in the interaction patterns histogram. This difference is partly due to the fact that task-level interaction patterns such as *Form Fill* do not require a specialized adapter class to function. Instead, task patterns use the adapter base class directly, reducing the number of interaction-specific adapters needed overall.

Still, this property of task patterns does not account for the reduced number of adapters for subtask patterns. In particular, the *List* pattern appears 20 times across

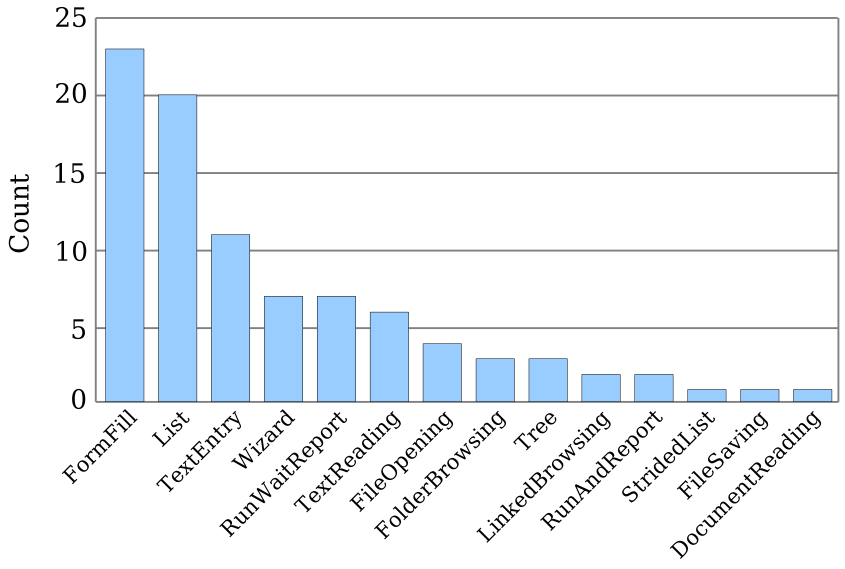


Figure 6.21: Interaction pattern use. The horizontal axis names both task and subtask level interaction patterns. The vertical axis states the number of times each pattern appears in one of the seven application scripts.

the seven scripts, but the similarly named *List* adapter appears only nine times. This apparent discrepancy is a result of the many-to-one relationship between adapter models and interaction pattern views/controllers. That is, many adapters can satisfy the model-interface requirements of a single interaction pattern, and so the values in the adapter histogram are spread more evenly over the bins. In the case of the *List* pattern, the adapters *Button List*, *Editable List*, *Checkbox*, *Column List*, and plain *List* all act as valid models.

6.5.1 Benefits

The data in the tables and graphs above support a few claims about the feasibility of Clique script development. First, the lines of code required for Clique scripts are likely less than that required for the original application providing both the business logic and GUI. In the case of the PUEU, NASA-TLX, and Web browser scripts, the number of lines of code is less than that used in the associated GUI applications. The survey scripts provide equivalent functionality as their associated GUI applications in

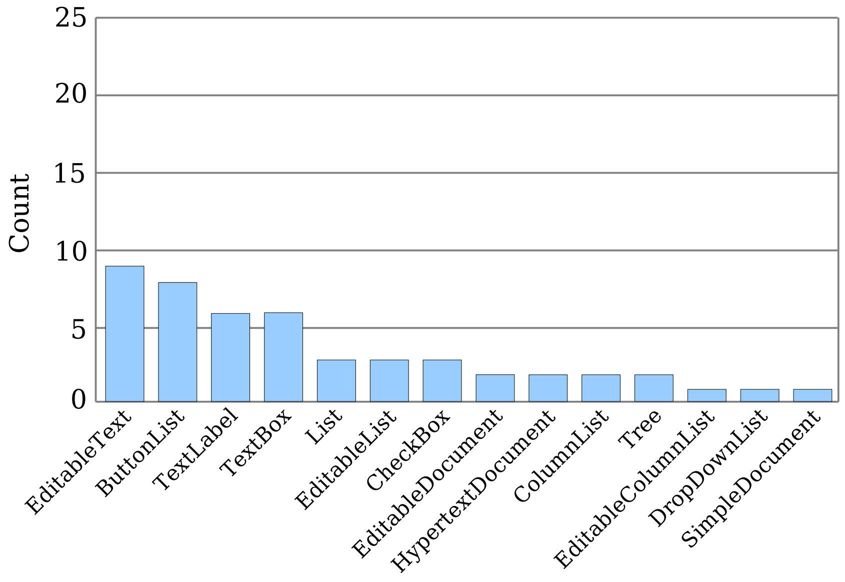


Figure 6.22: Interaction pattern use. The horizontal axis names model adapters patterns. The vertical axis states the number of times each adapter appears in one of the seven application scripts.

about eight times less code. The Web browser script does not provide access to all of the tasks supported by the Mozilla Firefox browser, but the lines of code needed to browse basic, static Web pages is certainly less than the code required to fetch, parse, and render the documents in the browser itself. It is very likely, but not proved, that a script supporting all browser features would require less code than the entire browser code base, even when accounting for differences in language verbosity. The same claim might not hold for the amount of code in the interaction patterns and adapters used by the three scripts. However, these components are reusable, so implementation is a one-time cost, not a per script expense.

Second, the ability to reuse interaction patterns is probably high across office productivity applications. This claim holds true for the seven implemented scripts at least. Their associated GUIs appear to repeat a few common interactions which are nicely suited to the pattern approach. Additional scripts for similar applications in the document editing, file management, and text-based communication categories seem likely to specialize many of the same patterns.

Third, the ability to reuse adapter classes is seemingly high as well. The existence of GUI toolkits and a limited set of widget types (e.g., button, text field, listbox, combobox) limits the number of different adapters required across the seven application scripts. Since many desktop applications share GUI toolkits and widgets, additional scripts are likely to reuse the same adapters as-is or with minor specializations to support implementation differences. Likewise, supporting widgets on entirely different platforms is presumably an exercise in reimplementing the same adapters components using different accessibility APIs. Still, the number of adapter classes is likely to eclipse the number of interaction pattern classes: the diversity of widget types overwhelms the fundamental tasks they support. The *List* pattern with its many compatible adapters is indicative of this trend.

Fourth, the extensibility of Clique is likely sufficient to support the development of new interaction pattern or adapter components not strictly tied to vision. The separation of models adapting GUI widgets from controllers handling user commands and from views reporting information aurally, enables scripts to customize any subset of these components independently. The calendar script, for example, customizes the view and model for browsing dates, but reuses the existing controller for browsing tree-like data. In addition, the macro concept guarantees arbitrarily complex, asynchronous GUI manipulations are possible when required by adapters to maintain proper external model state.

Fifth, the design of the input-output pipeline provides some assurance that new scripts and components will conform to the intended Clique user experience. The input pipeline guarantees commands are delivered to the proper recipients. The output pipeline ensures inactive components cannot dominate the primary assistants. Short of dynamically modifying the Clique framework at run time, a script cannot deviate greatly from the assistant-environment concept in Clique auditory display. This design limits the flexibility of Clique scripts, but relieves some of the burden of designing usable interactions from script writers.

Finally, the mere use of a general purpose programming language instead of a language specialized for script writing bears mentioning. All modern, commercial screen readers, particular JAWS, rely on custom, single-purpose languages for scripting. The ability to use a language under active development with a growing standard library surely provides more tools, ensures more flexibility, reduces bugs, and invites more developer interest than a special-purpose language with a closed scripting API.

6.5.2 Shortcomings

In spite of the benefits, some shortcomings exist in the Clique scripting framework. First, there is a lack of guidance in mapping GUI widgets to tasks and subtasks. Script writers are currently left to their own devices when deciding what sets of graphical displays and manipulations constitute a task. This shortcoming is a topic of future research discussed in the next chapter.

Second, a developer must update Clique scripts when application GUIs change across versions. The same requirement applies to screen reader scripts, and is a general problem when using accessibility APIs. As mentioned in Section 5.3.4 of Chapter 5, advances in API support for uniquely identifying widgets is the ideal solution to this problem.

Finally, every application requires a script to function in Clique. In contrast, screen readers attempt to report basic information such as widget names and focus changes in unscripted applications. At face value, it appears Clique requires drastically more developer effort than screen readers to enable use of applications in audio. However, two additional facts diminish this claim. First, Clique could fall back on screen reading for unscripted applications. The current prototype does not offer this option, but nothing about its design inherently prevents it.

Second, to support efficient, sustained use of complex programs, screen readers do require scripts. Only the most basic GUIs are truly usable without per application extensions. For instance, JAWS has scripts to improve the usability of Mozilla Firefox, Microsoft Outlook Express, and Winzip. In some cases, screen readers even require changes to their core code base to function properly with certain applications. For example, the core of the JAWS screen reader must contain a great deal of specialized code supporting Firefox. The JAWS script for Firefox is too simple to offer the full functionality actually provided, and only recent versions of JAWS are billed as Firefox compatible.

6.6 Conclusion

This chapter presented the results of five user evaluations of the Clique design and prototype implementation. The speech stream timing analysis proved that Clique concurrent streams provide faster access to utterances than a single stream speaking at the same rate. Whether users could actually take advantage of this improvement was a question left for a later user study.

The heuristic evaluation identified no major shortcomings preventing Clique from user evaluation. Nevertheless, the evaluation did reveal ways of improving the ambient information streams. The results indicated that ambient reports could benefit from better identification of their sources. Increased user control over peripheral reports, including when and how they are made, was suggested as a means of improving the flexibility of the display.

The results of a first user study showed that blind users are able to hear and report target utterances in concurrent streams with high accuracy. Paired with the proof in the timing analysis, these results confirmed that users could take advantage of simultaneous streams to hear desired information sooner. The non-zero accuracy of participants reporting non-target information also indicated a possible increase in bandwidth from the display to the listener over what is possible with a single stream.

The blind user study also showed improvements in user ability to search for content using Clique instead of JAWS. The benefits to using Clique were significant in a program not providing a built-in search utility, and modest in a program offering an innate search dialog. In addition, user accuracy in reporting both input given and output heard after learning to complete three unfamiliar tasks was significantly better with Clique than with JAWS. Fewer required interactions with Clique versus JAWS was suggested as the reason for this difference. On the other hand, no difference was seen between user performance with Clique and JAWS when working across four applications to complete common office tasks. Likewise, users reported similar levels of workload for both systems. Participant comments indicated the belief that their performance with Clique would surpass that with JAWS given more experience.

The results of sighted participant user study indicated that having either minimal or full access to email via Clique significantly improved time to email task completion when later working with a GUI. Nevertheless, the total interaction time was significantly greater when using Clique and a GUI instead of a GUI alone. The same study showed that subjective workload with the full auditory display was significantly higher than that associated with the minimal display. No overall significant difference was detected between the perceived usefulness of the two auditory display conditions. However, user ratings of ease of use did differ significantly.

The analysis of Clique script development suggested that Clique scripts require less code than the GUI programs they adapt, potentially by orders of magnitude for complex GUI programs. The data also indicated that interaction patterns and adapters are reusable across application scripts. Where reuse fails, the extensibility of the Clique pattern and adapter libraries is sufficient for developing components to support additional interactions, GUI widgets, and entire audio-aware programs written from scratch. The framework itself guides the development of new auditory views and controllers for tasks and subtasks so as to ensure a basic level of usability. Yet, writing scripts still requires developer finesse in defining meaningful tasks and subtasks based on GUI-supported interactions.

The final chapter discusses future work in light of the results of these evaluations. The chapter includes recommendations on how to improve the design of Clique and ideas on how to extend its capabilities to make it both more usable and useful.

Chapter 7

Conclusion

The goal of this research was to evaluate the following thesis statement:

> Adapting GUI applications to an auditory display that describes concurrent user tasks via multiple streams of spatialized speech and sound provides a better user experience than a single stream narration of the screen. Such an approach can improve the awareness and effectiveness of people with visual impairments working in a multitasking environment, benefit sighted users working away from a graphical desktop, and function across diverse applications using third-party scripts.

The evaluations described in Chapter 6 lend credence to these claims. Users with visual impairments are able to hear more information, sooner with concurrent audio streams than a single stream reporting on a multi-tasking environment. These same users are able to locate information faster in certain applications and describe how to use applications more accurately using the task-based tools in Clique versus the screen-based tools in a screen reader. Subjective ratings and open-ended comments made by experienced screen reader users favor Clique in contexts where intimate knowledge of the screen is not required. In addition, users with vision are able to complete email tasks in a GUI more quickly if they have previous exposure to the messages in Clique than if they had none at all. Finally, I was able to compose seven Clique scripts supporting diverse applications and tasks in less than 1400 lines of code total. All Clique components supporting these applications can be reused and extended in future scripts.

7.1 Suggestions for Future Research

The work I have done far from exhausts the possible avenues for research on Clique as an adaptation method, a software system, or a user interface. I believe the following topics are worth further investigation.

7.1.1 Further Evaluation

The existing Clique prototype invites additional questions about its usability for people with and without vision. Further user evaluations can elucidate additional benefits and shortcomings of the design, and spark new ideas for improvements.

- Users with visual impairments received relatively little training on Clique in comparison with their many years of experience using JAWS. How would the systems compare after extended Clique training and use? Would user performance in workflow trials remain similar with both Clique and JAWS?

- Sighted users completed email tasks roughly one to two minutes sooner after using Clique for ten minutes. How would time to completion decrease when increasing the duration of Clique use, if at all? Would using Clique improve task efficiency in other applications as well, or even work across applications?

- Results from the sighted user study suggest a level of interaction between the two levels studied may increase efficiency and satisfaction. How would users perform with such a hybrid? How would they rate it in terms of ease of use, usefulness, and workload?

- Users with visual impairments are able to hear and understand utterances in concurrent speech streams quicker than in a single stream with high accuracy. How would increasing or decreasing the rate of speech affect the intelligibility of concurrent streams? At what rate is a single stream equivalent to a set of concurrent streams in terms of accuracy and times to hear utterances?

- Users with vision made extensive use of the Clique memory tool when working between the auditory and visual domains. What advantage does this feature provide, if any, over the equivalent GUI selection tools available to screen reader users?

7.1.2 Visuals and Collaboration

The current design of Clique completely divorces what the user hears from the visual display. There are times, though, when knowledge of what is actually on the screen may be useful. For example, a Clique user collaborating with a sighted user may wish to explain what actions he or she is taking in terms of the GUI. Likewise, the Clique user may need to translate visual concepts the sighted user is describing into Clique concepts. There are ways to solve these problems without discarding the task-based auditory display.

One solution is to enable both task-based and screen-based interaction. In one mode, the user would interact with Clique as it is currently implemented, perhaps when working alone. In the alternative mode, the display would behave exactly like a screen reader, possibly to aid collaboration. The user would have the ability to switch between modes at will. The display would also share this ability to revert to screen reading when encountering unknown, unscripted applications.

The difficulty to overcome in this approach is the synchronization needed between GUI widgets and Clique tasks when switching modes, especially from screen reading to tasks. In this case, Clique would need to inspect the GUI to determine what task and subtask to activate. The adapters in the active task would then need to update their interal state to match the GUI. Finally, Clique would need to describe the working context to help the user understand the relation of the last active widget to the currently active task. Exactly how the switching of modes and synchronization would occur is a topic for future work.

If continuous interaction in terms of the GUI is not necessary, a different solution is possible: adding commands to Clique that describe tasks, subtasks, and actions in terms of the underlying GUI. The *Where am I command?* would describe not only the active subtask, task, and application, but also the widget, window, and so forth in focus. It is straightfoward to extend Clique models such that they can provide information about the visual state of the widget or widgets they adapt. A new command, *Explain this action*, would describe Clique commands in terms of the actions they perform on the underlying GUI. The Clique adapters and macro infrastructure could be enhanced to support this capability.

Once again, it is important to note that whether or not collaboration between blind and sighted users is most effective in terms of graphical concepts is unknown. It is equally possible that communication between these users is more natural in terms of the tasks they are trying to jointly accomplish. It is even more likely that the answer depends on the competancy of the two participants at using their respective user interfaces, the knowledge they share about the work at hand, the availability of common terms needed to describe their works, and so forth. Further study should be done on this topic before assuming a hybrid visual plus task-based auditory display is the best answer.

7.1.3 Automated Script Creation

The Clique scripting framework is flexible enough to support diverse applications and tasks. It is not, on the other hand, friendly to people without programming skills. A person must have knowledge about the programming language, Clique APIs, and platform accessibility architecture to successfully script applications. Moreover, to produce useful scripts, a developer must study user actions in applications and exercise some creativity in defining tasks. The prototype I implemented does not address these shortcomings, but nothing in the design of Clique bars a solution.

Long-term, programmatic monitoring of GUI use is a possible supplement or alternative to explicitly coding tasks into a script. In this solution, one or more users would volunteer to run the monitor software in the background while working with desktop applications. The monitor would watch for common sequences of mouse and keyboard input triggering commands in the GUI during user interaction. After noting "interesting" repetitions, the monitor would come to the foreground, and inquire about the purposes of these commands. The user would then assert whether the commands form a common task; name the task; modify the inputs needed to start and complete the task; and indicate the widgets involved in the task. The tool would generate script tasks and macros based on the provided information.

This solution could also generate Clique autotasks in the same manner, namely by watching for application initiated dialogs and monitoring user inputs to complete them. In addition, a wizard-like user interface could guide an expert user through the definition of a task without monitoring. Users would explicitly perform the steps needed to complete a task rather than having the monitor tool infer them from everyday work.

The fact that existing screen reader users could take advantage of this approach is especially interesting. Since the GUI is exposed when working through a screen reader, the technique of building scripts using the monitor tool would apply to this situation as well. Developing a hybrid screen reader plus task-based auditory display as described in the previous section also affords the possibility of integrating the monitor/generator tool directly into the display. In such a design, the display itself would watch the user interact with the GUI in screen reader mode and assist the user in defining tasks for the Clique-like mode.

One problem the monitoring approach does not solve is that of extending Clique to support new types of interaction patterns and adapter models. It may be best for this task to remain the duty of a software developer with insight into what constitutes a usable auditory display for a particular set of widgets and interactions.

7.1.4 User Customization

Customization of Clique is an important topic not explored in this dissertation. The current prototype follows a design having fixed assistants, static task definitions, predefined commands, and so forth, primarily for the purpose of the evaluations I conducted. The results of these studies indicate mixed participant opinions and performance using the rigid prototype. Greater flexibility and user control over the format of the display may better accommodate diverse abilities, preferences, and tasks. Aspects of Clique that lend themselves to customization without serious harm to the overall usability include the following:

- The maximum number of concurrent active streams. The system should support serialization or silencing of streams on demand.

- The use of speech versus sound to convey particular pieces of information. The system should provide a means of configuring notifications as either spoken utterances or non-speech sounds.

- The sound properties of all speakers. Clique should allow users to choose the location, timbre, baseline pitch, accent, etc. of all assistants.

- The assignment of assistants. Clique should let users define new or modify existing assistant responsibilities.

- The rate of interaction. The display should should let users configure rates of speech, durations of sounds, and lengths of forced delays.

- The grouping of tasks. The display should allow the user to form sets of related tasks enabling rapid navigation in addition to basic application-based groupings.

7.1.5 More Tools

Most of the user needs noted in Chapter 3 are directly supported by the features implemented in Clique, but not all of them. A more explicit help system, for example, would explain the abilities and limitations of Clique and the application in use. A single command, *What can I do here?* could report on all of the commands available in a given context. System wide undo, as another example, would better support user repair than the ability to cancel tasks alone. To support this feature, the input manager could log all input messages dispatched. In response to an *Undo* command, the manager would dispatch the message logged, but with a special *reverse* property set. The controller handling the message would be responsible for reverting the changes made by the equivalent, non-reversed command. In response to a *Redo* command, the input manager would simply dispatch the last message undone, this time with the reverse flag unset.

The goal of the concurrent streams in Clique is to allow rapid attention switching among them at any time. This feature allows fast navigation over the available views of the task space. However, this navigation is passive: it does not allow a user to direct user input to a particular assistant. All input goes to the subtask described by the content assistant. Providing modifiers to direct commands at particular assistants, and therefore particular tasks and subtasks, would improve multitasking efficiency and the utility of the peripheral streams. For example, issuing a *Take me there* command after hearing a report from an inactive task would provide a fast method of lateral navigation. At the same time, a *You do this* quasimode would allow the user to issue commands to inactive tasks and subtasks without explicitly suspending and resuming the active task.

7.1.6 More Tasks

The design of Clique focused exclusively on the searching, browsing, and creation of text. All user tasks, however, are not limited to character-based representations. Raster images, vector diagrams, waveform sounds, and videos are all examples of common media types either lacking text or not purely based on text. Though dealing with some of these information formats through an auditory display may not be ideal for all users (i.e., a sighted user is much better off viewing a raster image than trying to listen to it), some users with visual impairments may not have viable alternatives. Exploring how such media might fit into the Clique user experience is worth effort if enabling these users is a primary concern.

The browsing and editing of vector diagrams using Clique is not a far fetched scenario, especially if a diagram contains metadata describing its components and their relationships. For example, one could develop a graph interaction pattern supporting the task of traversing the transitions and states in a flow diagram. The content assistant might read any text associated with a node while the summary assistant previews incoming and outgoing connections. The overview streams might indicate the overall size and complexity of the diagram as well as the location of the user's point of regard within it. The input commands might allow browsing of properties associated with a state, while edges to traverse might appear in a list format common to Clique menus. The addition, deletion, and editing of nodes and edges might appear as conditional tasks in the task menu when they are supported.

Dealing with the other media types mentioned may be easier, as in the case of playing audio samples, or more difficult, as in trying to sonify raster images to convey meaning (Meijer, 1992). Any attempt at making these data available in Clique, however, can follow the same pattern used for text content: defining an interaction pattern (e.g., bitmap browsing), binding it to an adapter and GUI component (e.g., a bitmap canvas), and using the pattern to define a specific application task (e.g., listening to an image in a paint program). The user would activate such a task in the same manner as others, while the interaction pattern would be free to produce output using the existing assistant constraints or override them as necessary (e.g., using pitch to represent pixel intensity).

7.2 Final Comments

This dissertation shows quantitative and qualitative advantages of some Clique features over their screen reader alternatives for tasks not tied intimately to vision. For these advantages to become true benefits, they must find their way into the hands of people with visual impairments who truly need them. Fortunately, some key Clique tools such as quasimodal search, infinite memory, and concurrent streams can be readily integrated into existing systems. Concepts like task-based interaction and sound environments, on the other hand, require a shift in thinking about how people with visual impairments access computers. What I have seen in this research assures me that advocating for a more Clique-like approach is a worthwhile endeavor.

To this end, I have placed the Clique source code under a liberal open source license. While I do not plan to continue development on the original prototype, I will wholeheartedly support developers wishing to pilfer any of it to improve their assistive technology projects. I do, however, wish to explore the possibility of building a Clique-like auditory display on an open source software stack, if only to produce an alternative for those users who cannot afford, cannot use, or prefer not to use a screen reader.

Appendix A

User Study Surveys

The two summative user studies described in Chapter 6 incorporated questionnaires for collecting subjective user responses about the usefulness, ease of use, and workload of Clique. This appendix documents the content of these questionnaires, how they were scored, and how they were completed by study participants.

A.1 Perceived Usefulness and Ease of Use

The Perceived Usefulness and Ease of Use (PUEU) questionnaire (Davis, 1989) makes twelve statements, six about the potential usefulness of the system and six about its potential ease of use. Participants rate each statement on a seven point scale (one to seven, unlikely to likely). The ratings for the first six questions are summed to produce an overall usefulness score while the last six are summed to produce an overall ease of use score.

When summing, I assume the ratings exist on an interval scale with participants able to determine the equal distances among values after rating multiple statements about ease and usefulness. This assumption of an interval scale invites the use of parametric statistics when drawing inferences about overall ease and usefulness. When comparing ratings of individual statements, however, I treat the data as ordinal and apply more conservative non-parametric statistics.

The following two sections list the twelve statements for reference.

A.1.1 Usefulness

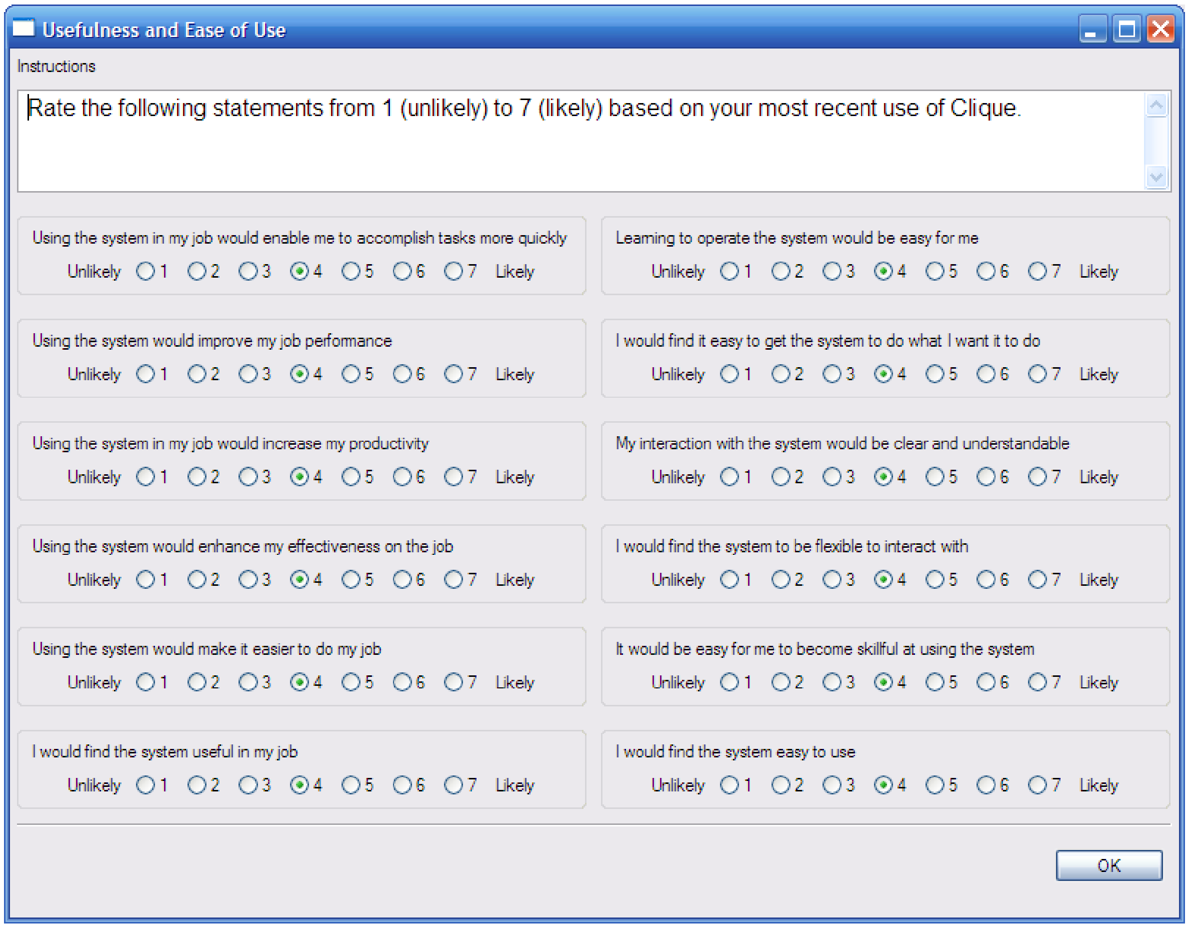
1. Using the system in my job would enable me to accomplish tasks more quickly
2. Using the system would improve my job performance
3. Using the system in my job would increase my productivity
4. Using the system would enhance my effectiveness on the job
5. Using the system would make it easier to do my job
6. I would find the system useful in my job

A.1.2 Ease of Use

1. Learning to operate the system would be easy for me
2. I would find it easy to get the system to do what I want it to do
3. My interaction with the system would be clear and understandable
4. I would find the system to be flexible to interact with
5. It would be easy for me to become skillful at using the system
6. I would find the system easy to use

A.1.3 Software

I created a computerized version of the PUEU questionnaire using the Python programming language and the wxPython widget toolkit. Since wxPython wraps the native Windows widgets, the resulting GUI was accessible via Microsoft Active Accessibility (MSAA) and could be adapted for use in Clique. Sighted study participants interacted with the graphical interface shown in Figure A.1 while participants with visual impairments used the Clique auditory equivalent described in Chapter 5.



Rate the following statements from 1 (unlikely) to 7 (likely) based on your most recent use of Clique.

Using the system in my job would enable me to accomplish tasks more quickly
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

Using the system would improve my job performance
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

Using the system in my job would increase my productivity
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

Using the system would enhance my effectiveness on the job
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

Using the system would make it easier to do my job
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

I would find the system useful in my job
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

Learning to operate the system would be easy for me
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

I would find it easy to get the system to do what I want it to do
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

My interaction with the system would be clear and understandable
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

I would find the system to be flexible to interact with
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

It would be easy for me to become skillful at using the system
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

I would find the system easy to use
Unlikely 1 2 3 [✓ 4] 5 6 7 Likely

Figure A.1: Perceived Usefulness and Ease of Use survey software.

A.2 NASA-TLX

The NASA Task Load Index (Hart and Staveland, 1988) asks participants to follow two procedures. First, participants assign ratings on a twenty value scale to the six workload sources described in Table A.1. Second, participants review the fifteen possible pairings of the workload sources and choose the one source of each pair that they believe contributed more to the overall workload. The workload rating for each source is multiplied by the percentage of times the source was selected over all fifteen pairings. The weighted scores are then summed to produce an overall workload score.

The use of weighting in this procedure assumes a ratio scale for all scores. Therefore, when comparing NASA-TLX weighted scores, I apply parametric statistics.

The instructions to the participant and description of the six sources are listed in the following three sections for reference.

A.2.1 Instructions for Ratings

We are not only interested in assessing your performance but also the experiences you had during the different task conditions. Right now we are going to describe the technique that will be used to examine your experiences. In the most general sense we are examining the "Workload" you experienced. Workload is a difficult concept to define precisely, but a simple one to understand generally. The factors that influence your experience of workload may come from the task itself, your feelings about your own performance, how much effort you put in, or the stress and frustration you felt. The workload contributed by different task elements may change as you get more familiar with a task, perform easier or harder versions of it, or move from one task to another. Physical components of workload are relatively easy to conceptualize and evaluate. However, the mental components of workload may be more difficult to measure.

Since workload is something that is experienced individually by each person, there are no effective "rulers" that can be used to estimate the workload of different activities. One way to find out about workload is to ask people to describe the feelings they experienced. Because workload may be caused by many different factors, we would like you to evaluate several of them individually rather than lumping them into a single global evaluation of overall workload. This set of six rating scales was developed for you to use in evaluating your experiences during different tasks. Please read the descriptions of the scales next to each rating carefully. If you have a question about any of the scales, please ask me about it. It is extremely important that they be clear to you.

Title	*Endpoints*	*Description*
Mental Demand	Low/High	How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc.)? Was the task easy or demanding, simple or complex, exacting or forgiving?
Physical Demand	Low/High	How much physical activity was required (e.g., pushing, pulling, turning, controlling, activating, etc.)? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?
Temporal Demand	Low/High	How much time pressure did you feel due to the rate or pace at which the tasks or task elements occurred? Was the pace slow and leisurely or rapid and frantic?
Effort	Low/High	How hard did you have to work (mentally and physically) to accomplish your level of performance?
Performance	Good/Poor	How successful do you think you were in accomplishing the goals of the task set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?
Frustration Level	Low/High	How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task?

Table A.1: NASA-TLX rating definitions.

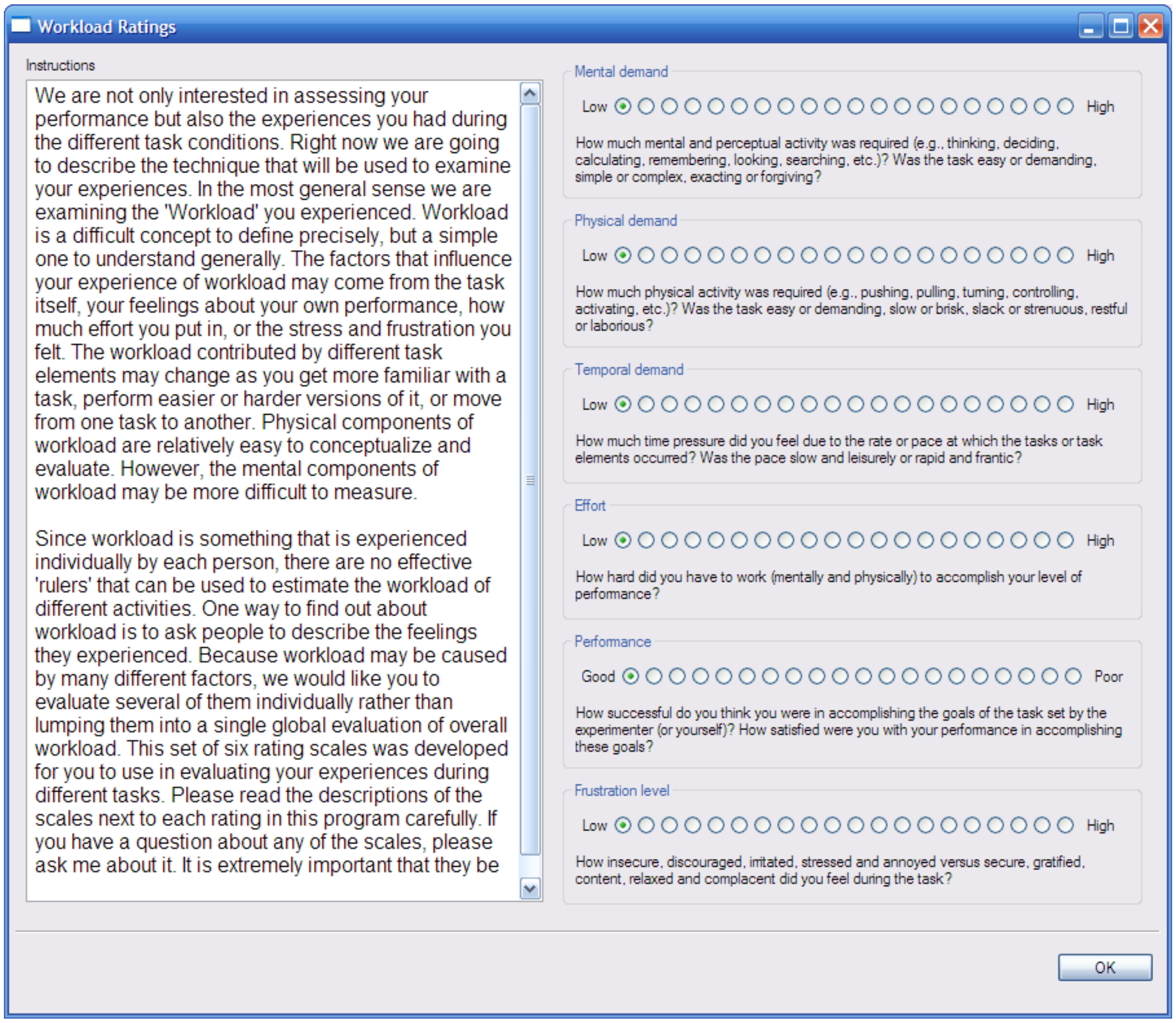
A.2.2 Instructions for Sources of Workload

Throughout this experiment the rating scales are used to assess your experiences in the different task conditions. Scales of this sort are extremely useful, but their utility suffers from the tendency people have to interpret them in individual ways. For example, some people feel that mental or temporal demands are the essential aspects of workload regardless of the effort they expended or the performance they achieved. Others feel that if they performed well the workload must have been, low, and vice versa. Yet others feel that effort or feelings of frustration are the most important factors in workload; and so on. The results of previous studies have already found every conceivable pattern of values. In addition, the factors that create levels of workload differ depending on the task. For example, some tasks might be difficult because they must be completed very quickly. Others may seem easy or hard because of the intensity of mental or physical effort required. Yet others feel difficult because they cannot be performed well, no matter how much effort is expended.

The evaluation you are about to perform is a technique that has been developed by NASA to assess the relative importance of six factors in determining how much workload you experienced. The procedure is simple: You will be presented with a series of pairs of rating scale titles (for example, Effort vs. Mental Demands) and asked to choose which of the items was more important to your experience of workload in the task(s) that you just performed. Select the Scale Title that represents the more important contributor to workload for the specific task(s) you performed in this experiment.

A.2.3 Software

A computerized version of the NASA-TLX software is available online (`http://www.nrl.navy.mil/aic/ide/NASATLX.php`). However, this software is inaccessible to Clique and screen readers, does not show complete instructions on the screen, and does not allow for the correction of mistakes during the selection of workload sources procedure. To account for these problems, I created a version of the NASA-TLX questionnaire using the Python programming language and the wxPython widget toolkit. This implementation ensured the resulting GUI was accessible via Microsoft Active Accessibility (MSAA) and could be adapted for use in Clique. Sighted study participants interacted with the two graphical interfaces shown in Figure A.2 and Figure A.3 while participants with visual impairments used the Clique scripts described in Chapter 5.



Instructions

We are not only interested in assessing your performance but also the experiences you had during the different task conditions. Right now we are going to describe the technique that will be used to examine your experiences. In the most general sense we are examining the 'Workload' you experienced. Workload is a difficult concept to define precisely, but a simple one to understand generally. The factors that influence your experience of workload may come from the task itself, your feelings about your own performance, how much effort you put in, or the stress and frustration you felt. The workload contributed by different task elements may change as you get more familiar with a task, perform easier or harder versions of it, or move from one task to another. Physical components of workload are relatively easy to conceptualize and evaluate. However, the mental components of workload may be more difficult to measure.

Since workload is something that is experienced individually by each person, there are no effective 'rulers' that can be used to estimate the workload of different activities. One way to find out about workload is to ask people to describe the feelings they experienced. Because workload may be caused by many different factors, we would like you to evaluate several of them individually rather than lumping them into a single global evaluation of overall workload. This set of six rating scales was developed for you to use in evaluating your experiences during different tasks. Please read the descriptions of the scales next to each rating in this program carefully. If you have a question about any of the scales, please ask me about it. It is extremely important that they be

Mental demand

Low ◉ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ High

How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc.)? Was the task easy or demanding, simple or complex, exacting or forgiving?

Physical demand

Low ◉ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ High

How much physical activity was required (e.g., pushing, pulling, turning, controlling, activating, etc.)? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?

Temporal demand

Low ◉ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ High

How much time pressure did you feel due to the rate or pace at which the tasks or task elements occurred? Was the pace slow and leisurely or rapid and frantic?

Effort

Low ◉ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ High

How hard did you have to work (mentally and physically) to accomplish your level of performance?

Performance

Good ◉ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ Poor

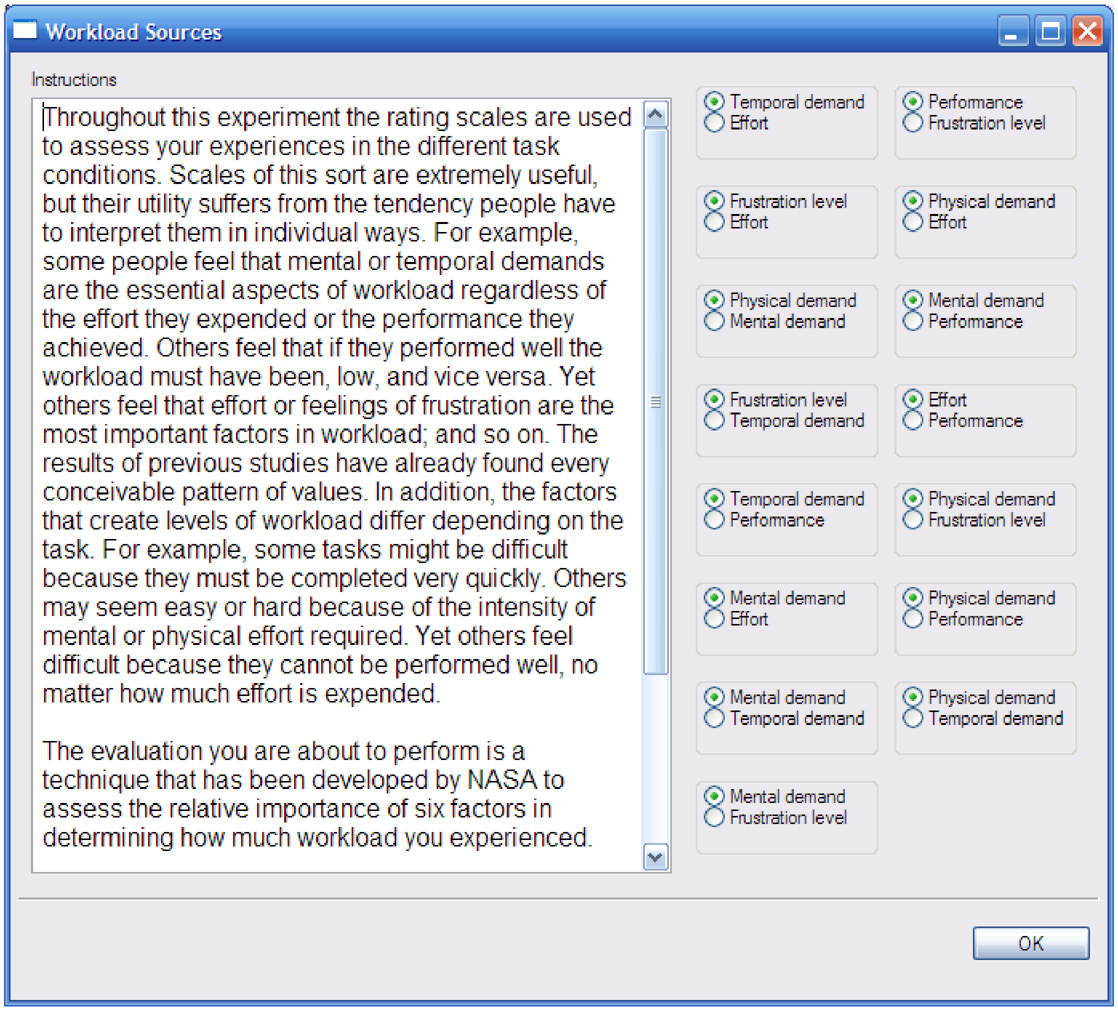
How successful do you think you were in accomplishing the goals of the task set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?

Frustration level

Low ◉ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ High

How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task?

Figure A.2: NASA Task Load Index ratings survey software.


Workload Sources

Instructions

Throughout this experiment the rating scales are used to assess your experiences in the different task conditions. Scales of this sort are extremely useful, but their utility suffers from the tendency people have to interpret them in individual ways. For example, some people feel that mental or temporal demands are the essential aspects of workload regardless of the effort they expended or the performance they achieved. Others feel that if they performed well the workload must have been, low, and vice versa. Yet others feel that effort or feelings of frustration are the most important factors in workload; and so on. The results of previous studies have already found every conceivable pattern of values. In addition, the factors that create levels of workload differ depending on the task. For example, some tasks might be difficult because they must be completed very quickly. Others may seem easy or hard because of the intensity of mental or physical effort required. Yet others feel difficult because they cannot be performed well, no matter how much effort is expended.

The evaluation you are about to perform is a technique that has been developed by NASA to assess the relative importance of six factors in determining how much workload you experienced.

[✓ Temporal demand] Effort

[✓ Performance] Frustration level

[✓ Frustration level] Effort

[✓ Physical demand] Effort

[✓ Physical demand] Mental demand

[✓ Mental demand] Performance

[✓ Frustration level] Temporal demand

[✓ Effort] Performance

[✓ Temporal demand] Performance

[✓ Physical demand] Frustration level

[✓ Mental demand] Effort

[✓ Physical demand] Performance

[✓ Mental demand] Temporal demand

[✓ Physical demand] Temporal demand

[✓ Mental demand] Frustration level

Figure A.3: NASA Task Load Index sources survey software.

Appendix B

Sounds and Source

B.1 Audio Recordings

Audio recordings of various aspects of Clique are available online under the Creative Commons Attribution-Share Alike 3.0 United States License. Visit `http://www.cs.unc.edu/~parente/clique` to access the sounds and view the terms of the license.

B.1.1 Attribution

The Clique prototype uses sounds from the Freesound repository (`http://freesound.iua.upf.edu/index.php`) licensed under Creative Commons. Visit the following URLs for the exact license terms and more information about the authors.

- By man (http://freesound.iua.upf.edu/usersViewSingle.php?id=14447): soldati-marcia.aif (`http://freesound.iua.upf.edu/samplesViewSingle.php?id=14624`)
- By jnr hacksaw (`http://freesound.iua.upf.edu/usersViewSingle.php?id=29612`): Zap.flac (`http://freesound.iua.upf.edu/samplesViewSingle.php?id=11221`)
- By csengeri (`http://freesound.iua.upf.edu/usersViewSingle.php?id=197070`): Cricket2.wav (`http://freesound.iua.upf.edu/samplesViewSingle.php?id=34218`)

B.2 Obtaining the Source

The full source code of the Clique prototype is available for download under the terms of the new BSD license. See the LICENSE file included with the source for details. Visit `http://www.cs.unc.edu/~parente/clique` to obtain the code.

B.3 Example Notepad Script

Listing B.1 shows the source code for the Microsoft Notepad script for Clique. I have included this code here for instructive purposes, namely as a concrete example of the ceoncepts discussed in Chapter 5. Readers wishing to explore or use the code in more detail should visit the URL in the previous section.

Listing B.1: Notepad script

```
import UIA
from UIA import Adapters
from View import Task, Control

class Main(UIA.Macro):
  def Sequence(self):
    # do nothing on start
    yield True

class DoStartDocumentWindow(UIA.Macro):
  def Sequence(self):
    # watch for notepad window
    self.WatchForNewWindow(Name='Untitled - Notepad', ClassName='Notepad')
    # run the notepad application
    self.RunFile('notepad.exe')
    yield False
    yield True

class DoCloseDocumentWindow(UIA.Macro):
  def Sequence(self):
    # watch for the are you sure dialog
    self.WatchForNewWindow(Name='Notepad', ClassName='#32770', name='confirm')
    # watch for the window closing
    self.WatchForWindowClose(self.model.Window, 'done')
    # close the window
    self.CloseWindow()
    yield False
    # close the confirmation dialog if it appears
    if self.name == 'confirm':
      self.SendKeys('N')
    yield True

class DoSaveFile(UIA.Macro):
  def Sequence(self):
    # press the save hotkey
    self.SendKeys('^{s}')
    yield True

class SaveFile(Task.RunAndReport):
  Name = 'save file'
  StartMacro = DoSaveFile()

class SaveFileAs(Task.FileSaving):
  # let the user choose a filename and location to save
  Name = 'save file as'
  Modal = True
  StartMacro = UIA.StartWindowByKey(key_combo='%{f}a',
                                    Name='Save as',
                                    ClassName='#32770')
  CompleteMacro = UIA.EndWindowByKey(key_combo='{ENTER}')
  CancelMacro = UIA.EndWindowByButton()
  filename_path = Task.FileSaving.EXTENDED_FILENAME_PATH

  def OnInit(self):
    self.FilterByExtension('txt')

class OpenFileStep(Task.FileOpening):
  # let the user choose a file to open
  StartMacro = UIA.StartWindowByKey(key_combo='^{o}', ClassName='#32770',
                                    Name='Open')
  CompleteMacro = UIA.EndWindowByKey(key_combo='{ENTER}')
  CancelMacro = UIA.EndWindowByButton()
  filename_path = Task.FileOpening.EXTENDED_FILENAME_PATH

class EditDocument(Task.FormFill):
  # allow the user to edit the document
  Name = 'new document'
  Permanence = Task.NO_COMPLETE
  body_path = '/client[3]/window[0]/editable text[3]'
  count = 1
  CancelMacro = DoCloseDocumentWindow()

  def OnInit(self):
    doc = Adapters.EditableTextBox(self, self.body_path)
    self.AddField(Control.TextEntry(self, doc, 'document'))

    # condition for save without save dialog
    def SaveCond():
      self.UpdateName()
      return not self.Name.startswith('edit untitled ')

    # always offer save as, but only offer save if file already has a name
    self.AddContextOption(SaveFileAs, True)
    self.AddContextOption(SaveFile, True, SaveCond)

  def OnGainFocus(self, message):
    self.UpdateName()

  def OnLoseFocus(self, message):
    self.UpdateName()

  def UpdateName(self):
    # pretty up the name of untitled documents
    try:
      name = self.model.Name
    except:
      return
    fn, tmp = name.split(' - ')
    unnamed = (name == 'Untitled - Notepad')
    if unnamed and not self.Name.startswith('edit untitled '):
      self.Name = 'edit untitled %d' % EditDocument.count
      EditDocument.count += 1
    elif not unnamed:
      self.Name = 'edit %s' % fn

class NewDocument(EditDocument):
  # start a new document and then let the user edit it
  StartMacro = DoStartDocumentWindow()

class OpenDocument(Task.Wizard):
  # open an existing document and then let the user edit it
  Name = 'open document'
  Successor = EditDocument
  count = 1
  StartMacro = DoStartDocumentWindow()
  CancelMacro = UIA.EndWindowByButton()

  def OnInit(self):
    self.AddStep(OpenFileStep)

  def OnLoseFocus(self, message):
    # give the open task a meaningful name
    unnamed = (self.Name == OpenDocument.Name)
    if unnamed:
      self.Name = 'continue opening file %d' % OpenDocument.count
      OpenDocument.count += 1
    else:
      pass

class AutoConfirmDialog(Task.FormFill):
  Modal = True
  Name = 'confirm overwrite'
  Permanence = Task.NO_CANCEL
  CompleteMacro = UIA.EndWindowByKey(key_combo='{ENTER}')
  yes_path = '/dialog[3]/window[0]/push button[3]'
  no_path = '/dialog[3]/window[1]/push button[3]'
  text_path = '/dialog[3]/window[3]/text[3]'
  def OnInit(self):
    bl = Adapters.ButtonList(self, [self.no_path, self.yes_path])
    label = Adapters.TextLabel(self, self.text_path)
    self.AddField(Control.List(self, bl, label))

  def Trigger(cls, event, model, container):
    return not (model.State & UIA.Constants.STATE_SYSTEM_SIZEABLE) and \
           model.Name.startswith('Save') and model.ClassName == '#32770'
  Trigger = classmethod(Trigger)

Tasks = [NewDocument, OpenDocument]
AutoTasks = [AutoConfirmDialog]
```

Bibliography

Alexander, C. (1979). *The Timeless Way of Building*. Oxford University Press, Oxford, United Kingdom.

Alty, J. L. and Rigas, D. I. (1998). Communicating graphical information to blind users using music: The role of context. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 574–581. ACM Press/Addison-Wesley Publishing Co.

Arnout, K. (2004). *From Patterns to Components*. PhD thesis, Swiss Federal Institute of Technology, Zurich, Switzerland.

Arons, B. (1992). A review of the cocktail party effect. *Journal of the American Voice I/O Society*, 12:35–50.

Arons, B. (1994). Efficient listening with two ears: Dichotic time compression and spatialization. In *International conference on auditory display*. International Community on Auditory Display.

Asakawa, C. and Itoh, T. (1998). User interface of a Home Page Reader. In *Proceedings of the third international ACM conference on Assistive technologies*, pages 149–156. ACM Press.

Asakawa, C., Takagi, H., Ino, S., and Ifukube, T. (2003). Maximum listening speeds for the blind. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Baldis, J. J. (2001). Effects of spatial audio on memory, comprehension, and preference during desktop conferences. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 166–173. ACM Press.

Barber, J. R., Razak, K. A., and Fuzessery, Z. M. (2003). Can two streams of auditory information be processed simultaneously? Evidence from gleaning bat Antrozous pallidus. *Journal of Comparitve Physiology*, 189:843–855.

Barnicle, K. (2000). Usability testing with screen reading technology in a windows environment. In *Proceedings on the 2000 conference on Universal Usability*, pages 102–109. ACM Press.

Barra, M., Cillo, T., Santis, A. D., Matlock, T., Petrillo, U. F., Negro, A., Scarano, V., and Maglio, P. P. (2001). Personal Webmelody: Customized sonification of web servers. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Bates, M. (1989). The design of browsing and berrypicking techniques for the on-line search interface. *Online Review*, 13(5):407–431.

Beament, J. (2001). *How We Hear Music: The Relationship Between Music and the Hearing*. Boydell Press.

Beaudouin-Lafon, M. and Conversy, S. (1996). Auditory illusions for audio feedback. In *Conference companion on Human factors in computing systems*, pages 299–300. ACM Press.

Bennett, D. J. and Edwards, A. D. N. (1998). Exploration of non-seen diagrams. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Bernsen, N. O., r, H. D., and r, L. D. (1998). *Designing interactive speech systems*. Springer-Verlag, New York, New York, United States.

Beshers, C. M. and Feiner, S. (1989). Scope: automated generation of graphical interfaces. In *Proceedings of the 2nd annual ACM SIGGRAPH symposium on User interface software and technology*, pages 76–85. ACM Press.

Best, V., Ihlefeld, A., and Shinn-Cunningham, B. (2005). The effect of auditory spatial layout in a divided attention task. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Bey, C. and McAdams, S. (2002). Schema-based processing in auditory scene analysis. *Perception and Psychophysics*, 64(5):844–854.

Bickmore, T. W. (2004). Unspoken rules of spoken interaction. *Communications of the ACM*, 47(4):38–44.

Binding, C., Schmandt, C., Lantz, K. A., and Arons, B. (1990). Workstation audio and window-based graphics: Similarities and differences. *Engineering for Human-Computer Interaction*, 1:233–247.

Blattner, M., Sumikawa, D., and Greenberg, R. (1989). Earcons and icons: Their structure and common design principles. *Human Computer Interaction*, 4:11–44.

Blenkhorn, P. and Evans, G. (2000). Architecture and requirements for a Windows screen reader. In *IEEE Seminar on Speech and Language Processing for Disabled and Elderly People*, pages 1–4. IEEE Press.

Borenstein, J. and Ulrich, I. (1997). The GuideCane: A computerized travel aid for the active guidance of blind pedestrians. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1283–1288.

Boyd, L. H., Boyd, W. L., and Vanderheiden, G. C. (1990). The graphical user interface: Crisis, danger, and opportunity. *Journal of Visual Impairment and Blindness*, 12:496–502.

Bregman, A. S. (1990). *Auditory Scene Analysis: The Perceptual Organization of Sound*. The MIT Press, Cambridge, Massachusetts, United States.

Brennan, S. E. (1990). Conversation as direct manipulation: An iconoclastic view. In Laurel, B., editor, *The art of human-computer interface design*, pages 393–404. Addison-Wesley, Reading, Massachusetts, United States.

Brennan, S. E. (1998). The grounding problem in conversations with and through computers. In *Social and cognitive psychological approaches to interpersonal communication*, chapter 9, pages 201–225. Lawrence Erlbaum.

Brennan, S. E. and Hulteen, E. A. (1995). Interaction and feedback in a spoken language system: A theoretical framework. *Knowledge-Based Systems*, 8:143–151.

Brewster, S., Lumsden, J., Bell, M., Hall, M., and Tasker, S. (2003). Multimodal 'eyes-free' interaction techniques for wearable devices. In *Proceedings of the conference on Human factors in computing systems*, pages 473–480. ACM Press.

Brewster, S. A., Wright, P. C., and Edwards, A. D. N. (1993). An evaluation of earcons for use in auditory human-computer interfaces. In *CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 222–227, New York, NY, USA. ACM Press.

Brown, L. and Brewster, S. (2003). Drawing by ear: Interpreting sonified line graphs. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Brown, L. M., Brewster, S. A., Ramloll, R., Burton, M., and Riedel, B. (2003). Design guidelines for audio presentation of graphs and tables. In *Proceedings of the international conference on auditory display*. International Community on Auditory Display.

Brungart, D. S. and Simpson, B. D. (2003). Optimizing the spatial configuration of a seven-talker speech display. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory display.

Burgess, D. A. (1992). Techniques for low cost spatial audio. In *ACM Symposium on User Interface Software and Technology*, pages 53–59.

Burns, E. M. (1999). Intervals, scales, and tuning. In Deutsch, D., editor, *The Psychology of Music*. Academic Press, San Diego, California, United States, 2nd edition.

Card, S. K. (1983). *Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates.

Card, S. K., Robertson, G. G., and York, W. (1996). The WebBook and the Web Forager: An information workspace for the World-Wide Web. In *Proceedings of the Conference on Human Factors in Computing Systems*.

Carroll, J. M., Mack, R. L., and Kellogg, W. A. (1997). Interface metaphors and user interface design. In Helander, M., Landauer, T., and Prabhu, P., editors, *Handbook of human-computer interaction*, pages 67–85. Elsevier, Amsterdam, The Netherlands, 2nd edition.

Cassell, J., Bickmore, T., Campbell, L., Vilhjálmsson, H., and Yan, H. (2000). Human conversation as a system framework: Designing embodied conversational agents. In *Embodied conversational agents*, pages 29–63. MIT Press.

Catledge, L. D. and Pitkow, J. E. (1995). Characterizing browsing strategies in the World-Wide Web. *Computer Networks and ISDN Systems*, 27(6):1065–1073.

Cherry, E. C. (1953). Some experiments on the recognition of speech, with one and with two ears. *Journal of Acoustic Society of America*, 25:975–979.

Choo, C. W., Detlor, B., and Turnbull, D. (2000). Information seeking on the web: An integrated model of browsing and searching. *First Monday*, 5(2):1.

Clark, H. H. and Wilkes-Gibbs, D. (1986). Referring as a collaborative process. *Cognition*, 22:1–39.

Clark, J. and DeRose, S. (1999). XML path language (XPath). Technical report, W3C.

Cohen, M. and Ludwig, L. F. (1991). Multidimensional audio window management. *International Journal of Man-Machine Studies*, 34:319–336.

Cusack, R., Deeks, J., Aikman, G., and Carlyon, R. P. (2004). Effects of location, frequency region, and time course of selective attention on auditory scene analysis. *Journal of Experimental Psychology: Human Perception and Performance*, 30(4):643–656.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13:318–341.

Demarey, C. and Plénacoste, P. (2001). User, sound context, and use context: What are their roles in 3D sound metaphor design? In *Proceedings of the international conference on auditory display*. International Community on Auditory Display.

Dewan, P. and Solomon, M. (1990). An approach to support automatic generation of user interfaces. *ACM Trans. Program. Lang. Syst.*, 12(4):566–609.

Duggan, B. and Deegan, M. (2003). Considerations in the usage of text to speech (TTS) in the creation of natural sounding voice enabled web systems. In *ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies*, pages 433–438. Trinity College Dublin.

Duncan, J., Martens, S., and Ward, R. (1997). Restricted attentional capacity within but not between sensory modalities. *Nature*, 387:808–810.

Edwards, A. D. N. (1988). The design of auditory interfaces for visually disabled users. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 83–88. ACM Press.

Edwards, A. D. N. (1991). Evaluation of Outspoken software for blind users. Technical Report YCS150, University of York, Department of Computer Science, York, England.

Edwards, A. D. N. (1992). Graphical user interfaces and blind people. In *Proceedings of the 3rd International Conference on Computers for Handicapped Persons*, pages 114–119.

Edwards, W. K., Mynatt, E., and Stockton, K. (1994). Providing access to graphical user interfaces-not graphical screens. In *Proceedings of the first annual ACM conference on Assistive technologies*, pages 47–54. ACM Press.

Ellis, D. and Haugan, M. (1997). Modeling the information seeking patterns of engineers and research scientists in an industrial environment. *Journal of Documentation*, 53:384–403.

Embley, D. W. and Nagy, G. (1981). Behavioral aspects of text editors. *ACM Comput. Surv.*, 13(1):33–70.

Farkas, W. and Jeon, H. T. (2006). Information music. `http://www.soundtomind.com/`.

Fernström, M. and McNamara, C. (1998). After direct manipulation - direct sonification. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California at Irvine, Irvine, California, United States.

Finlayson, L. and Mellish, C. (2005). The audioview providing a glance at java source code. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Flowers, J. H., Whitwer, L. E., Grafel, D. C., and Kotan, C. A. (2001). Sonification of daily weather records: Issues of perception, attention and memory in design choices. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Foley, J., Kim, W. C., Kovacevic, S., and Murray, K. (1989). Defining interfaces at a high level of abstraction. *IEEE Software*, 6:25–32.

Frauenberger, C., Höldrich, R., and de Campo, A. (2004). A generic, semantically based design approach for spatial auditory computer displays. In *Proceedings of the international conference on auditory display*. International Community on Auditory Display.

Frauenberger, C., Putz, V., Höldrich, R., and Stockman, T. (2005). In *Proceedings of the International Conference on Auditory Display*, pages 154–160, Limerick, Ireland. International Community on Auditory Display.

Frauenberger, C., Stockman, T., and Bourguet, M. (2007). paco ad - pattern design in the context space; a methodological framework for auditory display design. In *Proceedings of the International Conference on Auditory Display*, Montreal, Canada. International Community on Auditory Display.

Friberg, J. and Gärdenfors, D. (2004). Audio games: new perspectives on game audio. In *ACE '04: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 148–154, New York, NY, USA. ACM Press.

Gajos, K. and Weld, D. S. (2004). SUPPLE: Automatically generating user interfaces. In *Proceedings of the 2004 International Conference on Intelligent User Interfaces*, Funchal, Portugal.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of reusable object-oriented software*. Addison-Wesley Professional, 1st edition.

Gaver, W. W. (1986). Auditory icons: Using sound in computer interfaces. *Human Computer Interaction*, 2(2):167–177.

Gaver, W. W. (1993). Synthesizing auditory icons. In *Proceedings of the conference on Human factors in computing systems*, pages 228–235. Addison-Wesley Longman Publishing Co., Inc.

Gaver, W. W. (1997). Auditory interfaces. In Helander, M., Landauer, T., and Prabhu, P., editors, *Handbook of human-computer interaction*, pages 1003–1041. Elsevier, Amsterdam, The Netherlands, 2nd edition.

Gaver, W. W., Smith, R. B., and O'Shea, T. (1991). Effective sounds in complex systems: the ARKOLA simulation. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 85–90, New York, NY, USA. ACM Press.

Geisler, G. (2000). Enriched links: a framework for improving web navigation using pop-up views. Technical Report TR-2000-02, UNC, INLS, Chapel Hill, North Carolina, United States.

Geisler, G. (2003). *AgileViews: A framework for creating more effective information seeking interfaces*. PhD thesis, UNC, INLS, Chapel Hill, North Carolina, United States.

Geisler, G., Marchionini, G., Wildemuth, B. M., Hughes, A., Yang, M., Wilkens, T., and Spinks, R. (2002). Video browsing interfaces for the Open Video Project. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 514–515, New York, NY, USA. ACM Press.

Gelfand, S. A. (2004). *Hearing: An introduction to psychological and physiological acoustics.* Marcel Dekker.

Ghez, C., Rikakis, T., Dubois, R. L., and Cook, P. R. (2000). An auditory display system for aiding interjoint coordination. In *Proceedings of the International Conference on Auditory Display.* International Community on Auditory Display.

Goose, S. and Möller, C. (1999). A 3D audio only interactive Web browser: Using spatialization to convey hypermedia document structure. In *Proceedings of the seventh ACM international conference on Multimedia*, pages 363–371, Orlando, Florida, United States. ACM Press.

Gorny, P. (2000). Typographic semantics of webpages accessible for visually impaired users: Mapping layout and interaction objects to an auditory interaction space. In *International Conference on Computers Helping with Special Needs*, Karlsruhe.

Graham, I. (2002). *A Pattern Language for Web Usability.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Grice, H. P. (1975). Logic and conversation. *Syntax and Semantics: Speech Acts*, 3:41–58.

Grosz, B. J. and Sidner, C. L. (1986). Attention, intentions, and the structure of discourse. *Computational linguistics*, 12:175–204.

Hankinson, J. C. K. and Edwards, A. D. N. (1999). Designing earcons with musical grammars. *SIGCAPH Computers and the Physically Handicapped*, 1(65):16–20.

Harper, S. and Patel, N. (2005). Gist summaries for visually impaired surfers. In *ASSETS '05: Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, pages 90–97, New York, NY, USA. ACM Press.

Hart, S. and Staveland, L. (1988). Development of NASA-TLX (task load index): Results of empirical and theoretical research. *Human mental workload*, 1:139–183.

Hayes, G., Patel, S., Truong, K., Iachello, G., Kientz, J., Farmer, R., and Abowd, G. (2004). The personal audio loop: Designing a ubiquitous audio-based memory aid. In *Proceedings of Mobile HCI 2004: The 6th International Conference on Human Computer Interaction with Mobile Devices and Services.*

Helal, A., Moore, S., and Ramachandran, B. (2001). Drishti: An integrated navigation system for visually impaired and disabled. In *Proceedings of the 5th International Symposium on Wearable Computers.*

Hudson, S. E. and Smith, I. (1996). Electronic mail previews using non-speech audio. In *Conference companion on Human factors in computing systems*, pages 237–238. ACM Press.

Huron, D. (1991). Review of auditory scene analysis: The perceptual organization of sound. *Psychology of Music*, 19(1):77–82.

Inman, D. P., Loge, K., and Cram, A. (2000). Teaching orientation and mobility skills to blind children using computer generated 3-D sound environments. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Jacobson, R. (1998). Navigating maps with little or no sight: A novel audio-tactile approach. In *Proceedings of Content Visualization and Intermediary Representations*, Montreal, Quebec, Canada.

Kamel, H. M. and Roth, P. (2001). Graphics and user's exploration via simple sonics (GUESS): Providing interrelational representation of objects in a non-visual environment. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Kay, L. (1974). A sonar aid to enhance spatial perception of the blind: Engineering design and evaluation. *Radio and Electronic Engineer*.

Kennel, A. R. (1996). Audiograf: A diagram-reader for the blind. In *Proceedings of the second annual ACM conference on Assistive technologies*, pages 51–56. ACM Press.

Kline, R. L. and Glinert, E. P. (1994). UnWindows 1.0: X Windows tools for low vision users. *SIGCAPH Computers for the Physically Handicapped*, 1(49):1–5.

Kobayashi, M. and Schmandt, C. (1997). Dynamic Soundscape: Mapping time to space for audio browsing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201. ACM Press.

Koman, R. (1998). The scent of information: Helping users find their way by making your site "smelly". *Dr. Dobbs Journal*, 5(15):1.

Lai, J. and Yankelovich, N. (2003). Conversational speech interfaces. *The Human-Computer Interaction Handbook*, 1:698–712.

Lakshmipathy, V., Schmandt, C., and Marmasse, N. (2003). Talkback: a conversational answering machine. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 41–50. ACM Press.

Lemmens, P. (2005). Using the major and minor mode to create affectively-charged earcons. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Leplatre, G. and Brewster, S. A. (2000). Designing non-speech sounds to support navigation in mobile phone menus. In *Proceedings of the international conference on auditory display*. International community on Auditory display.

Lorho, G., Marila, J., and Hiipakka, J. (2001). Feasibility of multiple non-speech sounds presentation using headphones. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Lumbreras, M., Barcia, G. M., and Sánchez, J. (1996). A 3D sound hypermedia system for the blind. In *Proceedings of the first European conference on Disability, virtual reality, and associated technology.*

Lumbreras, M. and Rossi, G. (1995). A metaphor for the visually impaired: Browsing information in a 3D auditory environment. In *Conference companion on Human factors in computing systems*, pages 216–217. ACM Press.

Lumbreras, M. and Sánchez, J. (1999). Interactive 3D sound hyperstories for blind children. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 318–325. ACM Press.

Ly, E. and Schmandt, C. (1994). Chatter: A conversational learning speech interface. In *Symposium on Multi-Media Multi-Modal Systoms*, Stanford, California, United States.

Mankoff, J., Dey, A. K., Hsieh, G., Kientz, J., Lederer, S., and Ames, M. (2003). Heuristic evaluation of ambient displays. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 169–176, New York, NY, USA. ACM Press.

Marchionini, G. (1995). *Information seeking in electronic environments.* Cambridge University Press.

Marchionini, G., Geisler, G., and Brunk, B. (2000). AgileViews: A human-centered framework for information spaces. In *Proceedings of the Annual Conference of the American Society for Information Science*, pages 271–280.

Martin, P., Crabbe, F., Adams, S., Baatz, E., and Yankelovich, N. (1996). SpeechActs: A spoken language framework. *IEEE Computer*, 29(7):33–40.

McGookin, D. and Brewster, S. (2004a). Empirically derived guidelines for the presentation of concurrent earcons. In *BCS-HCI*, volume 2, Leeds, United Kingdom. BCS.

McGookin, D. K. and Brewster, S. A. (2002). Dolphin: The design and initial evaluation of multimodal focus and context. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

McGookin, D. K. and Brewster, S. A. (2004b). Space, the final frontearcon: The identification of concurrently presented earcons in a synthetic spatialised auditory environment. In *Proceedings of the international conference on auditory display.* International Community on Auditory Display.

Meijer, P. (1992). An experimental system for auditory image representations. *IEEE Transactions on Biomedical Engineering*, 39(2):112–121.

Mereu, S. W. and Kazman, R. (1997). Audio enhanced 3D interfaces for visually impaired users. *SIGCAPH Computers and the Physically Handicapped*, 1(57):10–15.

Midgley, L. and Vickers, P. (2006). Sonically-enhanced mouse gestures in the Firefox browser. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97.

Mitsopoulos, E. N. and Edwards, A. D. N. (1998). A principled methodology for the specification and design of non-visual widgets. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Mynatt, E. (1997). Transforming graphical interfaces into auditory interfaces for blind users. *Human-Computer Interaction*, 12:7–45.

Mynatt, E. and Weber, G. (1994). Nonvisual presentation of graphical user interfaces: contrasting two approaches. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 166–172. ACM Press.

Mynatt, E. D., Back, M., Want, R., Baer, M., and Ellis, J. B. (1998). Designing Audio Aura. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 566–573.

Mynatt, E. D. and Edwards, W. K. (1995). Metaphors for nonvisual computing. *Extraordinary human-computer interaction: interfaces for users with disabilities*, 1:201–220.

Nager, W., Teder-Sälejärvi, W., Kunze, S., and Münte, T. (2003). Preattentive evaluation of multiple perceptual streams in human audition. *Cognitive neuroscience and neuropsychology*, 14(6).

Nave, C. R. (2006). HyperPhysics: Sound and hearing. `http://hyperphysics.phy-astr.gsu.edu/HBASE/sound/soucon.html`.

Neuhoff, J. G. (2004). Auditory motion and localization. In *Ecological Psychoacoustics*. Elsevier Academic Press.

Nichols, J., Myers, B. A., Higgins, M., Hughes, J., Harris, T. K., Rosenfeld, R., and Pignol, M. (2002). Generating remote control interfaces for complex appliances. In *Proceedings of UIST*, pages 161–170, Paris, France.

Nielsen, J. (1993). *Usability Engineering*. AP Professional, Boston, Massachusetts, United States.

Nielsen, J. and Mack, R. (1994). *Usability Inspection Methods*. John Wiley and Sons, Inc., New York.

Novik, D. and Pérez-Quinoñes (1998). Cooperating with computers: Abstraction from action to situated acts. In *Proceedings of the North European conference on Cognitive ergonomics*, pages 49–54.

O'Donnell, M. (2002). Producing audio for Halo. In *Proceedings of the Game Developers Conference*.

Omojokun, O. and Dewan, P. (2007). Automatic generation of device user-interfaces? In *Fifth Annual International Conference on Pervasive Computing and Communications*, pages 251–261. IEEE.

Parente, P. (2004). Audio enriched links: Web page previews for blind users. In *ASSETS '04: Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility*, pages 2–8, New York, New York, United States. ACM Press.

Parente, P. and Bishop, G. (2003). BATS: The blind audio tactile mapping system. In *Proceedings of the ACM Southeast Regional Conference*, New York, New York, United States. ACM Press.

Parente, P. and Clippingdale, B. (2006). Linux Screen Reader: Extensible assistive technology. In *ASSETS '06: Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*, New York, NY, USA. ACM Press.

Peres, S. and Lane, D. (2003). Sonification of statistical data. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Petrucci, L. S., Harth, E., Roth, P., Assimacopoulos, A., and Pun, T. (2000). Websound: A generic web sonification tool and its application to an auditory web browser for blind and visually impaired users. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Pinelle, D. (2004). *Improving Groupware Design for Loosely Coupled Groups*. PhD thesis, University of Saskatchewan, Saskatoon, Saskatchewan, Canada.

Pirolli, P. and Card, S. K. (1999). Information foraging. *Psychological Review*, 106(4):643–675.

Pixley, T. (2000). Document object model (DOM) level 2 events specification. Technical report, W3C.

Raman, T. (1997). *Auditory User Interfaces*. Kluwer Academic Publishers, Boston, Massachusetts, United States.

Raman, T. (2001). Emacspeak: A speech enabling interface. *Dr. Dobb's Journal*.

Raman, T. V. (1996a). Emacspeak: A speech interface. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 66–71. ACM Press.

Raman, T. V. (1996b). Emacspeak: Direct speech access. In *Proceedings of the second annual ACM conference on Assistive technologies*, pages 32–36. ACM Press.

Raskin, J. (2000). *The humane interface: New directions for designing interactive systems*. Addison-Wesley, New York, New York.

Reitter, D., Panttaja, E. M., and Cummins, F. (2004). Ui on the fly: Generating a multimodal user interface. In *Proceedings of Human Language Technology Conference HLT/NAACL*.

Resnick, P. and Virzi, R. A. (1995). Relief from the audio interface blues: expanding the spectrum of menu, list, and form styles. *ACM Trans. Comput.-Hum. Interact.*, 2(2):145–176.

Rivenez, M., Darwin, C., and Guillaume, A. (2004). Perception of unattended speech. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Roberts, T. L. and Moran, T. P. (1983). The evaluation of text editors: methodology and empirical results. *Commun. ACM*, 26(4):265–283.

Roeber, N. and Masuch, M. (2005). Leaving the screen: New perspectives in audio-only gaming. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Ross, D. A. and Blasch, B. B. (2000). Wearable interfaces for orientation and wayfinding. In *ASSETS '00: Proceedings of the fourth international ACM conference on Assistive technologies*, pages 193–200, New York, NY, USA. ACM Press.

Roth, P., Petrucci, L., Pun, T., and Assimacopoulos, A. (1998). AB-Web: Active audio browser for visually impaired and blind users. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Roth, P., Petrucci, L. S., Assimacopoulos, A., and Pun, T. (2000). Audio-haptic Internet browser and associated tools for blind and visually impaired computer users.

Savidis, A., Stephanidis, C., Korte, A., Crispien, K., and Fellbaum, K. (1996). A generic direct-manipulation 3D-auditory environment for hierarchical navigation in non-visual interaction. In *ASSETS '96: Proceedings of the second annual ACM conference on Assistive technologies*, pages 117–123. ACM Press.

Sawhney, N. and Murphy, A. (1996). ESPACE 2: An experimental hyperaudio environment. In *Conference companion on Human factors in computing systems*, pages 105–106. ACM Press.

Sawhney, N. and Schmandt, C. (1998). Speaking and listening on the run: Design for wearable audio computing. In *Internation symposium on wearable computing*, Pittsburgh, Pennsylvania, United States. IEEE.

Sawhney, N. and Schmandt, C. (2000). Nomadic radio: speech and audio interaction for contextual messaging in nomadic environments. *ACM Trans. Comput.-Hum. Interact.*, 7(3):353–383.

Schmandt, C. (1998). Audio Hallway: A virtual acoustic environment for browsing. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 163–170, New York, NY, USA. ACM Press.

Schmandt, C. and Mullins, A. (1995). Audiostreamer: exploiting simultaneity for listening. In *Conference companion on Human factors in computing systems*, pages 218–219. ACM Press.

Schwerdtfeger, R. (1991). Making the GUI talk. *BYTE*, pages 118–128.

Shinn-Cunningham, B. (2000). Learning reverberation: Considerations for spatial auditory displays. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Shinn-Cunningham, B., Ihlefeld, A., and Schoolmaster, M. (2004). Selective and divided attention: Extracting information from simultaneous sound sources. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Shneiderman, B. (1992). *Designing the user interface: strategies for effective human-computer interaction*. Addison-Wesley Longman Publishing Co., Inc., 2nd edition.

Simpson, B., Brungart, D., Gilkey, R., Iyer, N., and Hamil, J. (2006). Detection and localization of speech in the presence of competing speech signals. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Skantze, D. and Dahlbäck, N. (2003). Auditory icon support for navigation in speech-only interfaces for room-based design metaphors. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Steel, A., Jones, M., Apperley, M., and Jehan, T. (2002). The use of auditory feedback in call centre chhi. In *CHI '02 extended abstracts on Human factors in computing systems*, pages 768–769. ACM Press.

Stockman, T., Hind, G., and Frauenberger, C. (2005). Interactive sonification of spreadsheets. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Sukaviriya, P., Foley, J. D., and Griffith, T. (1993). A second generation user interface design environment: the model and the runtime architecture. In *Joint conference of ACM SIG-CHI and INTERACT (INTERCHI)*.

Susnik, R., Sodnik, J., and Tomazic, S. (2005). Code of elevation in acoustic image of space. In *Proceedings of Acoustics*, Busselton, Western Australia.

Sweller, J., van Merrienboer, J. J. G., and Paas, F. G. W. C. (1998). Cognitive architecture and instructional design. *Educational Psychology Review*, 10:251–297.

Tan, D. S., Stefanucci, J. K., Proffitt, D. R., and Pausch, R. (2001). The Infocockpit: Providing location and place to aid human memory. In *Workshop on Perceptive User Interfaces 2001*.

Targett, S. and Fernström, M. (2003). Audio games: Fun for all? All for fun? In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Thatcher, J. (1994). Screen Reader/2: Access to OS/2 and the graphical user interface. In *ASSETS '94: Proceedings of the first annual ACM conference on Assistive technologies*, pages 39–46, New York, NY, USA. ACM Press.

Ueda, K., Nakajima, Y., and Akahane-Yamada, R. (2005). An artificial environment is often a noisy environment: Auditory scene analysis and speech perception in noise. *Journal of Physiological Anthropology and Applied Human Science*, pages 129–133.

Valkenburg, D. V. and Kubovy, M. (2004). From Gibson's fire to Gestalts: A bridge-building theory of perceptual objecthood. In *Ecological psychoacoustics*. Elsevier Academic Press.

van Welie, M. and van der Veer, G. C. (2003). Pattern languages in interaction design: Structure and organization. In *Ninth annual international conference on human-computer interaction*, Zürich, Switzerland.

Vanderheiden, G. C. (1990). Thirty-something (million): Should they be exceptions? *Human Factors*, 32:383–396.

Vargas, M. L. and Anderson, S. (2003). Combining speech and earcons to assist menu navigation. In *Proceedings of the International community on Auditory display*.

Vemuri, S., Schmandt, C., Bender, W., Tellex, S., and Lassey, B. (2004). An audio-based personal memory aid. In *Proceedings of Ubicomp 2004: Ubiquitous Computing*, pages 400–417.

Vickers, P. and Alty, J. L. (1998). Towards some organising principles for musical program auralisations. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Walker, B. N. and Lane, D. M. (2001). Psychophysical scaling of sonification mappings: A comparision of visually impaired and sighted listeners. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Walker, B. N., Nance, A., and Lindsay, J. (2006). Spearcons: Speech-based earcons improve navigation performance in auditory menus. In *Proceedings of the International Conference on Auditory Display*. International Community on Auditory Display.

Walker, B. W. and Kramer, G. (2004). Ecological psychoacoustics and auditory displays: Hearing, grouping, and meaning making. In *Ecological psychoacoustics*. Elsevier Academic Press.

Ware, C. (2000). *Information visualization.* Academic Press, San Diego, California, United States.

Whiteside, J., Archer, N., Wixon, D., and Good, M. (1982). How do people really use text editors? In *Proceedings of the SIGOA conference on Office information systems*, pages 29–40, New York, NY, USA. ACM Press.

Wickens, C. D. (1991). Processing resources and attention. In Damos, D., editor, *Multiple-task performance*, pages 3–34. Taylor & Francis, London, England, 1st edition.

Wickens, C. D. and Hollands, J. G. (2000). *Engineering psychology and human performance.* Prentice-Hall, Inc.

Wilson, T. (1997). Information behaviour: an interdisciplinary perspective. *Information Processing and Management*, 33(4):551–572.

Winberg, F. and Hellström, S. (2000). The quest for auditory direct manipulation: The sonified towers of hanoi. In *Proceedings of the 3rd International Conference on Disability, Virtual Reality and Associated Technologies*, pages 75–81, Reading, UK.

Winberg, F. and Hellström, S. O. (2003). Designing accessible auditory drag and drop. In *Proceedings of the 2003 conference on Universal usability*, pages 152–153. ACM Press.

Yankelovich, N. (1994). Talking vs. taking: Speech access to remote computers. In *Conference companion on Human factors in computing systems*, pages 275–276. ACM Press.

Yankelovich, N., Levow, G.-A., and Marx, M. (1995). Designing SpeechActs: Issues in speech user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 369–376. ACM Press/Addison-Wesley Publishing Co.

Zajicek, M., Powell, C., and Reeves, C. (1998). A web navigation tool for the blind. In *Proceedings of the third international ACM conference on Assistive technologies*, pages 204–206. ACM Press.

Zhao, H. (2006). *Interactive Sonification of Abstract Data - Framework, Design Space, Evaluation, and User Tool*. PhD thesis, University of Maryland at College Park.