

**NewsTime:
A Graphical User Interface to Audio News**

by
Christopher D. Horner


A.B., Computer Science
Harvard University
(1991)

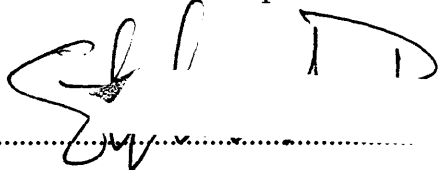
SUBMITTED TO THE
PROGRAM IN MEDIA ARTS AND SCIENCES,
SCHOOL OF ARCHITECTURE AND PLANNING
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 1993

© 1993 MIT All Rights Reserved

Signature of Author.....
Program in Media Arts and Sciences
May 7, 1993

Certified by.....
Christopher Schmandt
Principal Research Scientist, MIT Media Lab
Thesis Supervisor

Accepted by.....
Stephen A. Benton
Chairperson
Departmental Committee on Graduate Students

Rotch
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

JUL 12 1993

NewsTime: A Graphical User Interface to Audio News

by

Christopher D. Horner

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on May 7, 1993 in partial fulfillment of the
requirements for the Degree of
Master of Science

Abstract

A random scan through the radio dial through the course of a single day will demonstrate a stimulating variety of information programming. Speech is a natural channel for news because it keeps us informed while we do other activities: driving, office work, etc... Yet the ephemeral nature of speech underlies both its benefits and drawbacks. Although speech is an expressive communication medium, it is inherently slow, serial, and provides no immediate indication of what's ahead or what came before.

Current graphical user interfaces to stored audio can provide random access using familiar point and click paradigms. However, these interfaces do not address navigational issues for audio recordings that are very long, such as a newscast. What is needed is to offer visual feedback for different audio segments, such as where news, weather and traffic are located. Also needed is a way to maintain resolution despite increasing sound duration, and better methods to assist audio search. Finally, a facility for text support of audio files, such as annotation or transcripts is required.

This thesis describes a visual interface to audio news, "NewsTime", which addresses these shortcomings. News is a particularly interesting domain not only because it is inherently timely, but also because it demonstrates relatively structured speech, which is more amenable to a rich visual interface than unstructured conversation.

Thesis Supervisor: Christopher Schmandt
Title: Principal Research Scientist, MIT Media Lab

**NewsTime:
A Graphical User Interface to Audio News**

by
Christopher D. Horner

Thesis readers

Reader.....
Walter Bender
Principal Research Scientist
MIT Media Lab

Reader.....
Kate Ehrlich
Manager, Human Interface Group
Sun Microsystems

Acknowledgments

I would like to recognize the Media Lab community for its general support of this work, and in particular thank the following people:

My advisor, Chris Schmandt for encouraging my interest in this area, and his overall supervision of this thesis.

My readers, Walter Bender and Kate Ehrlich for their time and comments.

Barry Arons, Alan Blount, Karen Donoghue, Eddie Elliott, Mike Hawley, Debby Hindus, Charla Lambert and Jordan Slott for their direct contributions to this project.

Finally, I would like to acknowledge Sun Microsystems, and the member companies of the News in the Future consortium for their sponsorship.

Table of Contents

1	Introduction	7
	1.1 The Scenario	
	1.2 Why Audio?	
	1.3 Why News?	
	1.4 NewsTime Overview	
	1.5 Thesis Outline	
2	The SoundViewer Widget	14
	2.1 Introduction	
	2.2 The Original SoundViewer	
	2.3 Evolution of the SoundViewer	
	2.3.1 Audio Attribute Display	
	2.3.2 Prior Art	
	2.3.3 Distinguishing Speech from Silence	
	2.3.4 Semantic Content Display	
	2.3.5 Speed Control	
	2.3.6 SoundViewer Text Labels	
	2.3.7 Indicating SoundViewer Duration	
	2.3.8 SoundViewer Modes	
	2.3.9 Bookmarks	
	2.3.10 Audio Feedback Via a Visual Interface	
3	The MegaSound Widget	33
	3.1 Motivation: SoundViewer Limitations	
	3.2 Hierarchical Magnification of Long Sounds	
	3.3 Synchronized Text Annotations	
4	NewsTime	40
	4.1 Introduction	
	4.2 Recording Programs	
	4.3 NewsTime Window Synchronization	
	4.4 Next/Previous Speaker, Next/Previous Story	
	4.5 Segmentation of Television News	
	4.5.1 Closed Caption Synchronization	
	4.5.2 Closed Caption Results	
	4.5.3 Commercials	
	4.5.4 Keyword Search	
	4.5.5 Categorizing the News	
	4.6 Segmentation of Radio News	
	4.6.1 Music Detection for Next/Previous Story	
	4.6.2 Performance	
	4.6.3 Integrating Program Structure with Music	
	4.6.4 Results	
	4.6.5 Future Possibilities	

5	Implementation	63
5.1	Widgets' Technical Specification	
5.2	The OmniViewer Widget	
5.3	Chatfiles	
5.4	SoundViewer Hierarchy	
5.5	MegaSound Hierarchy	
5.6	Auxfiles	
	5.6.1 Speech and Silence database	
	5.6.2 Content database	
	5.6.3 Annotation database	
6	Conclusion	73
6.1	Goals Accomplished	
6.2	Predicting the Future	
A	Widget Terminology	76
B	Widget Documentation	78
B.1	Bboard Widget	
B.2	ChatMan Widget	
B.3	OmniViewer Widget	
B.4	MegaSound Widget	
C	Keywords for News Categorization	98
D	Description of NewsTime Auxiliary Programs	100
	Bibliography	103

Chapter 1

Introduction

1.1 The Scenario

Imagine a day in the near future when you approach your workstation to find that the entire day's news has been captured from a variety of audio sources, and assembled into a coherent whole. The news has been repackaged into sections, allowing you to scan the weather, traffic, local, national, international, and even business news as you see fit. While you accomplish other work, the news is read to you in the background. When the announcer starts reading news of little interest to you, say the weather in another country, you can either speed up the presentation or skip ahead to another section. Upon finding a segment of greater interest, you can mark it and annotate it for future reference. Although such interaction methods have long been possible for the traditional newspaper, an inherently visual and thus spatial medium, there has been little progress on how to map this interface style to sequential, time based media, such as stored speech.

As audio hardware becomes commonplace and the storage capacity of workstations increases, it will become possible to capture large quantities of speech for later browsing. For some, this may involve recording sounds from one's daily life, such as meetings or lectures. But for many, the natural application will be to capture existing sources of audio from the mass media for time-shifted listening, just as a VCR allows time-shifted television viewing. Despite advances in hardware however, the scenario described above is not realizable with current software. This thesis therefore investigates the four areas necessary to make this a reality:

- 1, Automatic gathering and segmentation of news recordings.
- 2, A more direct method for audio control via a visual interface.
- 3, A tool for magnifying different regions of a long recording.
- 4, A way of associating text annotations with audio files

1.2 Why Audio?

Interactive Voice Response (IVR) systems such as the movie-phone (333-FILM), or Amtrak's schedule line (1-800-USA-RAIL) demonstrate the recent success of audio information on demand. These systems provide a non-visual user interface, usually via the telephone touch pad, moving through a series of menus, to access brief snippets of recorded speech. Clearly there is a large market for prerecorded information on demand, but what makes these products successful is the precision of the audio interface. Although the database of recorded snippets as a whole adds up to a large amount of speech, users have the ability to find only those segments which have relevance to them. For example, a person calling Amtrak on a given day is given the option to listen to morning, afternoon or evening departures separately. The designers of the system realized that the longer an individual segment becomes, the less likely it will still be of interest.

Because such IVR systems are accessed over the telephone with the keypad, the highly structured "finite state machine" menu interface is appropriate. On the desktop however, a visual interface to audio can be infinitely richer, for instance by providing all menu choices in parallel. Although there is current work at the Media Lab on enhancing a purely audio interface to lengthy recordings [Aro93], as the duration of recorded audio increases, a visual interface which can let the user instantly find what they are looking for will be more efficient.

1.3 Why News?

Besides IVR systems, another readily available source of audio information literally surrounds us every day: the mass media. Just as IVR demonstrates the popularity of audio information on demand, so too has news on demand become an increasingly popular method of delivery. The success of CNN Headline News is a prominent example. However, there are other classes of news that lack a wide enough audience to be broadcast 24 hours a day. Time-shifted automatic recording of local news or traffic reports can make this information readily accessible, when it is not provided on demand by the broadcaster. The flexibility of the computer gives time-shifting even greater potential than on a simple VCR. Different types of news can be recorded from easily obtainable different sources (AM/FM radio, television audio) and be presented via one application.

Convenient time-shifting is not the only compelling reason for exploring the news domain. At any moment, television and radio airwaves are brimming with information programming, much of which we might find informative and much of which will be superfluous. Essentially, time-shifting gives the user a coarse grained filter for sifting through the entire broadcast spectrum at a program level. But this level of filtering will be inadequate for the active user who knows what interests him. Even a local newscast which is distinctively tailored for a particular geographic area will not entirely appeal to "all the people, all the time." After all, how many people in Boston really care about yet another fire in, say Chelsea? A newspaper reader can flip past such headlines at a glance, but someone listening to the 11 o'clock news has no such choice. A useful time saver would be an interface to help the user weed out such segments that are irrelevant, and highlight likely areas of interest.

Much work at the Media Lab has been done on filtering text based news to provide personalized electronic newspapers. NewsTime however tackles the filtering problem from a different angle. Rather than prior deletion of material from the news source, the newscast is recorded in its entirety, but then an enhanced user interface assists the user with accessing the news in a non-serial manner. As anyone who has used the fast-forward or rewind buttons on a VCR knows, this type of access is fundamentally different than listening to a broadcast synchronously. The fundamental premise of the NewsTime approach is that the user should perform the tasks that humans are best at, such as choosing the desired stories, while the computer does what it's best at, providing an efficient interface. Such an interface requires the ability to skip past an irrelevant segment, to quickly scan through an area for a particular segment, and to visually or textually mark a segment for later reference.

1.4 NewsTime Overview

This project can be divided into two main phases: news segmentation, and news presentation. Segmentation is important for audio newscasts because as a temporal medium, it is very difficult to locate and select individual stories without listening to a great deal of unwanted audio in between. Segmentation alone however does not overcome all the inherent linearity of audio. A pure audio interface to a perfectly segmented newscast would require one to sequentially skip from one segment to another, without any advance notice of what segment is coming up. Presentation becomes an important aspect of accessing recorded news because a visual interface is a spatial medium, which can overcome audio's linearity by indicating several segments in parallel.

The first objective of this thesis is to capture the news and segment it into semantically meaningful chunks, using whatever cues are available in the broadcast signal. Among these cues are: significant pauses that occur between speakers and stories, musical segues that are placed between program segments, and closed-captioned transcriptions of what is being said. Different combinations of these cues are used depending on their relevancy for different types of broadcasts.

The second objective is to display news recordings, taking advantage of any segmentation cues to provide random access. This is the job of the NewsTime application; it presents news from a variety of sources with one consistent interface, thereby giving audio news some of the same interface advantages of printed newspapers. The key issues for NewsTime are the display and browsing of large quantities of audio, and the integration of audio with text annotations.

NewsTime was written on a Sun SparcStation 2, in the X Window System (a.k.a. "X Windows"), and is built from a collection of interacting *widgets*. A widget is the technical term in X Windows for any class of interface component, such as a scrollbar or menu.¹

Figure 1.1 shows a representative NewsTime application window, with each main widget labeled. This example contains a typical ABC World News Tonight broadcast. Across the top of the screen is the menu bar, which lets one choose between various news programs, search for particular keywords in the news transcript, and skip from story to story, or speaker to speaker. The left-hand window which contains the transcript of the newscast (or program

¹ Since this thesis is concerned with visual interfaces to sound, all such interfaces are hereafter referred to as "audio widgets".

name if no transcript is available) is a collection of ordinary text widgets. The right-hand window contains a custom widget known as MegaSound, which was written specifically for NewsTime. MegaSound enhances the browsing of large audio newscasts in the four ways mentioned at the outset: automatic segmentation, more direct audio control, scale magnification, and text annotations.

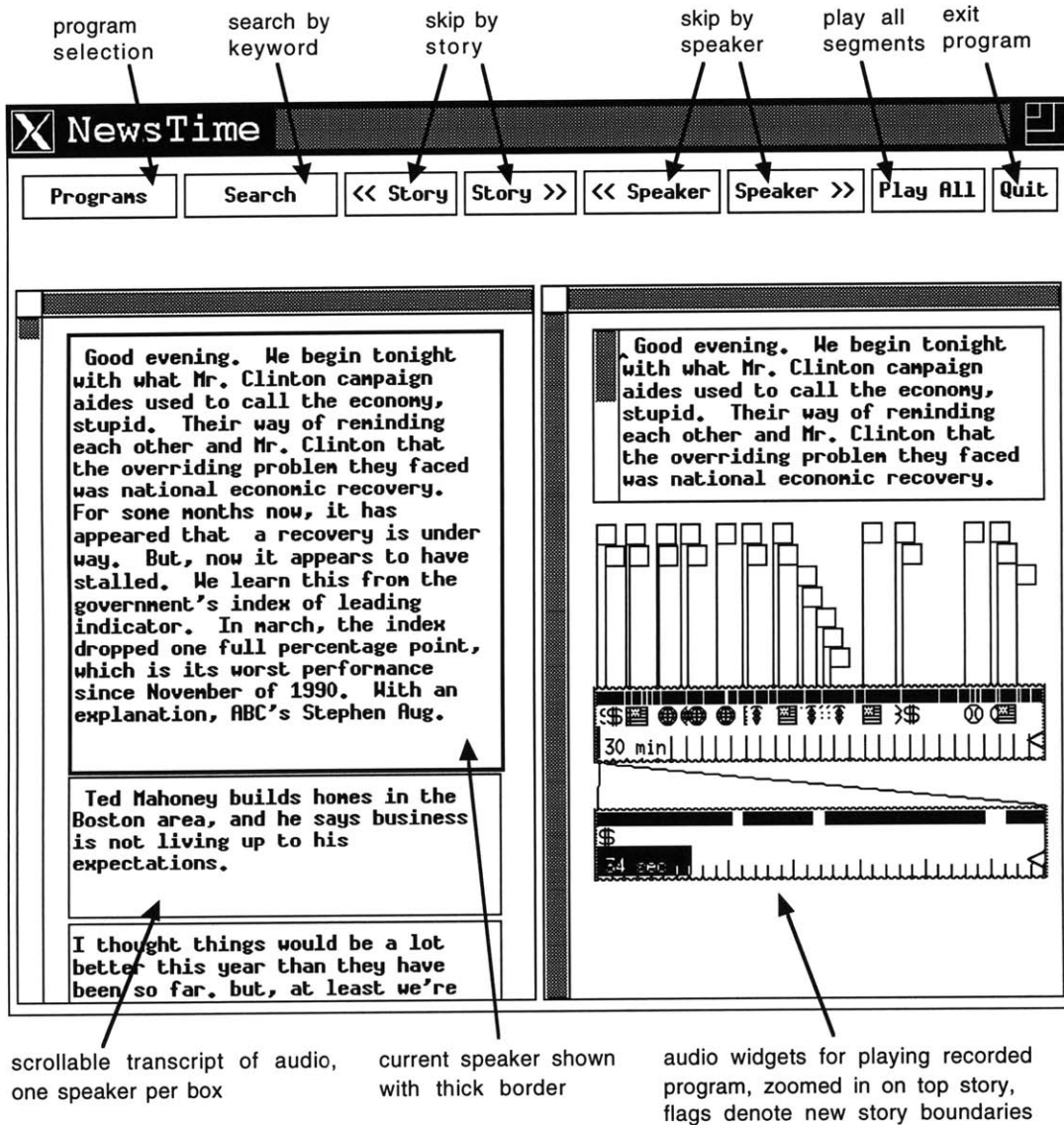


Figure 1.1: Overview of the NewsTime application

1.5 Thesis Outline

This thesis is written from the bottom up, first describing the interface of the widgets that comprise NewsTime, and later describing the NewsTime features *per se*. This facilitates an appreciation of how NewsTime itself was built from the bottom up using these widgets, and what the underlying issues were in building a widget to handle large quantities of audio.

Chapter 2 discusses earlier work in interfaces to recorded audio, both at the Media Lab and elsewhere. This chapter provides general background, and highlights the incremental improvements that were made to the Speech Group's existing audio interface. Chapter 3 describes the limitations to these incremental improvements and why the new widget, MegaSound was necessary. In chapter 4, the NewsTime application which makes use of MegaSound is examined at the feature level. This chapter also contains the techniques used for segmenting the news, and reports on their overall success. Chapter 5 discusses the technical details of how both the interface and application were produced. Finally, chapter 6 reflects back on the project as a whole. It explores future research directions, both in ways that NewsTime itself can be expanded, as well as suggesting related applications of this work.

Chapter 2

The SoundViewer Widget

2.1 Introduction

As mentioned in chapter 1, NewsTime utilizes the “MegaSound” widget. MegaSound is based on a simpler audio widget called the “SoundViewer”, which was originally developed by Ackerman and Schmandt at the Media Lab in 1987. This chapter discusses why the original SoundViewer was insufficient for dealing with the long audio recordings that were used in NewsTime. After detailing each improvement that was made to the SoundViewer to deal with some of these issues, it relates these improvements back to NewsTime in particular. Since SoundViewer enhancements alone did not solve all the problems unique to long recordings, chapter 3 discusses the MegaSound widget: an additional layer of interface on top of the SoundViewer that tackled the remaining problems.

2.2 The Original SoundViewer

A *sound file* (or *audio file*) is a Unix file which contains audio data. On the Sun SparcStation, sound files are usually stored in an 8 kHz, 8-bit mu-law format. A sound file can be arbitrarily long, so an entire newscast for instance will correspond to exactly one sound file, regardless of the program's duration. The SoundViewer presents a sound file as a bar delineated with one tick mark for each second of audio. With longer sound files each tick mark can represent more time, often a minute.

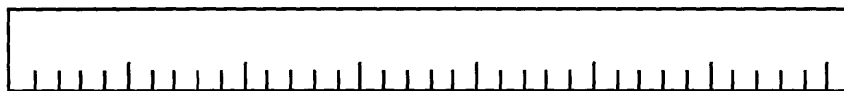


Figure 2.1: The Speech Group's SoundViewer

The SoundViewer first appears as an inert empty box, but a mouse click puts the SoundViewer into play mode. While the sound plays, the SoundViewer fills up with an indicator bar to denote position, resembling a car's fuel-gauge while being fueled up.

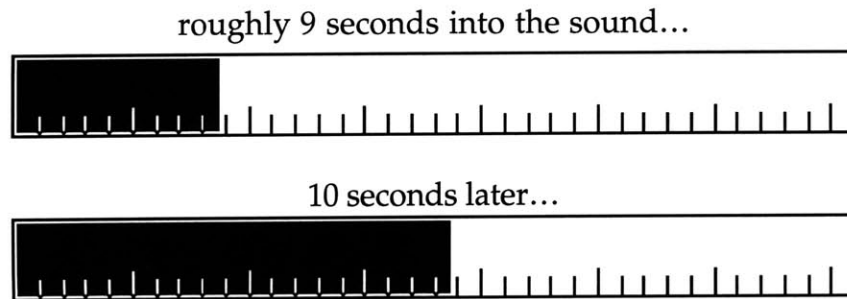


Figure 2.2: Time-lapse sequence of SoundViewer while playing

Myers found that such progress bars are essential to most computer users, to provide an assurance that something is indeed happening on their system, even while no other feedback is evident [Mye85].

The mouse that comes with Sun workstations has three buttons, and the SoundViewer makes use of all three. The left button acts as a play/pause control for the sound. The middle button can be used to click and drag the indicator bar to a new position, allowing random access. The right button can be used to click and drag out a selected segment of a SoundViewer, which can then be pasted into another application.

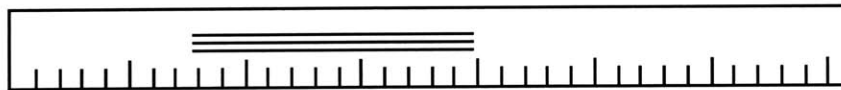


Figure 2.3: Selecting a SoundViewer region

2.3 Evolution of the SoundViewer

Because the SoundViewer is widely used in Speech Group code, the improvements motivated by NewsTime, outlined below, had a major impact.

2.3.1 Audio Attribute Display

A major drawback of the original SoundViewer was that it did not take into account the audio attributes of whatever sound file it was displaying. One sound file of someone slowly counting to ten would contain ten distinct segments of speech, while another file might be one continuous sound, perhaps a piece of music. If each sound was the same duration, the two SoundViewers that displayed them would be indistinguishable. The longer an audio recording gets, the greater the need is to map out audio segments in order to distinguish various parts of the file. My search through the literature found that none of the audio widgets previously developed sufficiently addressed the unique problems of visualizing large quantities of audio. With minor variation, these widgets shared the same basic interface: a simple linear timeline to allow random access, but no support for displaying content other than waveform based information. Waveform information is derivable from the audio signal itself, i.e. amplitude of the signal vs. time. This information is useful at a smaller scale, i.e. an individual story, rather than at the scale of an entire broadcast. For instance, one might want to visually locate a particular chunk of sound in order to select it for cut, copy and paste operations. Some earlier research projects provided insight on how the interface could be improved by displaying sound content.

2.3.2 Prior Art

Schmandt's "Intelligent Ear" [Sch81] was a digital Dictaphone which used signal magnitude to derive a waveform display of the recorded audio. The higher the average absolute magnitude of the signal, the higher the waveform appeared along the linear timeline, and the brighter its appearance. Although this provided a sense of how loud things were at a given point in time, news stories are generally reported in a calm, consistent speaking tone. Nevertheless, the general idea of displaying waveform derivable information is a good one, and the SoundViewer was given the new ability to display intervals of speech versus silence as alternating black and white bars respectively.

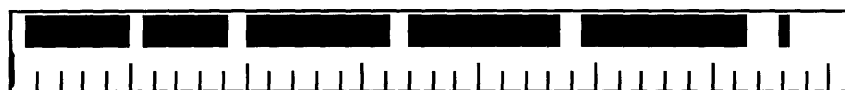


Figure 2.4: Display of speech and silence in a SoundViewer

This simpler binary display was used successfully in Xerox PARC's Etherphone [AS86]. In developing this office system, the authors argued that everyday users would not have any interest in viewing, storing or editing sound at the word, phoneme or even lower level. Rather, they noted that their users were interested in entire phrases or sentences. The threshold level to distinguish these was set so that natural pauses between sentences would be selected for. In NewsTime, the threshold was set so as to select for pauses between speakers and stories. The SoundViewer interface assisted in skipping between these segments as well. The space bar key was used to skip ahead, and the 'b' key jumped backwards. If the indicator bar was less than half a second after the beginning of the current speech segment, it would jump back again to the previous one, which prevented it from always jumping back to the beginning of the current one.

2.3.3 Distinguishing Speech from Silence

The speech/silence detection algorithm works by sampling the first 35 seconds of the file to get a sense of the average sound level in the newscast. The number 35 was chosen to be sufficiently large to get past pure theme music (which usually contains no silences) to a point where there was some speech and silence in the recorded programs. Fortunately news anchors do not change their dynamic range significantly during a program since they are paid to speak in an intelligible, consistent manner. Furthermore, even when the program changes from one reporter to another mid stream, it is the studio engineer's job to maintain consistent amplitude levels. From looking at the beginning of the sound file therefore, a threshold level for speech, T , can be derived for the entire program. Any samples of magnitude greater than T are labeled "speech" and any less are considered "silence."² Moreover, as in Etherphone, a silence had be of a minimum duration to be counted as significant. Any silence less than this minimum length was tacked onto the end of the preceding speech block. This prevents stop consonants in the middle of a sentence from breaking the continuity of the speech block.

Depending on the desired resolution of the speech blocks (i.e. phrase, sentence, or news story) that minimum silence duration can be set by the user. For voice mail, a value of 500 ms or even less is a good estimate, but for longer news casts, a value from 600-900 ms was more appropriate. Obviously the higher the number the more missed story boundaries there will be, and the smaller the number, the more "false alarms." This kind of tradeoff always exists in any decision procedure, and so by letting the programmer modify this value, flexibility is achieved depending on the application. There is some current work in the Speech Group on algorithms to compute what

² For a thorough review of speech and silence detection algorithms see [Aro92a].

the minimum silence window ought to be [Aro93], but for the purposes of this thesis, only the threshold value for minimum speech amplitude, T , is computed automatically. This is done by the following formula:

$$T = (0.08 \text{ MAX} + 2.92 * \text{MIN})$$

MAX refers to the maximum sample magnitude observed in the beginning of the file, while MIN is the minimum. Spurious pops or clicks which are due to noise are weeded out by throwing away any maximum sample which exceeds the prior sample by more than a certain ratio (25%). The formula above is based on Speech Group experience in recording voice mail over the telephone. It was found that the background noise level varied within a range of 5db or so. This implied that roughly $3.0 * \text{MIN}$ was a good threshold for determining the existence of any sound at all. Because both radio newscasts and telephone messages were recorded the same way, over a wire at the same sampling rate, a similar background noise level was assumed.

Measurements of the background noise alone however are not sufficient, because they do not take into account speech characteristics, but only the recording environment itself. Therefore further analysis on a variety of public radio news programs revealed that the sound of a reporter taking a breath between stories was at an amplitude approximately 8% of the full dynamic range of the recording, or $.08 * (\text{MAX} - \text{MIN})$. When this quantity is added to the $3.0 * \text{MIN}$ baseline, the formula given above for T is derived.

Given a threshold T for speech amplitude, and a minimum silence duration window, W , the speech and silence detection algorithm works by

finding the first silence block in the sound file, and then repeatedly finding the next silence block in the file, until there is no more audio data. Speech blocks are inferred between each silence block. The temporal resolution at which the boundaries of a speech or silence block are calculated is to the nearest 10 ms, which will be referred to as the *search interval* or *I*.

Rather than simply measure the amplitude of every search interval in the audio file, which takes too long, a faster method was used.³ To find the next block of silence from the current position, the routine first takes the average sample magnitude for one search interval, *I*. If the interval's average magnitude is greater than *T*, it means no allowable silence block of the minimum length *W* may contain this sample interval. Therefore, the routine skips ahead *W* ms in the sound file, and takes another 10 ms sample, hoping to fortuitously skip into the middle of the next silence block.

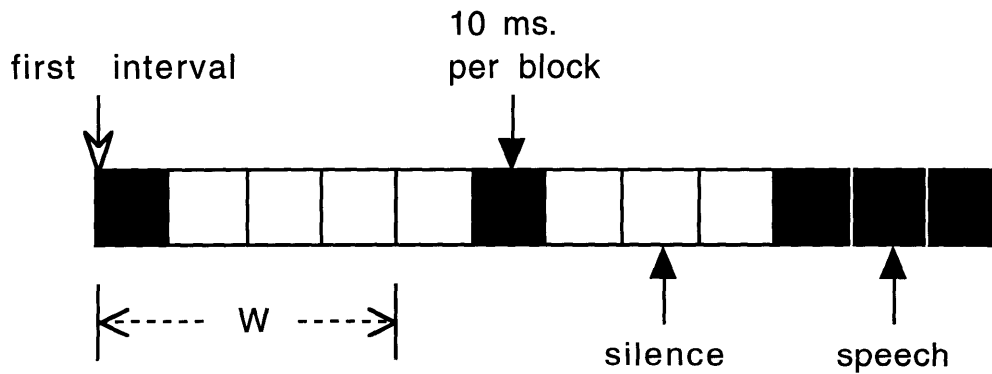


Figure 2.5: Skipping ahead to find next silence

The reason for this is that if the first interval is "speech" we can skip no further than *W* ms ahead without risking skipping over a salient window of silence. This worst case assumes the very next interval is "silence" which of course most of the time it will not be. At any rate, if it finds silence in this

³ The original speech/silence detection algorithm, written by Jordan Slott, was modified for this purpose.

new search interval, then a linear search is done to find the boundaries of the entire silence window. Assuming the total silence duration is larger than W , the function returns its location, otherwise the search for silence continues. This method works faster than a linear search of the entire file because assuming the silence window W is significant, silences will be few and far between. This means that average sample magnitude calculations can be ignored for large portions of the file. In reality though it still takes a significant amount of time to segment a large audio file, about 1 second per minute on a Sun SparcStation 2. For NewsTime therefore, a program called **paws**⁴ (a pun on the word “pause”) is called right after a news program is finished recording, which does this segmentation automatically. This makes the segmentation information available as soon as the application is run.

2.3.4 Semantic Content Display

Some high level structure is present in all speech: e.g. who is talking, what topics are being discussed, and so forth. There has been prior work in the Speech Group in semi-structured audio, for instance telephone calls, in which segmentation can be derived from turn taking detection, without using speech recognition [Hin92, HS92]. This thesis extends this work by utilizing the structures inherent in news programming: what program is on and where the story boundaries are. A person accessing large quantities of audio news obviously cares much more about this type of content information than what its waveform looks like. Waveform displays will not help distinguish the business news from the national news, since they look alike. Although speech and silence data are useful at the individual story level, for an entire newscast, segments must be displayed at a higher level of semantic and structural information. Unlike video, audio offers no such

⁴ A more thorough description of **paws**, and all other supplementary programs used by NewsTime is found in the appendix.

concept as a salient “still frame” to represent an entire segment. Instead, another straightforward way of displaying program information is to print it textually or iconically. Below is a simple example:

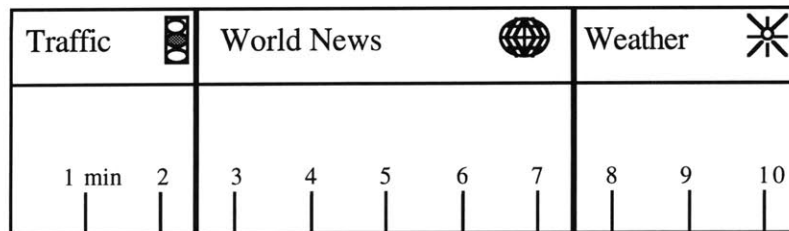


Figure 2.6: Initial design for semantic content display

This initial design was scrapped when it became clear that multiple text labels within the widget itself would overlap significantly for any real broadcast. This was evident after the label for sound file name and duration was added. Multiple text labels which are independent of the SoundViewer are instead incorporated separately as part of the MegaSound widget, which is described fully in the next chapter. Within the SoundViewer though, the iconic representation alone was used to display semantic content.

The binary display of speech vs. silence also inspired the presentation of semantic content in the SoundViewer. It was decided to have another layer of black and white bars drawn in below the speech and silence bars to distinguish between two types of audio.



Figure 2.7: Content bars displayed beneath segmentation layer

But while speech and silence was an obvious distinction for showing waveform derived characteristics, the question was whether or not a similar intuitive choice could be made to distinguish between two fundamental types

of higher level content? For instance, black bars could represent male speakers, and white bars female. Alternatively, black bars could indicate news programming, while white bars are commercials. Remember that the widget is not concerned about how these distinctions are made; it's job is simply to display the information which is provided from an outside source, whether human or computer generated.

I concluded that because the SoundViewer is a widget meant to be used across a variety of audio applications, some of which are unforeseen at the present time, that the default interpretation of the content bars was to distinguish between speech and non-speech audio. This way, the exact nature of the non-speech audio is determined by the application, and not the widget, which provides maximum flexibility for the programmer. In fact, the SoundViewer widget contains optional icon resources for extendibility. The first of these allows the programmer to specify an icon which represents the nature of the non-speech audio by overlaying the black bars.



Figure 2.8: Use of content icons with content bars

In the case of NewsTime, this icon was a musical note because the content database reflected the presence of speech vs. music. This is salient to news broadcasts because music is used frequently during commercials, and as a segue between major stories. An in depth analysis of the music detection algorithm will follow in chapter 4. Just as there were accelerators to skip to next or previous speech segment, NewsTime also maps accelerators onto the SoundViewer to skip to next music segment ('m'), or back to the previous music segment (shift-'B').

On top of this base layer of speech vs. non-speech content, additional icons can be overlaid to indicate even higher semantic levels of content, such as the topics under discussion.⁵ In NewsTime the icons were chosen to represent different types of news, e.g. weather, sports, and so forth, so that the widget could accept top level content description from NewsTime, and generate the appropriate visuals for a given news program. How this is actually accomplished, and what the icons used in NewsTime are described in chapter 4.

Note that although NewsTime is concerned with the presentation of audio news, news is representative of an entire class of speech domains which demonstrate structural layering: for instance lectures or meetings. Because the content database is extensible, and icons are not hardcoded into the widget, new icons representing different types of semantic content can be provided. This work will therefore be robust enough to allow other structured audio domains to be explored with the SoundViewer widget in the future.

2.3.5 Speed Control

Just as being able to skip from segment to segment increases the efficiency of listening to a long audio file, a more direct method for increasing speed of playback was added to the SoundViewer. Code was added to the **sound server**⁶ to change the default playback speed of audio files, anywhere from half of regular speed, to five times as fast. The playback speed is altered without modifying the speaker's pitch, which increases the intelligibility of the altered speech.⁷ One can comfortably comprehend normal speech played

⁵ The idea for a base content layer overlaid by additional content icons was given to me by Karen Donoghue of the Visible Language Workshop.

⁶ The **sound_server** is the background program that the SoundViewer makes all requests to for the playing of audio files. See [Aro92b] for details.

⁷ For a detailed discussion of various methods that accomplish this, refer to [Aro91].

back at twice normal speed, and when there is no pitch distortion it is possible to detect speaker changes at even higher speeds. The SoundViewer maps the entire row of number keys along the top of the keyboard to speed control keys. Keys '2' through '9' go from 0.5 to 2.5 times normal speed in 0.25 increments. The '0' and '1' keys both return to normal speed. To go beyond 2.5x the '+' and '-' keys also change the speed by + or - 0.25 relative to the current speed. In NewsTime, these speed controls make it very easy to rush through uninteresting parts of a broadcast, to a more interesting segment that is expected nearby. For instance, hitting the '9' key during a pledge break on public radio, then the '0' key the split second it ends is very convenient.

2.3.6 SoundViewer Text Labels

A major drawback of the original SoundViewer was that it did not take into account the attributes of the sound file it was displaying, in particular the file name. One SoundViewer might contain Beethoven's Ninth Symphony, and another a news report from Peter Jennings of ABC. Even if the sound files were named "beethoven9" and "jennings", the old SoundViewer could not indicate the difference. Applications that wanted to print sound file names had to do so separately, as in the **sbrowse** application that displays a SoundViewer for each sound file in a directory.

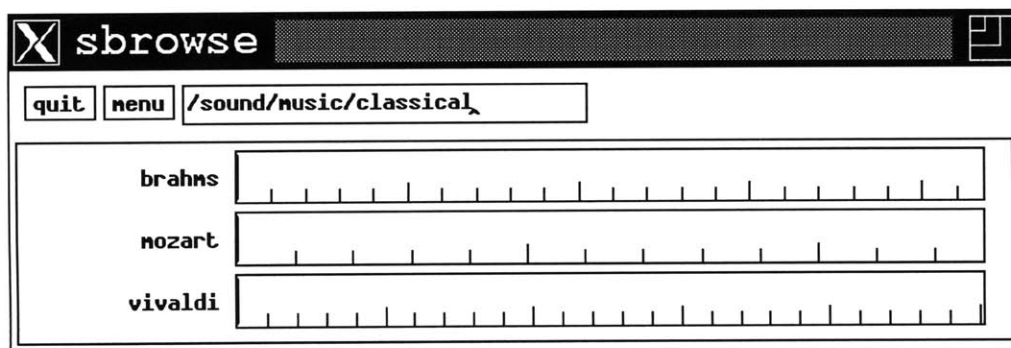


Figure 2.9: The **sbrowse** application

It was incumbent on the **sbrowse** application to annotate the SoundViewers with a separate text field, indicating what sound files they are associated with. One obvious enhancement to the SoundViewer therefore was the ability to show sound file names directly, without requiring extra code from the application writer.



Figure 2.10: Placing text labels in SoundViewers

The ability to print a text label directly in the SoundViewer does not have to be limited to sound file names either. The SoundViewer is meant to be a reusable module across any number of audio applications, some of which might need something besides the file name printed. Therefore, the label defaults to containing the sound file name but can be overwritten with any text string. This might have utility in simplifying the look of **vmail**, the Speech Group's desktop interface to voice-mail [Sti91].

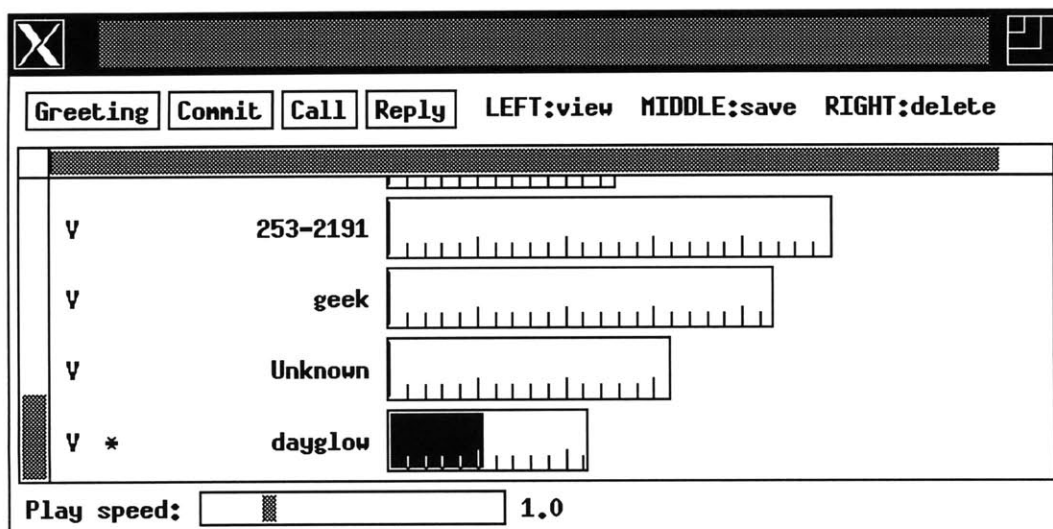


Figure 2.11: The **vmail** application

In **vmail** the caller's name or phone number is displayed if known, and not the sound file name. This is because the sound files in **vmail** are assigned fairly cryptic names by the recording program. Again, rather than printing the caller's name or number in a separate text widget, it could place it directly in the label field of the SoundViewer.

2.3.7 Indicating SoundViewer Duration

Another shortcoming of the original SoundViewer is that it was hard to tell sound duration because the user did not know whether a mark represented a second, a minute, or something else. The SoundViewer authors initially dealt with this problem by making the hash marks that corresponded to a minute taller than those corresponding to a second, and so forth. Here we see ten seconds of sound versus ten minutes:

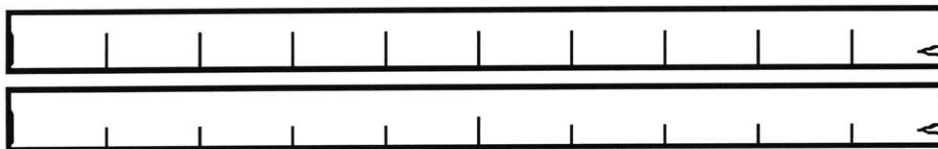


Figure 2.12: Ten minutes vs. ten seconds in a SoundViewer

In practice, users found it very difficult to quickly and reliably distinguish between the two cases, and so hash marks by themselves were deemed an inadequate way of displaying sound duration (although hash marks were kept in the SoundViewer as an option). As sound files get larger, this could lead to a hazardous situation, in which half an hour of sound is purged accidentally by a user thinking it's only a trivially brief snippet. Having earlier added the text label capability to the SoundViewer, I decided the duration could simply be printed in this label, using the appropriate level of detail: "549 msec", "53 sec", "1 min 20 sec", "15 min."



Figure 2.13: Printing duration in SoundViewer label

2.3.8 SoundViewer Modes

The most fundamental SoundViewer attribute is how it scales time to fit within its boundaries. The SoundViewer can be set to either fixed, truncated, scaled or “best” mode. A fixed mode and truncated mode widget take up a constant amount of screen space per unit of time. The difference is that a fixed mode widget always fills up the same amount of total screen space, leaving blank any portion not filled by sound. A truncated mode SoundViewer however only takes up as much space as necessary.



Figure 2.14: Truncated vs. fixed mode

A scaled SoundViewer always has the same width, but scales the amount of screen space per unit time to fit. The “best” mode will create a SoundViewer that is truncated, unless the duration exceeds some maximum time (specified by the application), beyond which the widget is scaled. When a SoundViewer in best mode exceeds the default maximum duration, the SoundViewer scales itself accordingly, and denotes this scaling by printing a ‘<’ symbol in its right margin.



Figure 2.15: “Best” mode showing scaled sound mark, ‘<’

2.3.9 Bookmarks

Over the years we have become accustomed to consumer technologies for dealing with large amounts of audio, e.g. the LP, the cassette, and the CD. A common problem with these technologies is the lack of a suitable interface to find and mark just the segment you are looking for. You might generally know where the interesting sound bites are in a long speech, but there is no easy way to “point” to a particular one and automatically jump to play from that location. Although many playback devices have a sequential counter, most users can only remember to zero it at a position of interest, which is then lost when the tapes or disks are switched.

Degen et al. built an improved “tape recorder” that let the user mark multiple points during playback [DMS92]. Their visual interface to these recordings, the SoundBrowser, would point out the location of these markers either with an arrow or an arrow with the word “todo” underneath (corresponding to items in a things to do list). Arbitrary textual annotations were not allowed. This was because their audio source was always personal recordings made on the portable recorder, which had two distinct marker buttons. Users found that playback via random access to these markers was one of the most useful qualities of the visual interface. They further found that visual bookmarks were sufficient, making voice annotation unnecessary.

Even a simple bookmark with no associated text can still be very helpful. A valuable application is to help people manually segment large audio streams. Instead of laboriously stopping to note points of interest, the audio can playback at high speeds, while the user types an annotation key every time a mark is desired. This capacity was added to the SoundViewer via the ‘^’ key. Each time this key is pressed, a bookmark is added at the current position, and the SoundViewer updates itself by drawing a '^' at the

note's location. In the NewsTime application, these marks are automatically generated to indicate new story boundaries.

2.3.10 Audio Feedback Via a Visual Interface

The original SoundViewer's manual interface, although easy to use, was insufficient for scanning. Recall that a left click is a play/pause control, while the middle mouse button moves the indicator. Although the user's ability to set position at random is useful, there is no audio feedback while setting this position. While the user holds down the middle button, and drags the indicator bar, sound play is disabled until the button is released, at which point playback resumes normally.

The problem with this method is that it makes scanning very difficult. Imagine the user wants to find a brief phrase embedded somewhere within a larger file, say the name of a particularly hot stock somewhere within the Wall St. report. The user must drop the indicator at random points, resume playing, and unless they hear what they wanted, they must pick it up and try it again. With lengthy audio files, all this button clicking and skipping around at random becomes very tedious. What was needed was a more direct scanning method, which would map mouse motion events to play commands. As the user clicks and drags the position indicator, there would be accompanying audio feedback. This interface paradigm is called direct audio manipulation via the visual display.

Such direct audio manipulation was used in Elliott's Video Streamer [Eli93], but audio was always played back at normal speed. If a user scanned faster than this, portions of the audio were skipped. MegaSound works in a similar fashion, but with some enhancements. First of all, if the user sweeps the mouse too quickly such that a skip in playback would occur less than 250

ms after the prior play command, that mouse action is skipped. This avoids a flood of play commands that interrupt each other so quickly that no intelligible sound is heard. Experimentation determined that 250 ms was still insufficient to actually make sense of what people were saying, but was sufficient to yield the overall acoustic characteristics of the sound. That is, one can tell if a man or woman is speaking, whether music is playing, or when a speaker change occurs. This is particularly useful for finding the transitions between news stories. When one wants to hear the sound play uninterrupted from the current position, one need only stop moving the mouse, take a brief listen, and then continue scanning by moving the mouse again. The total number of mouse clicks needed to find a particular spot is reduced from an undetermined number to a constant of 1.

The new SoundViewer also improves on the speed control approach taken in Degen et al's SoundBrowser. The SoundBrowser uses time scaling code which can speed up or slow down audio playback without changing the pitch of recorded voice. But whereas the SoundBrowser had a separate speed control, the SoundViewer includes speed control as part of the scanning interface. The faster the dragging motion, the faster the audio plays in order to cover the region swept out. If the user drags too fast some portions may still have to be skipped over, but continuous scanning is possible up to 5.0 times normal playback speed. On a backwards drag, the audio jumps back to the indicator position, plays a split second of audio normally at the given position, and then quickly jumps back again to keep up with the user's motion. This has the benefit of simulating the feel of a real world audio device such as the shuttle control on a VCR. However, it has the added advantage of being able to play faster or slower than normal without changing pitch, and of allowing comprehension during backwards play.

The reason enhanced scanning is beneficial for NewsTime is that any segmentation heuristics will not be perfect, and may leave the user in the middle of a story they wanted to skip to. In this situation it will be valuable for the user to quickly sweep back, listening for the audio cues that signal the beginning of the story. If the computer didn't skip far enough ahead, but the user senses the end of a story is near, they can conversely sweep forward rapidly. The combination of imperfect segmentation heuristics with a robust scanning interface makes it possible for users to quickly locate story boundaries.

Chapter 3

The MegaSound Widget

3.1 Motivation: SoundViewer Limitations

The improvements to the SoundViewer described in chapter 2, while necessary for the NewsTime interface were not sufficient by themselves. The SoundViewer is effective in a variety of sound based applications (e.g. voice-mail, things to do lists, etc...) in which the segments of audio are brief. But when scaled up, the SoundViewer model breaks down. To pack an hour's worth of audio into 120 pixels (a reasonable widget size), the SoundViewer would achieve resolution of 30 seconds per pixel. This leads to several problems: first is the breakdown of the selection mechanism. Even if you had the dexterity to drag the position indicator one pixel at a time, it would be impossible to pinpoint and select an interesting sound bite in less than 30 second increments. Chances are that the segments a user would want to select and save would be only a few seconds long: an important 800 number, the name of a good new movie, etc... The related scaling problem is that at 1 pixel per 30 seconds of real-time, the SoundViewer appears stagnant even while active. Simply changing the scale of the mapping, by zooming in and out, is a standard but unsatisfactory solution. By filling up the zoom window with only the current 30 seconds of interest, you lose a global sense of where you are in the sound, i.e. you can get lost.

The problem of showing the content of a sound by visual inspection is closely related to the problems of scaling up a SoundViewer's duration. Displaying speech and silence intervals as black and white bars is useful for brief segments, but upon scaling upwards this representation becomes too dense to convey much meaning. If 30 seconds is represented by only a single pixel, there may be many speech and silence segments that pixel is responsible

for. Such resolution limitations are intrinsic to the SoundViewer, and so a solution extrinsic to the SoundViewer was required. The solution was to create an additional interface layer above the SoundViewer, known as the MegaSound widget. MegaSound was implemented as a sound magnifier, which is able to expand and compress pieces of the audio stream on screen, without losing global positioning information.

MegaSound also generalizes the concept of bookmarks in the SoundViewer, by turning them into full fledged annotations, into which the user can place arbitrary text. This chapter describes both the zooming and annotation features of MegaSound.

3.2 Hierarchical Magnification of Long Sounds

Following the example of Mills et al. [MCW92], a hierarchical magnifying lens that can view individual portions of the sound to increasing levels of detail was used. This solves the problem of magnification without loss of overall context, because the hierarchical structure is maintained on screen. A MegaSound widget is really just two SoundViewer widgets with connecting lines that show the zoom region in its global context, and give a rough idea of its duration. Recall that the duration of the zoomed SoundViewer can be written into its text label directly, in case an exact measurement is desired. To make this possible, the SoundViewer widget first had to be manipulated to show only a zoomed portion of a sound file rather than the entire file.

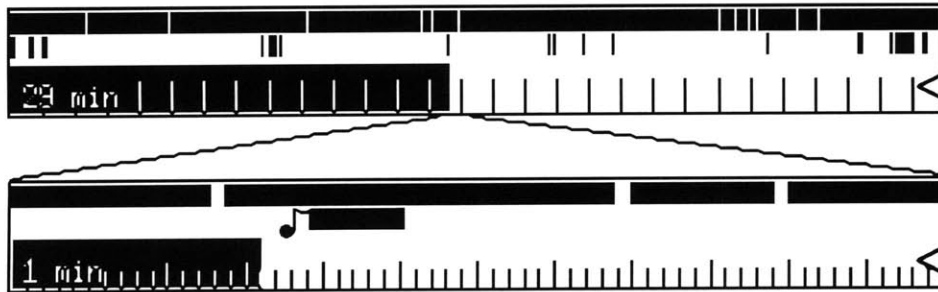


Figure 3.1: A MegaSound widget with both root and zoom levels

Because MegaSound incorporates the SoundViewer widget directly, all of the existing SoundViewer behaviors are preserved, such as speed control, bookmarks, skipping and scanning. Because the user can interact with either the root level or the zoomed SoundViewer, it is MegaSound's job to maintain the link between the two layers, so that the position indicators are synchronized. If either SoundViewer begins playing, the other will move its indicator in tandem. Also, if the user sets the root's indicator to a position beyond the boundaries of the current zoom region, either through direct mouse motion or through the keyboard accelerators, a new zoom region starting at the root's position replaces the old. Likewise, a change of position in the zoomed region will be reflected at the root level.

The zoom region can be set using the SoundViewer's selection mechanism: the desired region is dragged out with the third mouse button, either in the root or zoomed SoundViewer, then the 'e' key (for "expand") is pressed to zoom the selected region. When the user simply wants to browse through a large recording, MegaSound offers an automatic zoom mode. The default mode chooses a zooming region of one minute (although this can be modified), so that when the global play position is set at the top level, the zoom region changes to indicate the nearest minute. The zoomed sound begins playing, and when the full minute of audio has been played, the next minute is zoomed in on, and playing resumes. In this way the casual user

can listen to the entire recording with only one mouse click, but can also stop at any point for fine control instant replay.

Mills' interface was a Hierarchical Video Magnifier, not an audio magnifier. He could use still frames to represent content. Although this cannot be done with audio, the speech and silence bars and content icons of the SoundViewer serve this purpose. When multiple segments or content icons are clustered together at the top layer, the zoomed layer can reveal each segment or icon separately. In NewsTime, story and speaker boundaries are provided from an external source, which allows the application to magnify individual stories, one after the other. When an individual story is expanded, the finer resolution of speech and silence segments allow scanning at the phrase level. On a recent broadcast of ABC's World News Tonight, stories varied in length anywhere from 7 seconds to 5 minutes. If a constant zoom factor was hardcoded into the MegaSound widget, it would not be appropriate for different types of news which have different segment lengths. By allowing the MegaSound widget to magnify arbitrary regions of sound, this problem is avoided.

3.3 Synchronized Text Annotations

Just as the ability to place a text label in the SoundViewer was very helpful for identifying the general nature of the sound, being able to link text to specific points within a segment is likewise advantageous. The Intelligent Ear demonstrated one way that text could be integrated into an audio widget. In this case a speech recognition technique known as keyword spotting was used; so that if a keyword from a known set was recognized, it would be printed at the spotted location underneath the waveform display. Unfortunately, algorithms for such automatic keyword spotting are not robust enough to be used in NewsTime. Nevertheless, assuming that such a

mapping from the text spoken to the time spoken can somehow be provided independently, MegaSound has the capability to indicate salient portions of the text by “flagging” parts of the audio display.⁸

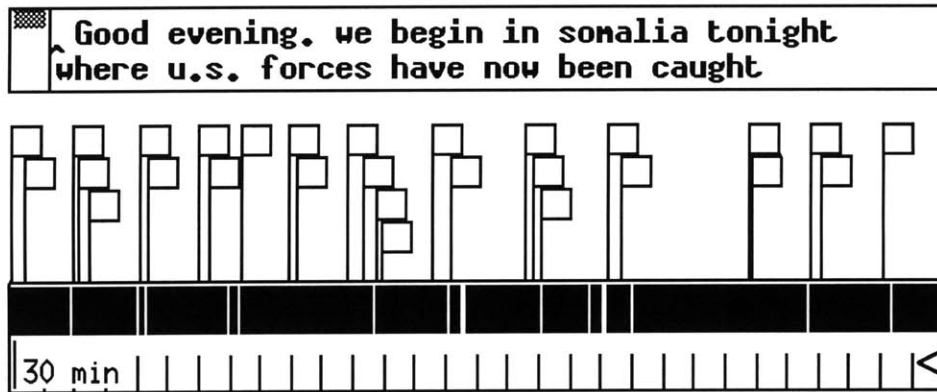


Figure 3.2: A non-zoomed MegaSound widget with annotation flags

Each flag indicates a text annotation linked to a particular time offset in the audio file. The MegaSound widget can separately initialize each flag to be visible or invisible, and can toggle this state whenever the controlling application needs to. As the pointer is moved into a given flag, its associated text is automatically displayed in the main note window of the MegaSound widget, and the sound's indicator automatically jumps to the flag's position. This allows rapid scanning of linked text and also facilitates editing. Clicking on a flag sets the zoom region of the MegaSound widget to the temporal boundaries for that flag, and plays the associated audio.

The Intelligent Ear only highlighted individual keywords but there is no reason to limit MegaSound in this regard. The text associated with a single flag can be up to 2048 characters, which corresponds to about two pages of printed text. There is no arbitrary limit on the total number of flags. This provides support for a full transcript to accompany the audio, which has

⁸ How this time-stamped text is actually generated for NewsTime will be taken up in Chapter 4, while the file format used is described in Chapter 5.

many potential uses even beyond the news domain. For example, given the MegaSound widget, a trivial application would be the presentation of a text translation while a story is read in another language. The user could control the audio playback by clicking on the text portion they want to hear, thus enhancing language learning. Recall that the widget is not responsible for figuring out this concordance of text and speech. It must come externally from another source and be stored within an associated auxiliary file. Besides text annotations, auxiliary files contain the segmentation and content information used by the SoundViewer. Auxiliary files are kept separate from their associated sound files so that they can be stored, shared, and modified more easily. The exact format of these auxiliary files will be deferred until chapter 5.

This thesis follows the example of Xerox PARC's Etherphone which let the user type arbitrary text annotations for a specified piece of sound. Although the initial text annotations are read in from the auxiliary file, MegaSound allows them to be modified at any time, either directly by the user or by the parent application. For annotations, the only information that is needed is start and stop offsets for the segment, and the text itself. The main note window of the MegaSound widget provides both read and write access, such that when the enter key is pressed, the active note is updated with the contents of that window.

The annotation flags are also related to the SoundViewer's simple bookmarks. When a bookmark is placed in a SoundViewer, it is flagged as an initially empty text annotation by the MegaSound widget, and becomes available for text editing. The ability to associate annotations with sound files has utility in semi-structured domains besides news. Although these domains are beyond the scope of this thesis, the MegaSound widget lays the

groundwork to make their exploration possible. For instance, during a meeting which is being recorded, as someone transcribes the minutes on-line, or clicks off agenda items, timestamps could be used to produce annotations describing the topics.

Chapter 4 NewsTime

4.1 Introduction

NewsTime is an X Windows application which utilizes all the features of the MegaSound widget previously discussed. It does this in order to facilitate the interaction of the average user browsing through the latest news.

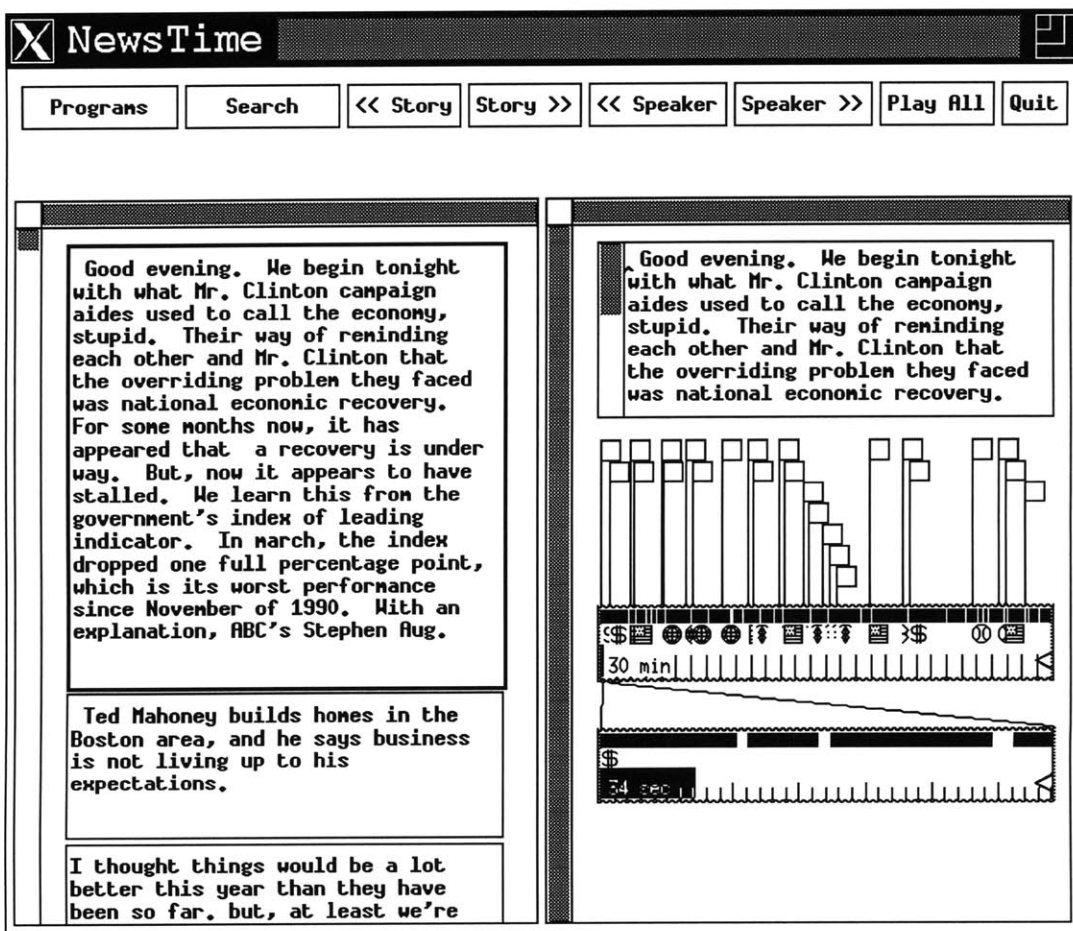


Figure 4.1: The NewsTime application

Here again is the ABC World News Tonight broadcast from the introduction. In the left-hand window, the transcript of the newscast is shown, broken down into individual text boxes, each of which represents a

new speaker. Clicking on any of the text boxes causes it to become active, i.e. highlight its border, and to have the related audio played. Likewise, in the right-hand window, a MegaSound widget is used to access the audio portion of the broadcast directly. The annotation flags default to being visible only at each new story boundary, rather than at every speaker change. As the user quickly sweeps the pointer through the flags, the headlines of each story can be rapidly scanned. At any time, one can decide to listen to the full story by clicking on the flag. This interaction method was extremely efficient at browsing a newscast for the first time, as well as going back to find an earlier story of interest. It maps very well onto the real world process of flipping through the pages of a newspaper.

4.2 Recording Programs

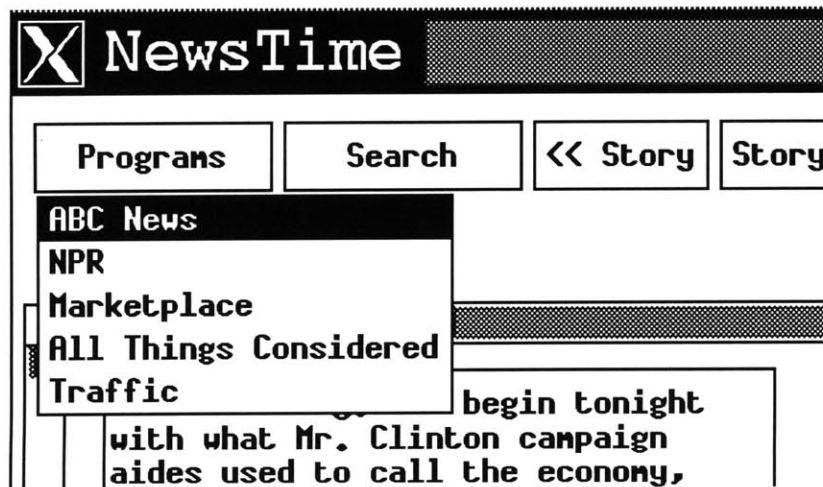


Figure 4.2: The "Programs" menu

The "Programs" menu simply lets one select between the various recordings available. Each of these programs, with the exception of Traffic reports and World News Tonight, was recorded live off of a cheap portable radio in my office, tuned to a local National Public Radio (NPR) affiliate, WBUR. The radio output was plugged directly into the microphone port of my Sun SparcStation 2. WBUR was chosen because it was a single station

with a wide variety of programs. Everything from business news (Marketplace), to national news, to more general interest information programming (All Things Considered) is available. This meant that the radio could be tuned into one station, and then left alone, which greatly facilitated the automatic recording process. An added benefit of NPR is their lack of commercials, and the guarantee that everything being recorded is at least newsworthy to begin with. Traffic information came from a different station, WBZ radio, and so was recorded off a separate NeXT computer with its own radio receiver. These traffic recordings were accessible to my SparcStation via a local area network (LAN), and were compatible with the audio file formats on my Sun workstation. ABC World News was also recorded on a NeXT computer and likewise made accessible on the LAN.

The "Programs" menu is possible because news is a reliable source of large quantities of speech audio in which signal processing is not needed to distinguish the overall program category. Regular program schedules are sufficient to yield top level content information, i.e. what type of news was recorded (traffic, business news, news updates, etc...). Fortunately, the selected radio news sources were predictable enough that an accurate recording schedule could be provided.

<u>Station</u>	<u>Program</u>	<u>Type of News</u>	<u>Duration</u>	<u>Frequency</u>
WBUR FM	Marketplace	Business	30 min	Once/day
"	All Things Considered	Misc. Reports & Interviews	60 minutes	Once/day
"	NPR News	National	5 minutes	Hourly
WBZ AM		Traffic	2 minutes	10 min.

Table 4.1: NewsTime program schedule

As you can see, a wide variety of recording strategies must exist for the different types of news depending on the frequency and duration of recording. Program durations last anywhere from a couple of minutes to a full hour.

Frequency of updates can be anywhere from once each hour to once per day. Recording was done automatically using a Unix **cron** job, which allowed for this kind of schedule flexibility.

NewsTime provides only up to date information at any given time. When an entirely new report comes in, as in the NPR news updates or the Marketplace program, the new report completely overwrites the old one at record time. However, when the news comes in frequent incremental bursts of new information, it is necessary to store a history. The traffic reports for example are given by a helicopter pilot who samples traffic congestion at different locations ten minutes apart. In this situation, NewsTime keeps a running buffer of the last six traffic reports and presents them all at once. The “Play All” button in the NewsTime menu is used to make all these reports play through automatically in sequence, without requiring the user to click on every one.

4.3 NewsTime Window Synchronization

Just as the MegaSound widget concurrently displayed annotations synchronized to the audio position, so too does NewsTime synchronize its left-hand window, containing the newscast transcript, along with its right-hand window containing the MegaSound widget. ABC World News Tonight was the only program for which a full transcript could be automatically generated. This transcript was created by one Unix process, **getcaption**⁹, which read the raw ASCII closed-captioning signal off a line 21 video decoder box, while the audio track of the program was recorded by a separate process,

⁹ The original **getcaption** program was written by Alan Blount, along with a set of filtering utilities: **remnull**, **grok**, **remhdr**, **capper**, **firstwordCAP** & **specialCAP** which make the captions easier to read. Following this, another program **cap2db** (written by myself) was run to create a database file for the soundViewer, containing timestamps for each text block.

`rec30min`¹⁰. Because both processes were triggered at the same time by a single Unix shell script, it was possible to roughly synchronize the two, (the method for doing this is described shortly). Given this time-stamped transcript, NewsTime scrolls through the closed captions in real-time, at the rate they were originally recorded. This scrolling is not line by line smooth scrolling as it was on the original closed captioning display, but rather scrolls an entire speaker block at a time. This however is a favorable feature as it allows one to read ahead of the announcer.

4.4 Next/Previous Speaker, Next/Previous Story

Although program schedules will imply the general category of a newscast, within individual programs a major challenge was how to segment the broadcast. This thesis does not intend to solve the problem of understanding what is being said in an audio recording, independent of a closed-caption transcript. Nevertheless, news exhibits enough structure that with a little more background knowledge, heuristics can provide a rough level of intra-broadcast segmentation.

Resnick and Virzi [RV92] implemented a method for skipping and scanning through hierarchically structured audio in the form of voice menus. Their system in which the user controlled the presentation of menu items, rather than having menus read automatically, sped up access times considerably. The main reason was that their system allowed users to skip over familiar, unwanted items to find the right one. NewsTime also provides skip ahead functionality which operates at either the story or speaker level. The remainder of this chapter examines the heuristics NewsTime uses to implement the Next/Previous Speaker, and

¹⁰ `rec30min` was modified from a demo program included with the NeXT computer, `recordfiletest`, which records audio in the background.

Next/Previous Story buttons for different types of news. Given that there are imperfections in any of these segmentation heuristics, the MegaSound visual interface enhances news browsing, within these limitations.

The Next Speaker or Story buttons are labeled with the familiar symbols for fast forward “>>”, while the Previous Speaker and Story buttons are prefixed with the rewind symbol “<<”. Note that for purposes of NewsTime, new story markers will always have a corresponding new speaker marker, so that pressing the “next speaker” button would also jump ahead to the next story if read by the same speaker.

4.5 Segmentation of Television News

The closed-captioning of television news makes segmentation a far different task than for radio news. This section examines the segmentation techniques that captions allow.

4.5.1 Closed Caption Synchronization

The closed captions provided by ABC World News Tonight (and also CNN) follow a consistent format which prefixes any new speaker with the string “>>”. This is to assist hearing impaired viewers because the video accompanying a news story often does not show who is talking, or does not show them concurrently with their speech. Likewise, whenever there is a new story the string “>>>” is transmitted. Upon startup, the **getcaption** program notes the time, and later whenever two ‘>’ characters arrive sequentially, the time since startup is written out enclosed by ‘<’ characters, which do not normally appear in closed-captioning. Since the **getcaption** program is run immediately after the program which records the audio,

rec30min, the timestamps for new speakers and new stories should be properly aligned with the audio file.

A similar method of synchronizing video news stories with closed captioned text was used in Judith Donath's "Electronic Newstand" program [Don86], except that the resolution of the timestamps was one for every individual character that came in. This resolution is unnecessary for NewsTime. A user will only need to browse at the story or speaker level and not at the word or letter level; and even if they did need this resolution for some reason, it would not be possible to provide it accurately. The reason is that the closed caption text itself is far from being perfectly synchronized to the audio. The words come in roughly at the rate they are spoken but there are usually significant and unpredictable delays. On a typical broadcast I measured these delays to be anywhere from less than a second to as much as 4 seconds. In fact, because the person entering the captions has a transcript of the news already prepared, the text can even precede the actual speech, though this almost never happens.

How then were text and audio perfectly synchronized to indicate new stories and speakers? This was done in two steps by a program called **syncs**, which takes the original caption timestamps and adjusts them to better reflect the actual timestamps that would synchronize the captions to the audio. This is based on the idea that significant pauses will often occur between speakers, and nearly all of the time between stories. This was verified by analyzing the pause data for a sample World News Tonight broadcast.

Pause Length	Between Stories	Between Speakers	Same Speaker
Average	.753	.580	.503
Min	.319	.309	.309
Max	1.158	1.497	1.118

Table 4.2: Pause data for World News Tonight¹¹
All times are given in seconds.

One would expect the numbers to decrease from left to right, as pauses between stories should be longer than pauses between speakers, or pauses taken by a single speaker. In fact this is correct for the average pause time, although the variation is wide across the three types of pause, and in fact the maximum intra-speaker pause exceeds that for inter-speaker pause. Since the difference in the average time between inter and intra-speaker pauses is negligible (580 vs. 503 ms), it was chosen to instead select for the between story pauses only. This was done by setting the minimum silence duration window to 800 ms, which exceeds the average such pause length of 753 ms.

Given the speech and silence segmentation database for the audio broadcast, the *syncs* program made a first pass through the caption database, finding the nearest speech segment to each new speaker timestamp, which should correspond to the new story boundaries. The delay from which the person begins speaking to the time the caption initially appears is referred to as a “synchronization delay.”

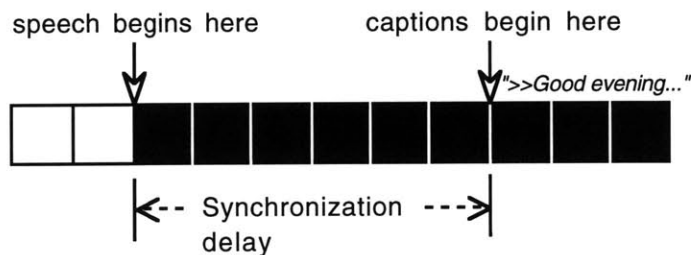


Figure 4.3: Definition of “synchronization delay”

¹¹ These results were derived by Charla Lambert of the Speech Group.

After the first pass, **syncs** sorted all these delays into a list, choosing the median as a default value to use for the second pass. During the second pass, the program tried to synchronize the new speaker timestamps, offsetting them first by this median delay. If the nearest speech block began at an absolute distance greater than twice this median value from the initial timestamp, then the median itself was used as a default offset value to adjust this timestamp. The resulting adjusted caption timestamps are then written back out to the annotation database.

4.5.2 Closed Caption Results

Total New Story	Hit	Miss	Hit Yield	Sig. Miss	Insig. Miss
131	118	13	90%	3%	7%
Total New Speaker	Hit	Miss	Hit Yield	Sig. Miss	Insig. Miss
489	374	115	76%	6%	18%

Table 4.3: Closed caption synchronization via **syncs**

Table 4.3 summarizes the results of running the **syncs** program on a week of ABC World News Tonight. The hit yield is the percentage of all segments perfectly synchronized, while the miss yield is broken down by significant misses (off by more than a single word), and insignificant misses (off by a single word or syllable). As expected, new story boundaries were correctly synchronized more often than new speaker boundaries. This is because pauses between stories are longer, and therefore more reliable to detect than briefer pauses between speakers in the same story. One major source of synchronization mistakes was very brief speech segments, on the order of the median synchronization delay itself. In one such case, the reporter asked a yes/no question, got an answer after a brief pause, then immediately asked another question, both the answer and the second question were originally mapped to the same speech block. To avoid any overlap the **syncs** program would adjust these two segments to be sequential,

but this did not guarantee the brief segment would be accurately synchronized. In reality though this is not a big problem because the very brief segments will likely not contain keywords of interest, so the user will not have done a search on them in the first place. Besides which, the speech segment in question will either be in the preceding or subsequent segment, which can be retrieved at the press of a button during playback.

The real problem was not in matching the text timestamps with the audio, but rather in relying on the new story indicators (“>>>”) that originally came with the captions. These indicators are not exactly “new story” indicators per se, but are better described as “change of focus” indicators. For example, the first “>>>” will tag the introduction to a new story by Peter Jennings, the anchorman; however another “>>>” will prefix the story itself which is given by a field reporter. On one broadcast there were 8 such segues that were marked as new stories, out of 40 overall markers. The upside of this liberal use of new story markers is that no actual new stories were left unmarked. That is, there are some “false alarms” but no “misses.”

4.5.3 Commercials

Another advantage that closed-captioning provides is the ability to reliably filter out commercials. Closed-captions on the news always use smooth scrolling with centered text, while television commercials have no scrolling, and can pop up in any location. The change between these two display modes is detectable by a control string. The raw caption data, having been passed through the standard filtering pipeline to make it more readable, is then handed off to a lex based parser called **cap2db**. This program then creates the time-stamped transcript readable by the SoundViewer and NewsTime. Whenever the control string indicated a switch from news to commercial captions, any ensuing text and time stamps were ignored. This

means commercials' text will not appear in NewsTime, but that the audio will still be there; however, one can instantly skip past it by clicking the next story or next speaker button.

4.5.4 Keyword Search

The "Search" menu in NewsTime is meant only for programs which have closed-captioned transcripts, i.e. ABC World News Tonight. NewsTime can do a keyword search through the closed-captioned text, making visible any annotation flags whose text contains the keyword, and hiding all others. Clicking on these flags plays the accompanying sound just as before, and an accelerator for jumping to the next annotation is also available. Shown in Figure 4.4 is a search for the word "president".

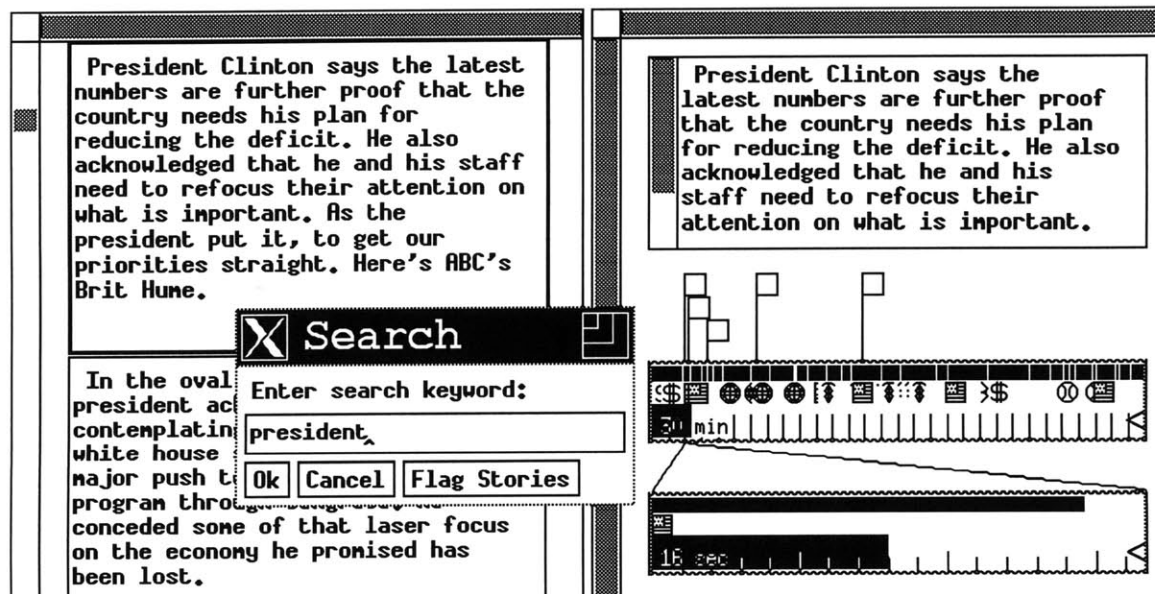


Figure 4.4: Keyword searching

Since the keyword search function hides the original flags which denoted story boundaries, the "Flag Stories" button recreates the initial state of all flags.

4.5.5 Categorizing the News

Obviously some words will provide more information content than others. "Wall Street" or "Dow Jones" will invariably give you the daily stock report, while "president" can give you any number of things. To assist NewsTime in identifying particular types of news, an automatic keyword search program called **karma**, was written, so named because it infers the overall feeling or atmosphere of a story. **karma** takes as input the closed caption transcript, and for each story determines how many key phrases in each of several categories are contained in the story. The news categories included are: international, national, economic/business, entertainment, health/medical, sports, and weather. The category with the most matches "wins" that particular story. In case of a tie, the later category is chosen because they tend to be more specific than the earlier ones, with fewer matching key phrases. In case there are absolutely no matches, the story is left as indeterminate.

Each category could include up to 256 key phrases, read in one per line from a standard text file. New phrases and categories can be easily added by modifying this file, instead of changing the **karma** program. This makes it easy to add world "hot spots" temporarily to the list of international locations, and remove them later. Phrases were chosen for each category by informal analysis of the transcripts from a week of news. The phrases are mostly place names, person names, and terms used in a particular field, e.g. "disease" for medical news, "interest rate" for economic news, etc... The phrases do not even have to be complete words, as long as they unambiguously indicated the category: for instance "econom" is a key phrase because it matches "economics", "economist", or "economy." A complete list of the phrases for each category is listed in the appendix.

Depending on the category computed for each story, NewsTime displays a content icon which is recognizably associated with the category.

- 🌐 International
- 🇺🇸 National
- 💰 Economic/Business
- 🎭 Entertainment
- 🏥 Health/Medical
- ⚽ Sports
- ☀️ Weather

Figure 4.5 News category icons

Because the icons are of a fixed width, when the story takes up less screen space than the icon that portrays it, icons from brief stories will overlap. Here we see the very brief Wall Street report sandwiched between two other stories:

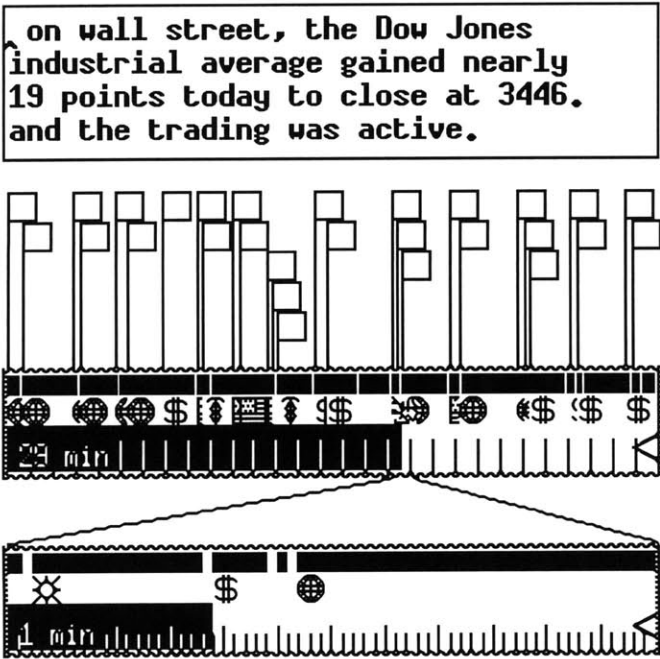


Figure 4.6: Zooming into story clusters

When this occurs the user can zoom the MegaSound widget into the cluster of stories to reveal their individual nature. Another approach to solving the icon overlap problem has been implemented in Marc Davis' Media Streams system for video annotation [Dav93]. Instead of a single content icon layer, a hierarchy of layers for specific icon types exists. Similarly, Thomas Smith's Anthropologist's Video Notebook could graph multiple layers, or "strata", associated with different keywords [Smi89]. As in Media Streams, multiple strata could overlap along the x (time) axis, because each had its own y coordinate. Such stratification may be a good future addition to the SoundViewer widget.

The *karma* program was very successful at distinguishing between the news categories used for this thesis. There were only one or two very short reports per program that were left uncategorized, and no falsely categorized stories. Current research at the lab is investigating how to automatically compute story understanding through various text similarity metrics [Has93]. It is hoped that an intelligent agent can be created to find stories which are attuned to the user's interests. Were such an agent available, NewsTime could make use of it to flag the stories of potential interest; but the main idea behind NewsTime is not to have the computer figure out what news stories you're interested in, but rather to let you make that decision.

4.6 Segmentation of Radio News

Closed captions are a subcarrier that is embedded in the news signal; in radio news however there is no such auxiliary signal. There is a standard known as Radio Data System, or RDS, which has been used in Britain and other European countries since the late 80's [Fel92]. It transmits digital information to complement to the audio broadcast, such as station name and program type. Someday RDS might offer a text transcript as well. Lacking

such a subcarrier today, we are dependent on the program signal alone to determine story boundaries. As there is no general technique for speaker independent continuous speech recognition, a different method was required.

A related project in the video domain shed some light on how this might be achieved. Elliott's "Video Streamer" knew nothing at a semantic level about what was being viewed, but could derive transition points based on signal characteristics. His program finds where likely scene changes are by checking how much the chrominance distribution has changed from frame to frame. In fact, Elliott's software was originally considered as a potential source of timestamps for NewsTime speaker and story boundaries. When it was found that the closed captions were sufficient for this purpose, this idea was abandoned. In the future though, video signal analysis might provide some benefit for partitioning non-captioned programs.

Determining when one speaker stops and another begins without solving the much more difficult problem of voice or speech recognition is appealing. Rather than identifying the speaker or what is being said, NewsTime answers the question: "Was there a long enough pause to denote a possible speaker change?" In the NPR hourly news updates, which are each 5 minutes long, different silence windows were tested against a day of news. Results for finding the speaker and story boundaries (which are treated the same in this case) as a function of the minimum silence window, W , are summarized for a day of NPR (11 separate updates) in Table 4.4.

<u>W (ms.)</u>	<u>Hit</u>	<u>Miss</u>	<u>False Alarm</u>
400	115	37	116
600	53	99	24
800	25	127	2
1000	20	132	0

Table 4.4: NPR new speaker and story yield vs. W

In order to increase the number of hits, the silence window had to be made more liberal (i.e. shorter), but this always leads to a corresponding increase in false alarms. Further, the law of diminishing returns applies, since the rate of additional false alarms quickly exceeds the rate of additional hits. In going from 600 to 400 ms, the number of hits doubles, but the number of false alarms explodes by a factor of five. The reason there will always be some misses is that news reporters are not completely consistent in pausing between stories, and getting every last story boundary would require false alarms within every story too. Of those 37 misses when W is 400 ms, 15 are rapid transitions within a story between one speaker and another. Personally, I found a setting of 400 ms for W to be bothersome, because of the equal number of hits and false alarms. Rather than hit every single speaker transition, I was more interested in hitting story transitions. Consequently I set the W parameter for NPR to 600 ms. Of the 99 misses, 65 were either speaker transitions, or station identifications. This left only 34 actual story boundaries missed, or 3.1 per recording.¹² As there were an average of 8 genuine new stories per recording, this yielded a hit ratio of around 61%, with only 2.1 false alarms per recording.

There is no one right answer to what the silence window should be. Longer radio programs such as Marketplace or All Things Considered do not follow the same limited formula of “while not done, read next story, pause, repeat...” Pauses may also depend on the speaking style of the person being interviewed, or reading a commentary. Figure 4.7 shows an hour long radio program, All Things Considered, displaying silences greater than 800 ms.

¹² One of these 3.1 missed stories was always the stock market update, which due to its brevity was never preceded by a long pause.

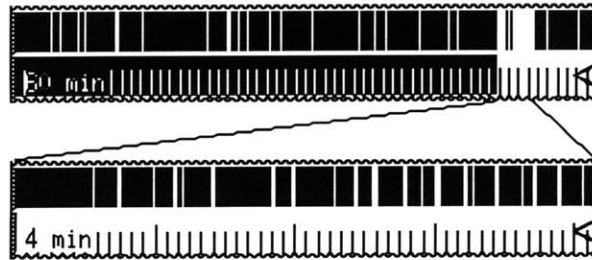


Figure 4.7: Frequent pauses reveal unique speaker

Each pixel represents roughly 10 seconds of audio. If a silence of the minimum duration exists within that pixel's 10 seconds, it is left unfilled. At the indicator's position there is a visible difference in speaking style, as this person clearly pauses much more often than anyone else in the show. In fact, this example is a commentator telling a story, while the rest of the broadcast is more in line with the hard news format of the hourly NPR updates. The isolated pauses at the beginning of the program can help to locate stories but not those near the end. But because this portion of the program is visibly different, the savvy user can quickly sweep the indicator through this region and notice when the voice changes to find the next story.

4.6.1 Music Detection for Next/Previous Story

The visual display of speech and silence can indicate extreme variation in speaking style, but for many programs there is a more prominent signal processing technique for finding story boundaries. Most programs have a theme song, but many also use musical segues of five to fifteen seconds to separate major stories. Michael Hawley has developed an algorithm which can detect the presence of music by finding sustained peaks in the magnitude spectrum [Haw93]. Viewed in a frequency vs. time spectrogram, these sustained peaks appear as sharp lines of frequency, while regular speech appears much more irregular. The algorithm samples the sound every 0.25 seconds, and measures the overall "striation" of each sample on an arbitrary

scale. Values on the scale mostly range from around 4 to 8, with a higher value denoting more striation. The values actually represent the average length of the sustained peaks for that sample, measured in “frames”, where there are 40 frames per second. A graph of these values plainly reveals the presence of music against the speech that borders it:

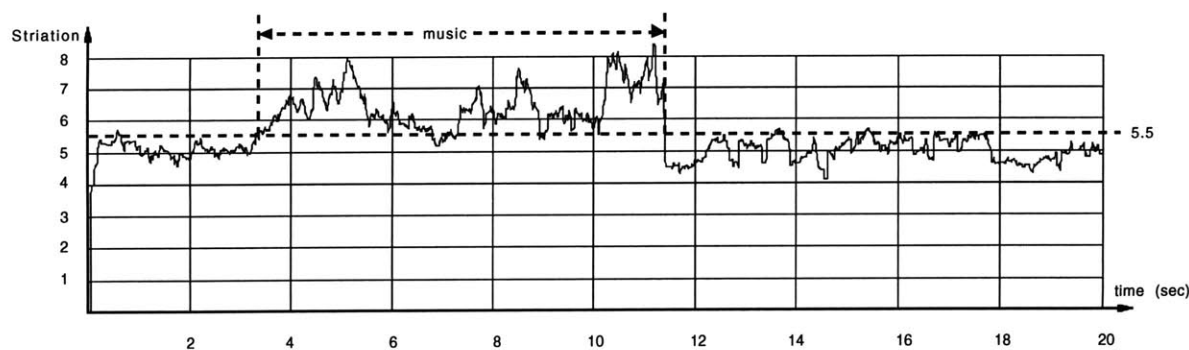


Figure 4.8: Striation of music vs. speech

The first three seconds are a person finishing up one story, while the last nine are the beginning of a new story. The peaks in the middle are a drum and bass combo of about eight seconds. From the above graph, we see that the average speech striation rarely goes above 5.5, while the music rarely dips below 5.5. These same numbers are constant across a wide variety of speakers and music. A prominent false alarm is someone who speaks in a large hall, or stadium which echoes. In this case the striation increases because each frequency component in the echoed speech is stretched out over time, which ends up looking more musical than ordinary speech.

4.6.2 Performance

Hawley's algorithm considers only peaks in the frequency range of 150Hz to 1050Hz, which is suitable because higher frequencies would merely appear as additional harmonics to the lower frequencies, and do not in themselves provide unique information. The algorithm ran on a NeXT station about twice as fast as real time, i.e. an hour long recording took 30

minutes to evaluate. For this reason, a program called **music** which incorporated his algorithm was run immediately after each broadcast. Just as the speech/silence detector had to specify a minimum speech threshold and a minimum silence window, so too did the **music** program have to specify a minimum striation threshold and minimum music window. After trial and error on various broadcasts, a value of 6.0 was chosen as a good striation minimum for music. The fact that most musical segues did not exceed this threshold for their entirety was not an issue because for any melody longer than a few seconds, it was found that this limit was always exceeded at least once. For the music window, a value of 3.0 seconds was chosen, for similar reasons. These were conservative estimates to minimize likelihood of false alarm. They do not guarantee to catch all possible musical signatures, but they consistently found those in the sample programs tested, and only averaged between two and three false alarms per hour with these settings.

4.6.3 Integrating Program Structure with Music

Though not all stories are prefixed this way, recent logs of All Things Considered showed an average of seven musical transitions, with between four and five stories not preceded this way. The unannounced stories were generally not the hard news ones, but were interviews, commentary or reviews. Interviews were often considered part of a larger story, which had already been initially tagged with music.

Just as radio programs occur at regular clock intervals throughout the day, so too do regular program segments occur at roughly predictable offsets. On Marketplace there is consistent music, but it is used to denote entire portions of the program rather than every individual story. As on All Things Considered, music plays while an introductory summary is given at the top of the show. On Marketplace though it also precedes the mid show summary,

the foreign business news, the top story, the day's corporate news, the stock report, news from the daily business press, and various other special reports. Here we see a picture of a Marketplace broadcast with the music segments flagged as new story markers.

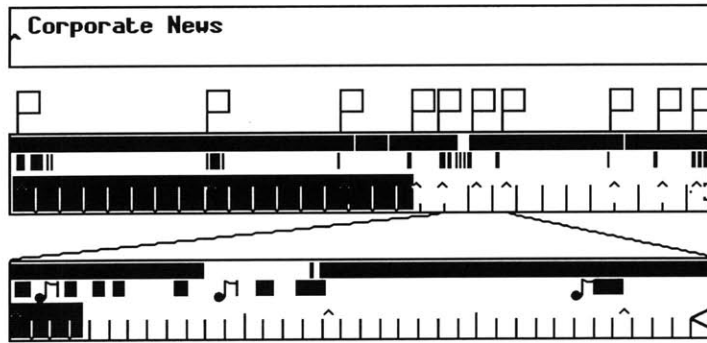


Figure 4.9: Music confirms story predictions

Within these reports the next speaker button can find new stories with the same accuracy as in the NPR hourly news updates. Ironically, this simpler speech and silence algorithm is more precise at finding individual stories than the more sophisticated music detector. One factor is that the announcer often reads interesting news while the “band plays on.” Because the human voice only adds frequency components to the background spectrum, it will not interfere with music detection, but it does mean that finding the actual news is not a simple matter of skipping past the music.

To assist with this problem, output from the **music** detector goes to a second process, **music2db**, which given the name of the program being analyzed, looks up a template for that program. These templates are entered by the programmer in advance to reflect any background knowledge about the predictable structure of the broadcast. Each line of the template lists five pieces of information: the minute and second offset at which to look for a musical segue, the expected length of the segue, whether or not the announcer speaks over the music, and a brief description of what the segment

ought to be. The Marketplace template which was determined by averaging a week's worth of shows is shown here:

Min	Sec	Dur	Keep Music?	Description
0	0	65	1	Theme/Summary/Top Story
7	38	26	1	Still To Come.../Foreign Desk
12	55	10	0	Feature Story 1
15	39	30	0	Corporate News
17	27	46	1	Stock Report
18	38	30	0	Daily Business Press
19	36	10	0	Feature Story 2
23	36	10	0	Feature Story 3
26	37	91	0	Final Notes on News

Table 4.5: Marketplace template

The **music2db** program first makes a pass through all musical segments, and groups them into clusters. This is because the conservative parameters fed to the **music** program meant that rather than identifying a single piece of music as a solid continuous block, it was more likely a fragmented collection of brief but closely spaced musical segments. Music fragments which were less than 25 seconds apart were assumed to be part of one large musical cluster. For each line in the template file, the start time was matched to the start time of the nearest cluster, and then depending on whether or not the “keep music” flag was set, an annotation containing the text description was added to the database either at the start or end of the music cluster.¹³ After the template was fully matched, any leftover musical clusters which were left unmatched would be tagged with the default annotation, containing “???”. This feature helped the user find any unexpected musical segues in the program, and would also compensate for any mislabeled segments. This way, even if a segment was in the wrong place, it would still be flagged, albeit with a meaningless label.

¹³ Actually if the music was to be skipped, the note would be placed 1.25 seconds earlier than the end time to be on the safe side.

If the new story marker is placed too early or too late with respect to musical interlude, the user interface once again compensates. One can easily jump back to the beginning of the current music block, in case any words were missed, or to skip to the end to save time. This capability to skip between music blocks is part of the improved SoundViewer.

4.6.4 Results

Marketplace is on Monday through Friday, and there are nine segments to find per show, so in the week that was tested there should have been 45 segments to correctly discover. In fact 35 of these were accurately labeled by the **mus2db** program, or roughly 80%. Of the ten that were missed, five were still found, but just labeled incorrectly with “???”, two were mislabeled as another segment of the show, one was omitted from the show in the first place, and only two were not detected at all by the **music** program. There were an additional eight false alarms or 1.6 per day; one of which actually was music, but was part of the story nevertheless.

4.6.5 Future Possibilities

Music detection is only one signal processing technique which can make use of *a priori* knowledge analysis of a newscast. In All Things Considered, there are usually two anchors, male and female, who alternate reading the headlines. Unlike what is found in spontaneous conversation, the two never interrupt each other, and each person always speaks for at least several seconds. The anchor will often read a headline, then yield to field reporters who have phoned in their stories. The audio quality of these phoned in reports is noticeably lower than that for the live anchor. Natalio Pincever's Home Moviemaker's assistant [Pin91] successfully demonstrated how statistical analysis, Fast Fourier Transforms, and Linear Predictive

Coding could be used to detect significant changes in the audio stream. His purpose was to parse video scene transitions through audio analysis alone, such as changes in root mean squared amplitude of background noise. Pincever found that most shot changes could indeed be discovered through audio analysis. With future improvements in pitch tracking software, it should be possible to spot such patterns in the audio and use them as “signposts” for further scanning in NewsTime.

Chapter 5

Implementation

5.1 Widgets' Technical Specification

We have now explained widgets like MegaSound at their functional level, i.e. as “object[s] providing a user-interface abstraction” [Pet89]. The term “widget” also has a more precise technical definition, as one of the underlying data abstractions of the X Window system. Widgets themselves are in fact realized as individual X Windows, and the Xt Intrinsics Library (colloquially known as “Xt”) is provided as an application programming interface (API) to the widget sets. Different widget sets are available, such as Motif or OpenLook, which all are based on Xt and its abstractions. The work described herein is all based on the Athena widget set, which comes with the standard X Windows release.

Within any widget set is a number of widget classes, the Scrollbar¹⁴ class for example. Each class contains a number of data fields and methods, and new widgets are written by subclassing an existing class. Scrollbar is a subclass of the Simple class, which itself is subclassed off the root class known as Core. A number of elemental classes such as Simple and Core are provided with Xt, along with some methods essential to all widgets: Initialize, SetValue and GetValue, Redisplay, etc... A widget inherits all the data fields and methods of its superclass, and then adds its own. In addition, the subclass has the option to overwrite any of the superclass' default values or methods.

¹⁴ By convention, all widget class names are Capitalized.

5.2 The OmniViewer Widget

One of the underlying ideas of Xt is that widget semantics are kept independent of widget geometry. That is, widgets are supposed to have no control over where they are placed on screen, relative to other widgets. This stems from the idea that every widget stands alone, unaware of the existence of sibling widgets, or of the overall application in which it is used. This is in keeping with the object oriented approach of Xt, which lets the same widget be used across any number of applications without side-effects.

In order to handle widget layouts Xt provides another widget class called Composite. Composite widgets are designed to manage the geometry of a number of child widgets. Two composite widgets come with the Athena widget set: Form and Box, both of which let the programmer specify the position of each child widget relative to its siblings. Neither one lets you specify the position of a child in absolute coordinates, and perhaps surprisingly, no such “bulletin board” style widget exists in the Athena widget set. To alleviate this problem a widget class called “Bboard” was created.¹⁵ The geometry manager in Bboard simply grants all its childrens' layout requests, and resizes the Bboard to exactly fit all children.

The Bboard is very useful but also very generic. Since it knows nothing about the nature of its children, it needed to be subclassed to specifically manage audio and text widgets. The subclass created for this purpose was the ChatMan widget, which is short for chat manager. ChatMan manages a database of text and audio items, but does not actually create any widgets to represent those items. This is done by a subclass of ChatMan called OmniViewer.

¹⁵ The original Bboard widget was created by Russ Sasnett and Mark Ackerman of MIT/Project Athena.

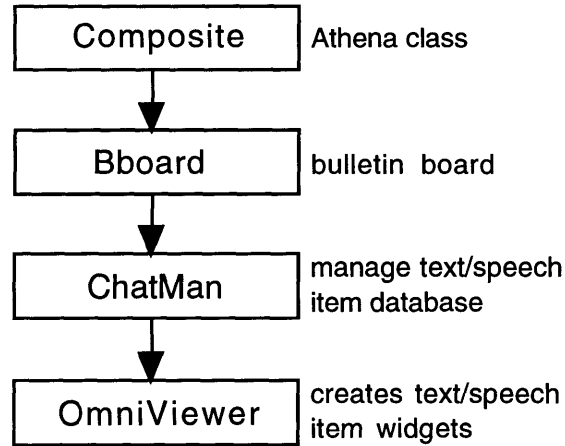


Figure 5.1: The OmniViewer class hierarchy

ChatMan stores all relevant information for each item in its database: such as type of item (text or audio), sequence number, x and y coordinates. Sound items additionally have their Unix file name, start time, stop time and duration managed, while text items simply need their contents stored. External procedures are provided by the ChatMan widget for adding new items, deleting items, moving items, changing an item's contents, and reading or writing the entire item database.

ChatMan was designed as a more general superclass of the ChatViewer widget, which was written by Debby Hindus of the Speech Group [Hin92]. The ChatViewer not only managed the items' data, it also created the widget realizations of those items. The ChatViewer was created to view telephone conversations as a series of SoundViewer widgets, according to one of only three layout styles: a horizontal row, a vertical row, or in miniature. ChatMan was spun off separately because there could be any number of applications for displaying text and speech items, any one of which might have its own unique layout technique. For extreme example, the Speech Group's "Scrap" application creates a scrapbook of sound and text items for reference, pasting them onto the screen at any possible location.

By separating out the code which managed the item database, the widget realization of those items could be left up to the subclass, e.g. what type of widget to use for sound items. The choice of how to lay items out can also be left up to the subclass. The OmniViewer is one such subclass, so named because as the Scrap application demonstrates, it can handle all possible layouts. Like the original ChatViewer, the OmniViewer uses Athena Text widgets for text items, but instead of SoundViewers, uses MegaSound widgets. The OmniViewer has parallel procedures to ChatMan's for adding, deleting, moving and modifying its items. These procedures affect the children widgets on screen, but additionally call the equivalent ChatMan procedures to update the internal item database. The appendix lists all resources, translations, callbacks, and external procedures for the Bboard, ChatMan and OmniViewer widget classes.¹⁶

5.3 Chatfiles

Much of the underlying code for the ChatMan was modified from the ChatViewer. The ChatViewer employed two different file formats to read and write its item databases, both of which were preserved for ChatMan. The first format is a *chatfile*, which is simply a flat text file, with one item per line. Each line is either the text item itself, or a reference to a sound file name, with segment start and stop times optional. For instance:

This is a text item.
<sound file: /sound/news/today/npr>

The chatfile format merely lists the contents of each item, but cannot reflect supplementary item information, such as item coordinates. If this type of information is desired, another file format called a *dbfile* may also be used. This format is supported by a Speech Group database, **ndb**, written to

¹⁶ For definitions of these Xt terms, see appendix.

maintain key-value pairs. The **ndb** database is also the native language used by both the ChatViewer and ChatMan widgets internally. The exact format of dbfiles are discussed in the appendix.

NewsTime is written by putting two OmniViewers side by side. The one on the left holds the show's text, while the one on the right holds the show's audio. When there is no transcript to put in the text window, as is true with all radio programs, the name of the program is placed there instead. When someone selects a new program from the "Program" menu of NewsTime, the OmniViewers will purge all existing text and audio items, and then reload the appropriate chatfiles for either the text or audio of the show. These chatfiles are found in a network accessible directory for audio news, and are simply named chat1.text - chat5.text, and chat1.sound - chat5.sound.

5.4 SoundViewer Hierarchy

The SoundViewer widget class is based on another custom widget: the Ind (or Indicator) which is subclassed off the Xt class, Simple. The SoundViewer deals with all the details of playing an audio file, while the Indicator (which knows nothing about audio) deals with sizing the widget, drawing in hash marks, and implementing the black indicator bar which moves across the screen. The indicator bar works by noting when it is activated, then repeatedly calling the IndicatorTimerProc method. This checks the clock again to see how much time has elapsed since the bar was switched on, and based on this number the bar updates its position. The SoundViewer needed to overwrite the IndicatorTimerProc method with its own SoundViewerTimerProc. Rather than checking the clock at each iteration, this method queried the sound server [Aro92b] about the current play position in the sound file, and updated the bar based on this instead. This

made it possible to change the playback speed of sounds repeatedly, and have the indicator bar slow down and speed up accordingly.

Some important resources that determine the SoundViewer's appearance are the showSegs, showContent, showNotes, showDuration, and showLabel flags. These respectively determine whether or not speech and silence, content icons, annotation markers, segment duration, and the optional text label are displayed. Finally the useAuxFile resource determines whether the auxiliary file should be loaded during the Initialize method, or written out if there are any changes, such as new bookmarks or new segmentation parameters.

5.5 MegaSound Hierarchy

Since the MegaSound has already had an entire chapter devoted to it, we will now only discuss its class hierarchy. MegaSound, like the ChatMan widget, is also subclassed off the Bboard widget. In its Initialize method, all children of the MegaSound widget are created: the root SoundViewer and any zoomed SoundViewer region, the optional annotation flags, (which are Athena Command widgets) and optional main note entry window (an Athena Text widget). The flagNotes resource controls whether the annotation flags are drawn, while the autoMode resource determines if a zoom region is automatically created. The duration of the zoom region is specified by the autoWindow resource. The appendix contains complete MegaSound widget documentation.

5.6 Auxfiles

To support the speech and silence segmentation, semantic content, and annotation databases associated with a sound file, the auxfile format (short for “auxiliary file”) was created. These three databases are just concatenated together to create a file with the same name as the sound file, with the suffix “.aux” added, and prefixed by “.” to make the file invisible to the Unix shell. The databases themselves are managed by the same **ndb** library that the ChatMan widget uses to maintain key-value pairs. A description of what those pairs are for each of the three databases follows.

5.6.1 Speech and Silence database

The first record in the segmentation database contains the database type (SegDb) for verification, as well as the detection parameters used to segment the file.

```
Start of db
{Record
{{Type} {SegDb\00}
{SilenceLength} {800\00}
{SilenceLevel} {0\00}
{FirstSeg} {1\00}
}
```

As in all dbfiles the first item enclosed in {}'s is the key, while the second is the value. Note that all values end with the string “\00” by convention. The segmentation information in this database reflects silences at least 800 ms long by the SilenceLength field. The maximum SilenceLevel of 0 indicates the level was determined by the formula given in chapter 2, based on the first 35 seconds of the audio data.¹⁷ A FirstSeg of 1 indicates the sound begins with speech, while a 0 would have meant silence.

¹⁷ $T = (0.08 \text{ MAX} + 2.92 * \text{MIN})$

Following the header record, all subsequent records contain only one field:

```
Record{
  {{SoundLen} {22720\00}
}
```

Subsequent records indicate the duration of the next interval, which alternately is speech or silence, and depends on the binary value of the FirstSeg field in the header record.

5.6.2 Content database

The content database also begins with an identifying header record:

```
Start of db
{Record
  {{Type} {ContentDb\00}
  {SoundStart} {-1\00}
}
```

Because the items in this database get sorted in increasing order by the SoundStart field, the -1 in the header record is to insure it remains the first record.

Following the header is an arbitrary number of content records, each of which contains four fields:

```
Record{
  {{Contents} {1\00}
  {SoundStart} {6154\00}
  {SoundLen} {7770\00}
  {SoundStop} {13924\00}
}
```

The SoundStart, SoundLen, and SoundStop fields reflect the location and duration in the sound file of a segment with a particular type of content. The Contents field is a number used by the SoundViewer to indicate which

content icon will be drawn at the specified start time. A Contents field of 0 also indicates to the SoundViewer to draw in a black bar spanning the duration of the content segment, over which the icon is placed at the beginning. With any non-zero value, only the content icon is drawn, and not the bar. This is how regions of music can be drawn in as a base content layer, while other types of semantic content can overlay this base with additional icons.

5.6.3 Annotation database

The annotation dbfile reflects both bookmarks or annotations the user has added through the SoundViewer or MegaSound widget, as well as the transcript of the audio generated by closed captioning. For this reason the first record requires more explanation than for the other dbfiles.

```
Start of db
{Record
{{SyncDelta} {-1155\00}
{NewStory} {1\00}
{NewSpeaker} {1\00}
{SeqNum} {1\00}
{Visible} {1\00}
{Type} {TextItem\00}
{SoundStart} {6154\00}
{SoundLen} {7770\00}
{SoundStop} {13924\00}
{Contents} {Good evening. We begin tonight with a story from
Washington.\00}
}
```

In annotation dbfiles, the first record is no different than any other record with the exception of the optional field, SyncDelta. Recall the discussion earlier about the synchronization delay that was encountered for closed-caption text. The **cap2db** program which generates the annotation dbfile from the filtered caption data does not in itself compute the median synchronization delay. This is done separately by the **syncs** program, which

when fed an auxfile adjusts the SoundStart, SoundStop and SoundLen fields appropriately. Initially, if the **syncs** program was run more than once on the same annotation dbfile it would compute a different synchronization delay the second time, thereby destroying the properly adjusted timestamps. To fix this problem, the synchronization delay is inserted into the first annotation record, the first time the **syncs** program is run. If it finds it there the second time, nothing more is done.

The NewStory and NewSpeaker fields are inserted into a record by the **cap2db** program when the appropriate prefixes are found in the captioning. An annotation or bookmark added by the user will have neither of these fields set. The Visible field is another optional field, that is only set by default for NewStory captions and not NewSpeaker captions. As there were usually around 100 NewSpeaker captions¹⁸, flagging them all would clutter the MegaSound widget. Yet the visible state of individual annotations can still be changed by the user, as in NewsTime's keyword Search feature described in chapter 3.

The SeqNum and Type fields are not necessary for either the SoundViewer or MegaSound widget, and are technically not mandated by the annotation dbfile format. They are only inserted by the **cap2db** program to generate a dbfile that can also be used by the OmniViewer widget. This allows NewsTime to take the one dbfile, and display it as the newscast transcript in its left hand window. The same dbfile is incorporated into the auxfile which MegaSound uses for the same newscast.

¹⁸ Actually 91.4 on average for the week of news examined.

Chapter 6

Conclusion

6.1 Goals Accomplished

The major focus of this work has been the enhancement of the visual interface to large quantities of speech audio, within the context of news broadcasts. The reusable MegaSound widget made both the NewsTime programmer's and user's job easier. Logging enough knowledge about different news broadcasts, and devising reasonably effective segmentation techniques was a major problem to tackle. Even though segmentation was not perfect, there were relatively high hit ratios, even using the simple speech and silence detection algorithm. With music detection or closed captioning, story segmentation achieved extremely high accuracy, while keeping the false alarm rate low.

Given that no segmentation data was perfect, I found that the enhanced MegaSound interface did indeed compensate in several ways. In terms of usability, I personally found NewsTime enhanced my listening, and made me enjoy my favorite programs even more. It truly provided for the first time the ability to immediately “flip through” an audio newscast the same way I would through a newspaper. For instance, I liked being able to zoom in on Marketplace's stock reports very quickly at the end of the day. Even though the report was correctly labeled only 80% of the time, the music was still detected, and a click or two of the next story button would find the report, within a second or two. But even with no segmentation information, MegaSound's enhanced scanning with audio feedback, and its zooming feature are a boon in distinguishing particular pieces of sound. This was especially true if the sound characteristics were audibly different even in a split second of feedback, such as music vs. speech, or a change of speaker.

The ability to display content icons within the SoundViewer was both accurate and helpful in distinguishing among the general news categories determined by the **karma** program. A good NewsTime enhancement for the future would be an improved version of **karma** that could go beyond its very general categories to let one target stories which are more likely to be of particular interest. User modeling could help to detect which stories I actually listen to in a broadcast, and then future stories would be highlighted that are more similar to those I personally select. Another more trivial enhancement would be to expand the "Search by keyword" feature to allow each user to configure a set of keyword libraries for themselves, and then flag all segments that contain any of the keywords, instead of only one at a time.

In terms of future segmentation techniques, pitch tracking and other signal processing algorithms could be very beneficial in finding new speaker and new story boundaries. I found that while listening for new stories by skipping over silences, I could almost instantly distinguish between a hit and a false alarm. The announcer would have a distinctively higher pitch, and speak in a louder voice when starting a new story, than when merely pausing within a story. Although speech recognition is not robust enough to generate a transcript from the audio, keyword spotting algorithms could be used to target predictable, repeatable theme music, or tag lines that indicate particular program segments. On the local 11 o'clock Eyewitness News television broadcast, the weather and sports report always begin with the same prerecorded introduction. Finally, although it was not necessary for NewsTime, a "talking heads" television news show, which contains interviews or debates, but is not closed-captioned, may be broken up into new speaker segments by using video continuity detection, as in Elliott's Video Streamer.

6.2 Predicting the Future

I believe that someday there will be the capability to automatically record all news programs coming in over the television or radio on a particular day, and that the ability to browse through all this time based media will become as commonplace as browsing through a newspaper. NewsTime has taken the first steps by showing what is currently possible given the hardware and software of today. Because no news provider explicitly broadcasts signals to assist NewsTime, the fact that the program is fairly successful in the first place is rather impressive. I predict that in the future a standard for broadcasting high level content information along a subcarrier will indeed be the norm. The question of whether this information will be used wisely to educate and enlighten people is ultimately up to the consumer.

Appendix A

Widget Terminology Resources, Callbacks, External Procedures, and Translations

The Xt widget abstraction supports the hiding of widget details from the application programmer. That is, data fields within the widget structure are not to be directly accessible unless made explicitly so through the *resource* mechanism. In the SoundViewer widget for instance, the speed of playback is a setting that the user or programmer would want to modify, and so it is coded as a resource. Other data fields that the widget maintains for its own purposes are kept private. Resources are written with a default value specified, but can be set and received at run-time by a call to the respective Xt functions `XtSetValues()` and `XtGetValues()`. Default resource values may also be overwritten in a resource file that is written specifically for each application.

A translation table is a text resource which maps user behaviors to widget actions. For instance, a user who clicks on the SoundViewer triggers both the `ToggleSound()` and `ToggleIndicator()` actions in the widget. This particular translation is written into a resource specification as follows:

```
<Btn1Down>: ToggleSound() ToggleIndicator()
```

A special type of resource known as a *callback* list can also be specified. A callback list is a list of procedures which the programmer can append to. Procedures on the callback list are triggered when the widget reaches a certain condition. For instance, the SoundViewer has a resource named `finishedCallback`, whose list is triggered whenever its indicator bar stops moving.

An external procedure is provided by the widget for the programmer to request certain behaviors which are either not accessible through resources, or would too cumbersome to accomplish through resources. One example from the SoundViewer is XawStartSound() which both starts the audio playback, as well as triggering the indicator bar. External procedures are like the widget actions that can be specified in a translation table, except that they are called directly from the application's C code, rather than only being called via Xt's translation table mechanism.

Appendix B

Widget Documentation

This section lists the developer's documentation for all custom widgets associated with NewsTime. The SoundViewer widget is not included since it is not really NewsTime specific, but is so widely used in other applications.

B.1 Bboard Widget

/*****

Bboard widget

Authors: Russ Sasnett
MIT/Project Athena
and GTE Laboratories

Mark Ackerman
Chris Horner
MIT Media Lab

Updated: 5/7/93

*****/

SUMMARY

The Bboard widget ("bulletin board") is an extremely simple composite widget, which simply grants all geometry requests from its children, and sizes itself to fit them all.

RESOURCES

The Bboard widget has no additional resources, in addition to those of Core and Composite (Athena). It is a subclass of the Composite widget. See the Athena widget documentation for additional details on Composite resources.

CALLBACKS

The Bboard widget has no callbacks.

TRANSLATIONS

The Bboard widget has no default translations.

ACTIONS

There are no actions installed for the Bboard widget.

CONVENIENCE ROUTINES

```
extern void
BBCalculateNewSize(w, width, height)
    Widget w;
    Position *width, *height;
```

Returns the size of the bounding box needed to contain all the Bboard's children.

B.2 ChatMan Widget

```
/******
```

```
ChatMan widget
```

```
Author:      Chris Horner  
            MIT Media Lab
```

```
Updated:    5/7/93
```

```
*****/
```

SUMMARY

The ChatMan widget ("chat manager") is for managing databases of text and sound items. It is assumed that this widget will be subclassed to create actual text and sound widgets that the ChatMan manages. See the OmniViewer documentation for one such subclass.

This was spun off from the original ChatViewer widget, written by Debby Hindus and myself. The original ChatViewer managed both the database and the widgets by itself.

Data stored for an item includes: type, sequence number, "marked" status, x and y coordinates, start time, stop time, duration and speaker (if sound), contents (if text) or sound file name. This data is reflected in the Item type:

```
typedef struct Item {  
    int type;  
    char *contents;  
    long sound_start;  
    long sound_len;  
    long sound_stop;  
    char *speaker;  
    Boolean marked;  
    int x,y;  
} Item;
```

FILE FORMATS

Item databases are stored either as db files or chat files. Db files are a format supported by the Speech Group's ndb library which maintains databases of key-value pairs. This is the internal representation the ChatMan uses to keep track of item information. An example db file with one text and one audio item looks like this:

```
Start of db  
{Record  
{Contents} {This is a text item.\00}  
{XCoord} {10\00}  
{YCoord} {0\00}  
{SeqNum} {1\00}  
{Type} {TextItem\00}  
}
```



```
Record
{{Contents} {/sound/horner/things/thing.001\00}
{XCoord} {10\00}
{YCoord} {23\00}
{SoundStart} {0\00}
{SeqNum} {2\00}
{Type} {SoundItem\00}
{SoundLen} {3200\00}
{SoundStop} {3200\00}
}
```

A chat file on the other hand was a format developed for the original ChatViewer. It is a more human-readable text format, with one line per item. The above db file has this equivalent chat file:

```
This is a text item.
<sound file: /sound/horner/things/thing.001>
```

Note that the sequence numbers are implicit, and the coordinate information is not saved in a chat file. Also the sound start and stop are not explicitly required unless the sound item is only a segment of a larger file. In this case the sound item looks like this:

```
<sound file: /sound/horner/things/thing.001::1000:2000>
```

The two numbers are the start and stop times in milliseconds of the desired segment. When not given, the entire file is shown by default.

RESOURCES

The ChatMan widget has the following resources, in addition to those of Core, Composite (Athena), and Bboard widgets. It is a subclass of the Bboard widget. See the Bboard widget documentation for additional details on Bboard resources.

```
chatDir
  resource name      XtNchatDir
  resource class     XtCChatDir
  resource type      String
  default            NULL
```

This is the name of the sub-directory in the user's /sound directory in which ChatMan expects to find its sound files.

```
chatName
  resource name      XtNchatName
  resource class     XtCChatName
  resource type      String
  default            NULL
```

This is the default name of the chat file the ChatMan loads and saves its items to.

```
fileDescriptor
  resource name      XtNfileDescriptor
  resource class     XtCFileDescriptor
  resource type      Int
  default            -1
```

This is the Unix fd for all sound files. It is passed to the sound system.

If this is less than zero, the widget is set to insensitive at Initialize or SetValues time.

editable

resource name	XtNeditable
resource class	XtCEditable
resource type	Boolean
default	True

This controls whether or not items are editable (True) or "markable" (False). In "marked" mode, a subclass can select an individual item without triggering that item's default actions.

modified

resource name	XtNmodified
resource class	XtCModified
resource type	Boolean
default	False

If any modifications are made to the item database, this resource becomes True.

isChatFile

resource name	XtNisChatFile
resource class	XtCIsChatFile
resource type	Boolean
default	False

This reflects what type of data file the chat items are loaded and saved into. A True value means that this is a chat file, while False implies a db file.

CALLBACKS

notifyMarkCallback

resource name	XtNnotifyMarkCallback
resource class	XtCCallback
resource type	XtCallbackList
default	NULL

This callback list is triggered by a subclass to indicate an item has been selected by a user ("marked").

call_data is used for the number of the chosen item.

TRANSLATIONS

The ChatMan itself does not have translations, since it does not manage any Xt widgets that represent its item database.

ACTIONS

There are no actions installed for the ChatMan. Again, this is the job of its subclasses.

CONVENIENCE ROUTINES

```
extern int
XawCMIsChatFile (w)
    Widget w;
```

Returns the value of the isChatFile resource.

```
extern void
XawCMDefaultSoundPath(w, default_sound_path)
    Widget w;
    char *default_sound_path;
```

Returns in the buffer pointed to by default_sound_path the full pathname of the sub-directory in the user's /sound directory, in which ChatMan expects to find its sound files. This is specified by the chatDir resource.

```
extern void
XawCMMarkItemInDB (w, item_id)
    Widget w;
    int item_id;
```

Sets the "marked" status of the given item to True.

```
extern void
XawCMUnmarkItemInDB (w, item_id)
    Widget w;
    int item_id;
```

Sets the "marked" status of the given item to False.

```
extern int
XawCMNumItems(w)
    Widget w;
```

Returns the number of items in the database.

```
extern void
XawCMGetCurrentItem (w, id)
    Widget w;
    ITEM_ID id;
```

Returns the number of the current (last modified) item.

```
extern void
XawCMGetPreviousItem (w, id)
    Widget w;
    ITEM_ID id;
```

Returns the number of the item preceding the current item.

```
extern void
XawCMGetNextItem (w, id)
    Widget w;
    int id;
```

Returns the number of the item after the current item.

```
extern void
XawCMAddItem (w, id, info)
    Widget w;
    int id;
    Item *info;
```

Creates a new item at position id with the given info.

```
extern void
XawCMDeleteItem (w, id)
    Widget w;
    int id;
```

Deletes the item at position id.

```
extern void
XawCMMoveItem (w, from, to)
    Widget w;
    int from,to;
```

Moves the item at position "from" to position "to", moving all other item positions as necessary to preserve the sequence.

```
extern void
XawCMGetItemInfo (w, id, info)
    Widget w;
    int id;
    Item *info;
```

Places into the buffer pointed to by info the item information at position id.

```
extern void
XawCMSetItemInfo (w, id, info)
    Widget w;
    int id;
    Item *info;
```

Sets the information for the item at position id to that pointed to by info.

```
extern void
XawCMSetItemCoords (w, id, x, y)
    Widget w;
    int id;
    int x, y;
```

Sets the x and y coordinates for the item at position id as specified.

```
extern int
XawCMReadFile (w, filename)
    Widget w;
    char *filename;
```

Reads the chat or db file with the given filename,
appending the items to the existing database.

```
extern void
XawCMWriteDB (w, fp)
    Widget w;
    FILE *fp;
```

Writes out the item database to the stream pointed
to by fp as a db file.

```
extern void
XawCMWriteChat (w, fp)
    Widget w;
    FILE *fp;
```

Writes out the item database to the stream pointed
to by fp as a chat file.

B.3 OmniViewer Widget

/*****

OmniViewer widget

Author: Chris Horner
MIT Media Lab

Updated: 4/26/93

*****/

SUMMARY

The OmniViewer widget (so named because of its flexibility across a number of applications) is for displaying text and sound widgets whose database is managed by its superclass, ChatMan. The ChatMan manages the general information about each item while the OmniViewer manages the on-screen realization of those items. The text widgets used are Athena Text widgets while the sound widgets are the custom MegaSnd widget.

FILE FORMATS

The OmniViewer's parent class handles all file interaction, which is either a db or a chat file. See the ChatMan documentation for details.

RESOURCES

The OmniViewer widget has the following resources, in addition to those of Core, Composite (Athena), Bboard and ChatMan widgets. It is a subclass of the ChatMan widget. See the ChatMan widget documentation for additional details on ChatMan resources.

playable

resource name	XtNplayable
resource class	XtCPlayable
resource type	Boolean
default	True

This is equivalent too the editable resource of the ChatMan. A sound widget which is playable keeps its default translations while a non-playable widget is markable. In non-playable mode, a user can click on an individual soundViewer, which calls the notifyMarkCallback list from the ChatMan.

whenToScale

resource name	XtNwhenToScale
resource class	XtCWhenToScale
resource type	Int
default	10000

This determines how much time (in ms) a soundViewer can display before it begins to compress its time scale. The OmniViewer's

soundViewers use mode XawIndModeBest, which means that sounds of less than this duration are truncated while anything longer is scaled. See SoundViewer.doc for more information.

vertDist

resource name	XtNvertDist
resource class	XtCVertDist
resource type	Int
default	10

This determines the default vertical spacing of new items in the OmniViewer.

indentDist

resource name	XtNindentDist
resource class	XtCIndentDist
resource type	Int
default	10

This determines the default spacing of new items from the left margin of the OmniViewer.

playAllMode

resource name	XtNplayAllMode
resource class	XtCPlayAllMode
resource type	Boolean
default	False

If playAllMode is True, then whenever a soundViewer plays through to the end, the next soundViewer is automatically started playing.

CALLBACKS

There are no callbacks in the OmniViewer, although the notifyMarkCallback is called when the OmniViewer's playable resource is set to False, and the user clicks on an item. See the ChatMan documentation for a description of the notifyMarkCallback.

TRANSLATIONS

There are no default translations on the OmniViewer.

ACTIONS

The actions for the OmniViewer widget are:

OVLAYOUT

This should be called from a child text or sound widget. It calls the convenience routine XawOVLAYOUTItems() described below.

OVCleanUp

This should be called from a child text or sound widget.
It calls the convenience routine XawOVCleanUp() described below.

OVShrinkWrap

This should be called from a child text or sound widget.
It calls the convenience routine XawOVShrinkWrap() described below.

CONVENIENCE ROUTINES

```
extern void
XawOVVisualMarkItem (w, id)
    Widget w;
    int id;
```

Visually grays out the border pixmap of the item with number id.

```
extern void
XawOVVisualUnmarkItem (w, id)
    Widget w;
    int id;
```

Visually restores the border pixmap of the item with number id to normal.

```
extern int
XawOVNumItems (w)
    Widget w;
```

Returns XawCMNumItems(), the number of items managed. See ChatMan documentation.

```
extern Widget
XawOVItemToWidget (w, id)
    Widget w;
    int id;
```

Returns the widget id of the item with number id.

```
extern int
XawOVWidgetToItem (w, target)
    Widget w;
    Widget target;
```

Returns the item number of the target if it is a child of the OmniViewer.

```
extern void
XawOVAddItemAtEnd (w, info)
    Widget w;
    Item *info;
```


Adds an item described by info to the end of the OmniViewer's list.
extern void

```
extern void
XawOVAddItem (w, id, info)
    Widget w;
    int id;
    Item *info;
```

Adds an item described by info at position id in the OmniViewer's list. Adjusts other item numbers if necessary.

```
extern void
XawOVDeleteItem (w, id)
    Widget w;
    int id;
```

Deletes the item at position id from the OmniViewer. Adjusts other items' numbers if necessary.

```
extern void
XawOVDeleteAll (w)
    Widget w;
```

Deletes all items from the OmniViewer.

```
extern void
XawOVMoveItem (w, from, to)
    Widget w;
    int from, to;
```

Moves the item at position "from" to position "to". Adjusts other items' sequence numbers as necessary.

```
extern void
XawOVCleanUp (w)
    Widget w;
```

This arranges all items, 1 per line, all in a row against the left margin, with spacing determined by the vertDist and indentDist resources.

```
extern void
XawOVLayoutItems(w)
    Widget w;
```

This arranges all items, 1 per line, all in a row preserving their X-coordinates. Spacing is determined by the vertDist resource.

```
extern void
XawOVShrinkWrap(w)
    Widget w;
```

Sizes the OmniViewer to just fit all its children.

```
extern void
XawOVUpdateTextItems(w)
    Widget w;
```

Updates the internal database to reflect any changes the user has typed into the text widgets.

```
extern void
XawOVUpdateItemCoords(w)
    Widget w;
```

Updates the internal database to reflect any changes the user has made by moving widgets.

```
extern void
XawOVPlayAllItems (w)
    Widget w;
```

Sets the playAll resource of the OmniViewer to True, and starts playing the first sound.

```
extern void
XawOVStopPlayAll (w)
    Widget w;
```

Sets the playAll resource of the OmniViewer to False, taking the OmniViewer out of "play all" mode. This does not stop the current sound playing if any.

```
extern int
XawOVReadFile (w, filename, origin_x, origin_y)
    Widget w;
    char *filename;
    int origin_x, origin_y;
```

Reads in the chat or db file named by filename, placing all its items with respect to origin_x and origin_y as (0,0).

```
extern void
XawOVWriteDB (w, fp)
    Widget w;
    FILE *fp;
```

Calls XawCMWriteDB(), writing out the item database in db file format to the stream pointed to by fp. See ChatMan.doc for more details.

```
extern void
XawOVWriteChat (w, fp)
    Widget w;
    FILE *fp;
```

Calls XawCMWriteChat(), writing out the item database in chat file format to the stream pointed to by fp. See ChatMan.doc for more details.

B.4 MegaSound Widget

/*****

MegaSnd widget

Author: Chris Horner
MIT Media Lab

Updated: 5/7/93

*****/

SUMMARY

The MegaSnd widget ("mega sound") is for viewing large sound files and supports magnification and viewing of the sound at hierarchical levels of detail. It also allows users to view and modify text annotations.

RESOURCES

The MegaSnd widget has the following resources, in addition to those of Core, Composite (Athena), and Bboard widgets. It is a subclass of the Bboard widget. See the Bboard widget documentation for additional details on Bboard resources.

soundName

resource name	XtNsoundName
resource class	XtCSoundName
resource type	String
default	NULL

This is the name of the sound file for the sound system.

fileDescriptor

resource name	XtNfileDescriptor
resource class	XtCFileDescriptor
resource type	Int
default	-1

This is the Unix fd for the sound file. It is passed to the sound system.

If this is less than zero, the widget is set to insensitive at Initialize or SetValues time.

segmentDuration

resource name	XtNsegmentDuration
resource class	XtCDuration
resource type	long
default	0

This is the time duration for the data. The segmentDuration gives the actual length of the sound. This is a long.

```

segmentStart
    resource name      XtNsegmentStart
    resource class     XtCStart
    resource type      long
    default            0

```

This is the time offset for the start of the actual data which is not necessarily the beginning of the file.

```

whenToScale
    resource name      XtNwhenToScale
    resource class     XtCWhenToScale
    resource type      long
    default            0

```

This is the maximum duration in ms the soundViewers will be in truncated mode, beyond which they are shown in scaled mode.

```

borderPix
    resource name      XtNborderPix
    resource class     XtCBorderPix
    resource type      Pixmap
    default            NULL

```

This is the pixmap of the border for the entire composite.

```

vertDist
    resource name      XtNvertDist
    resource class     XtCVertDist
    resource type      int
    default            10

```

This is the vertical spacing between layers of the hierarchy as well as between the text entry widget and the top level soundViewer.

```

flagNotes
    resource name      XtNflagNotes
    resource class     XtCFlagNotes
    resource type      Boolean
    default            True

```

This controls whether or not sound annotations for the sound are "flagged" with command widgets, and whether the text entry widget for these notes is shown.

```

autoMode
    resource name      XtNautoMode
    resource class     XtCAutoMode
    resource type      Boolean
    default            True

```

When this is true, the MegaSound automatically creates a two layer hierarchy with the beginning of the file in the zoom window. Also, when a widget lower in the hierarchy reaches the end, a new widget is created.

```

autoWindow
    resource name      XtNautoWindow
    resource class     XtCAutoWindow
    resource type      long
    default            60000

```

This is the duration in ms of the default zoom window for autoMode.

```

backgroundPixmap
    resource name      XtNbackgroundPixmap
    resource class     XtCPixmap
    resource type      Pixmap
    default            XtUnspecifiedPixmap

```

This is the background pixmap for the widget. Lines connecting layers of the hierarchy, and connecting annotation flags to the top soundViewer are superimposed on this background.

CALLBACKS

There are no callbacks for this widget.

TRANSLATIONS

The MegaSnd itself does not have translations; however default translations for the composite's children should be specified in the application defaults file as follows:

```

*MegaSnd.SoundViewer.translations: #override \
    <Btn2Up>:      StopSound() StopIfIndicator() MoveIndicator() \
                  MoveSound() StartIfSound() StartIfIndicator()
NewMSWindow()\n\
    <Key>e:        ExpandSel() \n\

*MegaSnd.Text.translations: #override \
    <Enter>:       no-op() \n\
    <Key>Return:  EnterNote()

*MegaSnd.Command.translations: #override \
    <Enter>:       MakeHotNote() \n\
    <Btn1Up>:     PlayNote()

```

ACTIONS

The actions for the MegaSnd widget are:

```

MakeHotNote(w, xevent)
    Widget w;
    XEvent *xevent;

```

This should be called from an annotation flag (a command widget) on entry. It displays the note's contents in the MegaSnd's text entry widget, and lets the user modify the contents.

```
EnterNote(w, xevent)
    Widget w;
    XEvent *xevent;
```

This should be called from the MegaSnd's text entry widget, to set the contents of the current note to whatever's in the text widget.

```
PlayNote(widget, event, params, num_params)
    Widget widget;
    XEvent *event;
    String *params;
    Cardinal *num_params;
```

This should be called from an annotation flag on the Return key. It jumps to the sound location marked by the flag, and plays the sound from that location.

```
ExpandSel(w, xevent)
    Widget w;
    XEvent *xevent;
```

This should be called from a soundViewer in the MegaSnd. It calls XawMSExpandSel() (see below).

```
NewMSWindow(w, xevent)
    Widget w;
    XEvent *xevent;
```

This should be called from the top level soundViewer widget to create a zoom window starting at the current top level position and extending for autoWindow ms, as specified by the resource.

CONVENIENCE ROUTINES

```
extern Widget
XawMSLastUsed (w)
    Widget w;
```

Returns the widget id of the last soundViewer widget the user played.

```
extern void
XawMSStopSounds(w)
    Widget w;
```

Calls XawStopSV() and XawStopIndicator() on the last soundViewer widget the user played, if any.

```
extern Widget
XawMSLevelToWidget (w, level)
    Widget w;
    int level;
```

Returns the widget id of the soundViewer at the specified zoom level (0 == top level).

```
extern int
XawMShotNoteIndex(w)
    Widget w;
```

Returns the number of the currently viewed visible text annotation.

```
extern void
XawMShotNoteInfo(w, soundStart, soundStop, soundLen)
    Widget w;
    long *soundStart, *soundStop, *soundLen;
```

Modifies the parameters to reflect the characteristics of the currently viewed text annotation.

```
extern void
XawMSMakeHotNote(w, index)
    Widget w;
    int index;
```

Makes the visible text annotation with the given index the "hot note", i.e. the one that will be accessed if XawMShotNoteInfo(), XawMShotNoteIndex() or XawMSDisplayHotNote() (see below) is called.

```
extern void
XawMSDisplayHotNote(w)
    Widget w;
```

Displays in MegaSnd's text entry widget the contents of the "hot note".

```
extern void
XawMSEExpandLevel(w, start, stop)
    Widget w;
    long start;
    long stop;
```

Sets the MegaSnd bottom level zoom region to the specified boundaries (in ms).

```
extern void
XawMSEExpandSel(w)
    Widget w;
```

If any soundViewer in MegaSnd owns the selection, it sets the zoom region to this selection, otherwise calls NewMSWindow() (see above).

```
extern void
XawMSCollapseLevel(w)
    Widget w;
```

Removes all zooming from the MegaSnd, leaving the single top level soundViewer.

```
extern void
XawMSShowNote(w, noteNum)
    Widget w;
    int noteNum;
```

Shows in MegaSnd's text entry widget the contents of the visible note with the given noteNum.

```
extern void
XawMSShowSpeaker(w, noteNum)
    Widget w;
    int noteNum;
```

Shows in MegaSnd's text entry widget the contents of the visible note with a new speaker index of noteNum.

```
extern void
XawMSShowStory(w, noteNum)
    Widget w;
    int noteNum;
```

Shows in MegaSnd's text entry widget the contents of the visible note with a new story index of noteNum.

```
extern long
XawMSStoryToSpeaker(w, index)
    Widget w;
    long index;
```

Returns the speaker index of the story with the given index.

```
extern long
XawMSTimeToSpeaker(w, soundStart)
    Widget w;
    long soundStart;
```

Returns the speaker index of the note with the given start time (in ms).

```
extern long
XawMSSpeakerToVisNote(w, speakerNum)
    Widget w;
    long speakerNum;
```

Returns the index of the visible note with the given speaker index of speakerNum.

```
extern void
XawMSSearchCapForKeyword(w, keyword)
    Widget w;
    char *keyword;
```

Hides any existing visible notes, and makes visible only those containing the keyword.


```
extern void  
XawMSShowAllNotes(w)  
    Widget w;
```

Makes all notes visible.

```
extern void  
XawMSShowStoryNotes(w)  
    Widget w;
```

Hides any existing visible notes, and makes visible only those which are new story headlines.

Appendix C

Keywords for News Categorization

These keywords were used by the **karma** program, and were derived empirically from a week of news to demonstrate the concept of story categorization. They are not meant to be a complete set of such keywords.

International News Keywords

AIR STRIKES
AMMUNITION
ARTILLERY
BOMB
COMBAT
EMBASSY
ESPIONAGE
INFANTRY
MILITARY
MISSILE
SOLDIER
SPY
TROOP
WAR
BOSNIA
BRITAIN
BRITISH
CANADA
CANADIAN
CHINA
CHINESE
CROAT
EGYPT
FRANCE
FRENCH
GERMAN
HONG KONG
IRELAND
IRISH
ISRAEL
JAPAN
LONDON

PARIS
RUSSIA
SERB
SOMALIA
UNITED NATIONS
PRIME MINISTER
YELTSIN
JOHN MAJOR
CATHOLIC
CHRISTIAN
HINDU
HINDI
ISLAM
JEW
MOSLEM
MUSLIM
PROTESTANT

National News Keywords

ATLANTA
BOSTON
CHICAGO
DALLAS
DETROIT
HOUSTON
NEW YORK
LOS ANGELES
L.A.
SEATTLE
WASHINGTON
AMERICA
CALIFORNIA
COLORADO

FLORIDA
MASSACHUSETTS
TEXAS
UNITED STATES
U.S.
CAPITOL HILL
CONGRESS
DEMOCRAT
FEDERAL
LAW
PENTAGON
REPUBLICAN
SUPREME COURT
WHITE HOUSE
AIR FORCE
ARMY
MARINES
NAVY
ATTORNEY
GENERAL
CHIEF OF STAFF
CONGRESSMAN
PRESIDENT
SENATOR
CLINTON
STEPHANOPOULOS

**Economic/
Business News
Keywords**

\$
BOND
BUSINESS
COMPANY
DOW JONES
ECONOM
EMPLOY
FACTORY
FACTORIES
GROSS DOMESTIC
PRODUCT
G.D.P
GROSS NATIONAL
PRODUCT
G.N.P
INCOME
INDUSTRY
INTEREST RATE
JOB
MANAGEMENT
MANAGER
MANUFACTUR
MARKET
MONEY
RECESSION
STOCK
TAX
TARIFF
WALL ST
YEN

**Entertainment
News Keywords**

ACADEMY AWARD
ACTOR
ACTRESS
ACTING
ALBUM
CINEMA
COMEDIAN

CRITIC
ENTERTAIN
FILM
GRAMMY
HOLLYWOOD
MUSIC
MOVIE
POET
REVIEW
SINGER
TELEVISION
WRITER

**Medical News
Keywords**

AIDS
CANCER
CURE
DISEASE
DOCTOR
DRUG
HEALTH
HOSPITAL
INTENSIVE CARE
MEDIC
NURSE
NURSING
OPERATING ROOM
PHYSICIAN
SURGEON
SURGERY

**Sports News
Keywords**

ATHLET
COACH
BASKETBALL
BASEBALL
FOOTBALL
GOLF
HOCKEY
RACING
SKIING

SPORTS
TEAM
TENNIS

**Weather News
Keywords**

BLIZZARD
BREEZE
CELSIUS
CLOUD
DEGREES
FAHRENHEIT
FORECAST
HIGH PRESSURE
LOW PRESSURE
LIGHTNING
RAIN
SHOWERS
SLEET
SNOW
STORM
SUN
THUNDER
WEATHER
WIND

Appendix D

Description of NewsTime Auxiliary Programs

This section describes all of the supplementary programs that assist NewsTime, in order of their appearance in the thesis.

Source	Usage
paws.c	<p>paws <i>sound-filename</i> <i>silence-level</i> <i>silence-length</i></p> <p>Given a sound file and the segmentation parameters for maximum silence level, and minimum silence length, performs speech and silence detection on the sound file, and adds the segmentation database to the sound's auxfile.</p>
rec30min.c	<p>rec30min <i>sound-filename</i></p> <p>Records audio on a NeXT computer from the microphone for half an hour. Output is an 8012 Hz, 8 bit mu-law NeXT audio file with the given name. Original code from NeXT demos.</p>
getcaption.c	<p>getcaption <i>seconds</i></p> <p>Reads raw closed-caption data from a tty port on a NeXT computer, connected to a video line 21 decoder. Runs for the requested number of seconds, dumping the output onto stdout. Timestamps since the beginning of the recording are placed at new speaker and story boundaries, in the format <i><sec msec<</i>. The raw data contains control characters and formatting strings, and is all uppercase letters, making it almost illegible:</p> <pre>^@^@^@^T&^T&^T-^T-^Tr^Tr>><8 321<<8 353< GOOD EVENING.^@^@^@^T&^T&^@^@^@^@^@^T&^T&^T-^T- ^Tr^TrWE BEGIN IN WASHINGTON^@^@^@...</pre> <p>To make it more readable it is passed through a Unix pipeline of utilities, remnull, grok, remhdr, capper, firstwordCAP & specialCAP. These utilities strip away all non-alphabetic characters, capitalize only relevant words, and produce paragraphs for each new speaker or story. These paragraphs retain the timestamps from getcaption, e.g.</p> <pre>>> <8 321<<8 353< Good evening. We begin in Washington...</pre> <p>Original getcaption and filters written by Alan Blount.</p>

cap2db.c

cap2db

A lex based parser which reads from stdin a filtered, cleaned up closed-caption transcript, and writes to stdout an annotation database compatible with the SoundViewer's auxfile format, as well as the ChatMan's database format.

syncs.c

syncs *sound-filename*

Given a sound file whose auxfile contains an annotation database with raw closed-caption time-stamps, first performs speech and silence segmentation on the sound file if necessary, then synchronizes the original time-stamps to reflect the actual story and speaker boundaries following pauses.

karma.c

karma *sound-filename*

Reads in the auxfile for the given sound file, which should contain in the annotation database, the synchronized transcript from **cap2db** and **syncs**. Generates a content database, categorizing each story according to its most salient keywords, and adds it to the original auxfile.

music.c

music [-l music-level] [-d music-duration] *sound-filename*

Reads the given NeXT audio file and runs Hawley's music detection algorithm on it. Output is directed to stdout; each line contains three numbers pertaining to a single segment:

segment-duration *segment-start* *music-or-not*

segment-duration is the length (in ms.) of the segment.
segment-start is the offset in the file (in ms.) of the segment.
music-or-not contains a 1 if entire segment is above the music threshold, i.e. is music, or 0 if entire segment is below the threshold.

The segments do not have to be continuous and will not generally comprise the entire sound-file. This is because segments less than the minimum duration are ignored.

The -l paramter sets the minimum sustained striation level for music (default is 5.6). The -d parameter sets the minimum duration window for music (default is 3 seconds). Original code written by Mike Hawley.

music2db.c **music2db**

Reads from stdin the output of the music program, and writes to stdout a content database compatible with the SoundViewer's auxfile format.

Bibliography

- [Aro91] B. Arons. Techniques, Perception, and Applications of Time-Compressed Speech. In *Proceedings of 1992 Conference*, pages 169-177. American Voice I/O Society, 1992.
- [Aro92a] B. Arons. Techniques of Adaptive Speech Detection. Technical report, MIT Media Lab, April 1992.
- [Aro92b] B. Arons. Tools for Building Asynchronous Servers to Support Speech and Audio Applications. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 71-78, 1992.
- [Aro93] B. Arons. Interactively Skimming Recorded Speech. Submitted for publication, 1993.
- [AS86] S. Ades and D. Swinehart. *Voice Annotation and Editing in a Workstation Environment*. In *Proceedings of 1986 Conference of American Voice I/O Society*, pages 13-28, Sept 1986.
- [Dav93] M. Davis. Media Streams: An Iconic Visual language for Video Annotation. Research paper. MIT Media Lab, 1993.
- [DMS92] L. Degen, R. Mander and G. Salomon. Working with Audio: Integrating Personal Tape Recorders and Desktop Computers. In *Proceedings of CHI '92*. pp. 413-418, 1992.
- [Don86] J. Donath. The Electronic Newstand: Design of an Intelligent Interface to a Variety of news Sources in Several Media. Master's thesis. MIT Media Lab, 1986.
- [Eli93] E. Elliott. Watch, Grab, Arrange, See: Thinking with Motion Images via Streams and Collages. Master's thesis. MIT Media Lab, 1993.
- [Fel] L. Feldman. Radio with Pictures. *Video Review* 12:16, March 1992.
- [Has93] K. Hasse. Multi-Scale Parsing Using Optimizing Finite State Machines. To appear in *ACL-93*, 1993.
- [Haw93] M. Hawley. Structure out of Sound. Doctoral thesis in preparation. MIT Media Lab, 1993.

- [Hin92] D. Hindus. Semi-Structured Capture and Display of Telephone Conversations. Master's thesis. MIT Media Lab, 1992.
- [HS92] D. Hindus and C. Schmandt. Ubiquitous Audio: Capturing Spontaneous Collaboration. In *Proceedings of CSCW '92*, November 1992.
- [MCW92] M. Mills, J. Cohen and Y. Y. Wong. A Magnifier Tool for Video Data. In *Human Factors in Computing Systems, CHI 92 Proceedings*, pages 93-98, 1992.
- [Mye85] B. Myers. The Importance of Percent-Done Progress Indicators for Computer-Human Interfaces. In *Human Factors in Computing Systems, CHI 85 Proceedings*, pages 11-17, ACM 1985.
- [Pet89] C. Peterson. *Athena Widget Set - C Language Interface, X Window System, X Version 11, Release 4*. MIT X Consortium, 1989.
- [Pin91] N. Pincever. If you could see what I hear: Editing assistance through cinematic parsing. Master's Thesis. MIT Media Lab, 1991.
- [RV92] P. Resnick and R. Virzi. Skip and Scan: Cleaning up Telephone Interfaces. In *Human Factors in Computing Systems, CHI 92 Proceedings*, pages 419-426, 1992.
- [Sch81] C. Schmandt. The Intelligent Ear: A graphical interface to digital audio. In *Proceedings IEEE Conference of Cybernetics and Society*. pages 393-397, October 1981.
- [Smi89] T. A. Smith. If You Could See What I Mean... Descriptions of Video in an Anthropologist's Video Notebook. Master's Thesis. MIT Media Lab, 1989.
- [Sti91] L. Stifelman. Not just another voice mail system. In *Proceedings of 1991 Conference*, pages 21-26. American Voice I/O Society, 1991.