Contents

List of Figures

Chapter 1

Introduction

Audio media is one of the most neglected areas in the Hypermedia / Multimedia domain. To play sound files in traditional hypermedia systems, for example the World Wide Web (WWW), users click upon links (or hyperlinks) within a browser. This action will download the sound file to the local machine and then activate an application to play the file. More recently however, applications can stream the file over the network instead, thus reducing the overhead of having to download it first.

In multimedia applications such as Microsoft Encarta, users click on areas of the screen called "hot-spots" that will cause an event to occur. This event could cause an audio file to be played, a picture to be displayed or it could cause some other event. In both of these systems audio is usually just the result or side-effect of some action caused by the user.

In the Hypermedia domain the information required to traverse the links is embedded within the document using the HyperText Markup Language (HTML). In multimedia systems the link information is stored in an internal format, which the user does not have access to. This creates problems because only the original author of the hypertext or multimedia document can modify, edit or create new links. Therefore because this link information is difficult to modify, these systems are known as "closed" systems.

The aim of this research is to investigate how audio could be used in Open Hypermedia Systems, specifically how links can be used with streaming audio.

Chapter 2 introduces the audio domain and describes how audio has been used in multimedia authoring tools, "Open" Hypermedia Systems and the World Wide Web.

This chapter also describes existing standards used in both the hypermedia and multimedia communities and the development of a simple audio tool.

Chapter 3 describes the traditional internet protocols, TCP/IP and the protocol used to transport real-time information over TCP/IP, the Real-time Transport Protocol (RTP). A new streaming protocol, the Real Time Streaming Protocol (RTSP), is then discussed.

Chapter 4 gives an overview of the original Soundviewer Tool and how it has been used. The design and implementation of a *streaming* audio extension to this tool is also discussed. Finally, an example scenario is outlined for the use of this new tool.

Chapter 5 assesses the work that has already been done and explores some possibilities for future work.

Chapter 2

Background

This chapter describes the audio domain and how it is used in a number of systems, specifically multimedia authoring tools, the World Wide Web and "Open" Hypermedia Systems. Existing standards, such as *MHEG* and *HyTime* are discussed and the chapter draws to a close by describing how a simple audio tool can be created using a modern programming language e.g. Visual Basic.

2.1 Introduction: The Five Senses

Traditionally, vision has always been regarded as the primary sense for normally sighted people. In general, however, humans interact with the outside world by receiving information through a *mixture* of the five senses[1], processing this information and then reacting or responding to it. For example, the sense of smell and sight could be used to determine if a piece of food had gone off. The user might react to this situation by disposing of the food. In the development of graphical user interfaces, all of the other senses are regarded as being less important than sight. This is mainly due to the complexity of the other senses.

Dix et al. [1] describe how the majority of interactive computer systems are completely visual in nature, offering rudimentary audio support. As the complexity of these systems increase, more and more visual information could be presented on the screen, making it harder for the user to understand. The authors discuss how the other sensory

[1] Sight, hearing, touch, taste and smell.

channels[2] could be used to relieve the pressure of the visual channel and thus reduce the information overload. By increasing the number of sensory channels, users would be able to interact with their computers in the same way that they would interact with their everyday environment.

Most commercial computer systems, however, provide limited support for two of the other senses, *hearing* and *touch*. Systems that produce a *haptic* response (the sense of touch) are being used by the virtual reality (VR) community and joystick manufacturers. For example, VR users can wear a special type of glove which contains small inflatable pockets. As the user in the VR world picks up an object, these pockets fill with air giving the impression that the user has actually picked up that object. Joystick manufacturers have developed *tactile feedback* joysticks which have small motors that move the joystick depending on the type of feedback required.

These interactive systems also support audio, although traditionally it is only used to provide warnings, alarms and status information. Most modern operating systems such as Microsoft' s Windows 95 or NT also provide support for soundcards, which can be used to record, edit and playback audio samples. The majority of the time, however, soundcards are used for playing games. Dix et al. [1] describe how users, when playing a game, will score more points when the sound is turned on rather than off. Users can detect vital information and clues via the changes in the sound, which can be used with the visual information to increase their scores. The authors also describe how audio and visual information can help increase the accuracy of speech recognition systems. For example a camera can be used to video the lip movements of the speaker. The sounds will also be analysed. By using the video footage and the sound information, words and phrases can be more accurately resolved.

Overall these interactive systems use the visual channel as the main medium for transferring information. The auditory channel is rarely used, although the amount of information that can be conveyed using audio is underestimated. The following section describes the audio domain in more detail.

[2]There are two types of channel, input and output. With humans an input channel represents one of the five senses, whilst an output channel represents a response, e.g. moving a leg, walking, talking etc.

2.2 The Audio Domain

To understand the audio domain an overview of the human auditory system is required. Dix et al. [1] and Moore [2] both describe how the human ear consists of three parts and these are:

1. The *Outer Ear* which consists of the *pinna* (the visible part of the ear) and the *auditory channel*. Both the pinna and the auditory channel amplify certain high frequencies, whilst the channel itself secretes a waxy substance that prevents insects and dirt from reaching the more sensitive middle ear.

2. The *Middle Ear* consists of a small cavity containing three of the smallest bones in the human body, the *ossicles*. These connect the outer ear via the eardrum or *tympanic membrane* to the *cochlea* in the inner ear.

3. The *Inner Ear*. This consists of the cochlea which has rigid bony walls and is filled with a special type of fluid. The cochlea also contains tiny hairs or *cilia* which move when the fluid vibrates. This vibration causes small electrical impulses to be passed up the auditory nerve to the brain.

Moore [2] describes how the process of hearing originates with the vibration of an object. This vibration causes a pattern of changes to occur within the surrounding medium (usually air), which results in the creation of a *sound wave*. This wave travels through the air until it eventually reaches the outer ear, mentioned above. The sound wave will then pass down the auditory channel to the ear drum, causing the drum to vibrate. The vibrations of the ear drum are passed via the ossicles, in the middle ear, to the cochlea. These vibrations change the pressure within the cochlea, which in turn moves the cilia.

This process of passing the sound waves from the outer to the inner ear, ensures the efficient transfer of the actual sound information. Dix et al. [1] describe the different characteristics of sound, such as the pitch and amplitude. If the pitch of the sound increases, then the frequency[3] will also increase. The amplitude of the sound is proportional to its loudness and therefore, an increase in the amplitude will cause an increase in the volume of the sound.

[3]The human ear can hear frequencies from about 20 Hz to 15 KHz.

The human ear can also identify the sound's location, since the two ears receive slightly different sounds. If a sound occurs to the left of a users head, then the left ear will receive the sound wave first. It will take longer to reach the right ear due to its location and the fact that the wave will also reflect off the users head.

Overall, sound can convey a remarkable amount of information. The human ear can use this information to detect different types of sounds, where they are coming from and how far away they are. However, the sense of hearing has always been regarded as secondary to that of sight and this can be clearly seen in the development of computer systems. By combining this sound information with visual information, users would be able to interact with computers in a more natural way, see **Section 2.1**.

2.3 Multimedia Authoring Tools

This section describes some of the most popular multimedia authoring tools that are available today. Multimedia authoring allows users to combine text, hypertext, pictures, animation, sound and video into a single application that can be distributed on and over a variety of media, for example the internet, CD-ROMs, floppy disks etc. Users can design anything from an interactive Web site to an electronic product catalogue.

Designing a multimedia presentation, however, can take a lot of time and effort. Therefore the majority of multimedia packages try and reduce this by using a range of authoring techniques. These include the *timeline* and *flowchart* methods and the "*book*" metaphor. The following sections give a brief overview of these methods and the products that use them (a more detailed description of the products are in [3]).

2.3.1 The Timeline authoring method

The leading products in this area of authoring are developed by Macromedia and they are called Director and Flash. In these applications a timeline consists of *layers* which span over several *frames*. Each layer contains one or more elements (*cast members*), which exist in either one frame or they can span over several. For example a simple presentation could contain three layers; layer one could contain a picture of blue sky, layer two could contain a picture of a beach and layer three a picture of a palm tree. If each of the layers span 10 frames and the user presses the "play" button, then the

presentation will show a picture containing all of the elements in the layers e.g. blue sky, a beach and a palm tree. If layer one however only spans 5 frames from the beginning, then the blue sky would only show for 5 frames and then disappear for the remaining 5. Users can also modify a cast member in each frame of a single layer which will result in a simple animation, e.g. modifying a birds wings so that in one frame they are up and in another they are down, giving the impression of flight.

Both of these packages support audio. The Flash application allows users to import audio files directly into a layer, whereas Director uses a separate digital audio layer. **Fig. 2.1** shows a section of a simple presentation in the Flash program. A section of the timeline that contains audio samples, the "speaker flashes" layer, can be seen at the top of the screen. Director has support for up to 10 digital audio channels depending on the hardware used. Audio files can be played in the background of a presentation or they can be activated by several other types of event e.g. a mouse click, the transition from one scene to another etc.

Links can not be followed from within an audio event. Audio can be placed on one layer and at a certain time / frame an event can occur e.g. an URL fired, screen transition, video started etc. on another layer. Everything, however, is followed when the "play head" touches the beginning of a cast member in the timeline. So sound again is just another element used within a presentation.

2.3.2 The Flowchart authoring method

Several products use this method and they include Macromedia's Authorware, Asymetrix's IconAuthor and Linotype's Dazzler. All of these applications use "drag and drop" to pick up and place icons on the presentation page. These icons represent:

- *events* such as mouse clicks, key press,
- *actions* to be performed after an event e.g. a transition, a sound,
- *routines* to perform loops, conditional branches etc.

The flow of control within the flowchart is usually from the top to the bottom. Branches are allowed, i.e. loops, decision and interaction branches etc., and users can also insert smaller flowcharts within the main flowchart. Clicking on each icon usually brings up

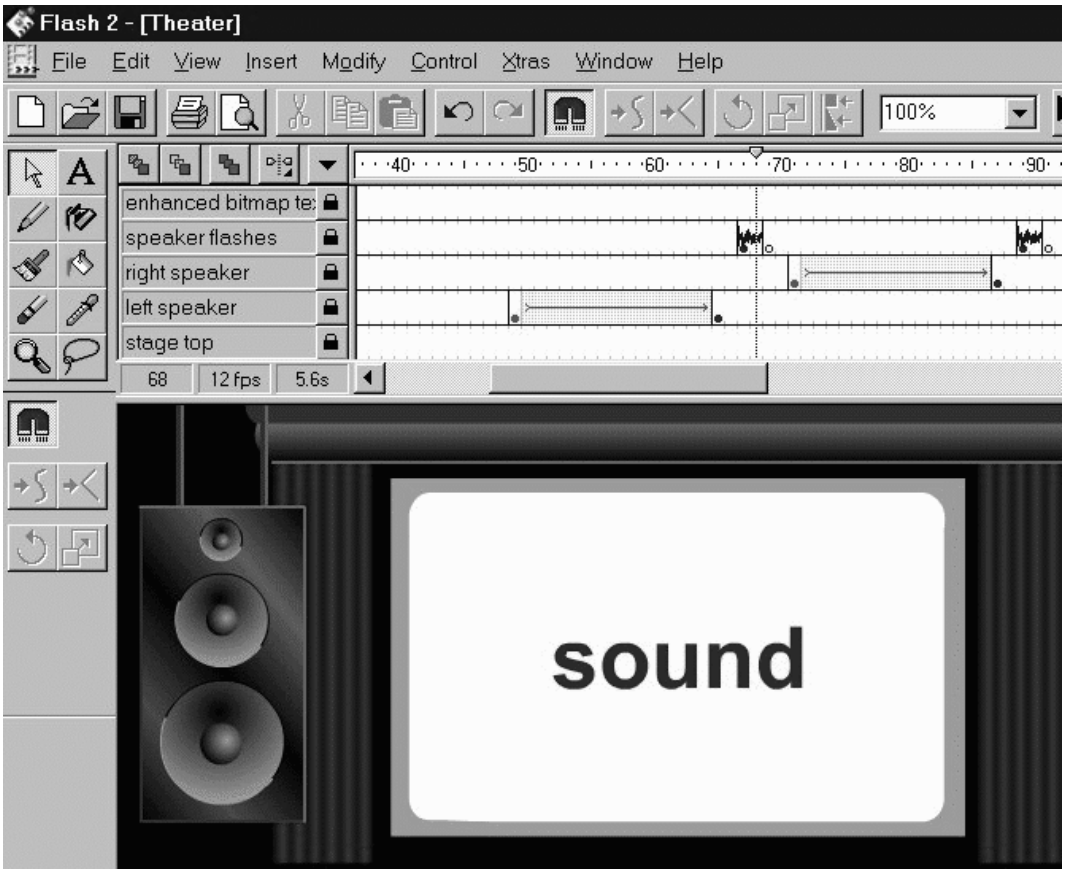

Figure 2.1: A screen-shot of a presentation in Flash.

the icon's properties, which can be easily changed. A presentation is built by inserting one object after another e.g. a simple application could contain just three icons; the first could be a picture, the second a sound icon and the third a text icon. When the presentation is started the user would see and hear all three icons together. **Fig. 2.2** is a screen-shot of Macromedia's Authoware program.

An audio icon can be inserted directly onto the author's development page. The properties of this icon, however, are usually quite limited; it only allows the audio file to be located and imported. Again there is no functionality to create anchors within the audio and, therefore, links to and from the audio.

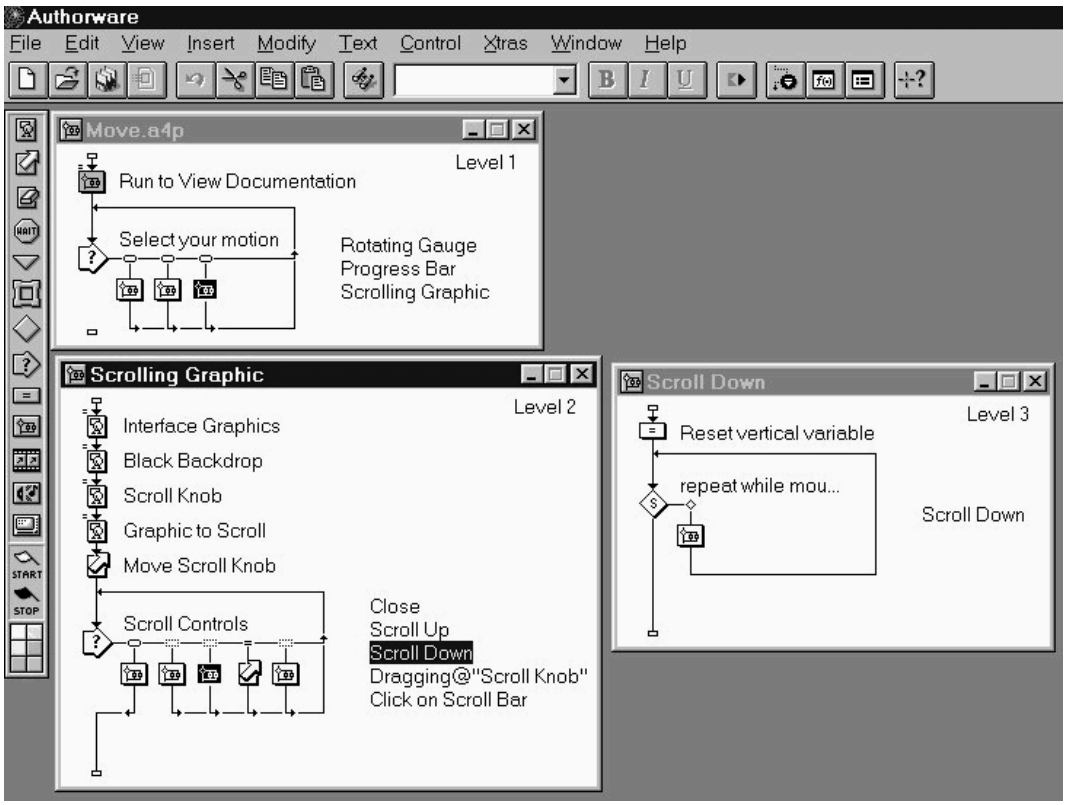

Figure 2.2: A screen-shot of an Authorware presentation.

2.3.3 The "Book" Metaphor

Asymetrix's ToolBook, Digital Workshop's Illuminatus, Scala Computer Television's MM200 and MatchWare's Medi8or all use this method of authoring. Basically when the application is started, the user is shown a *page* in which certain objects can be placed e.g. text, pictures, buttons etc. By inserting objects into several pages, a multimedia "book" is eventually created. The author can create transitions between pages and on the objects themselves e.g. zoom text in and out, cause a picture to flow onto the page etc. **Fig. 2.3** is a screen-shot of the design process used in MatchWare's Medi8or program.

Several of the applications allow a sound object to be placed directly onto the page. Other objects can then be inserted into the page and arranged in such away that they will appear or do something at a certain time, during the audio playback. The audio, however, is not directly controlling these objects and therefore it can be described as

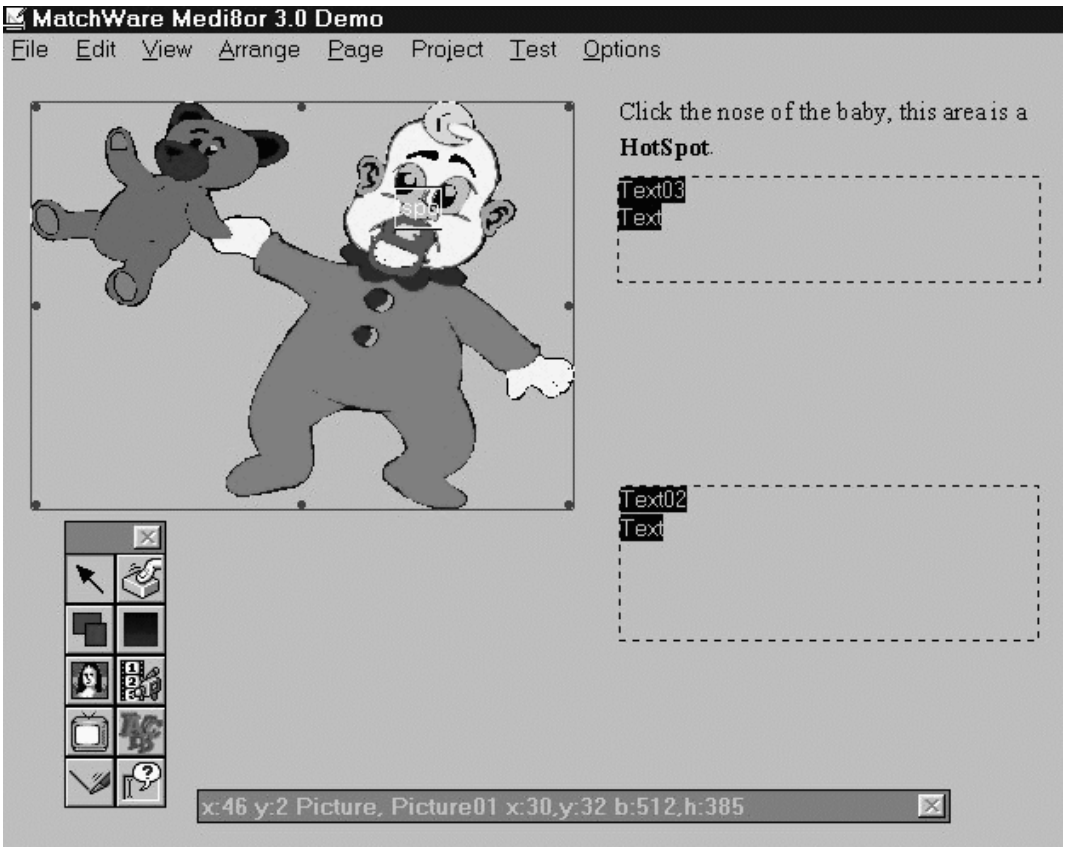

Figure 2.3: MatchWare's Medi8or design process.

being just another entity or object used within the presentation. The majority of the time audio output is caused by an event e.g. a button being pressed, the mouse cursor moving over a hotspot etc.

2.4 The World Wide Web (WWW)

The World Wide Web or W^3 was originally designed in 1990 by *Tim Berners-Lee* at CERN, the European Laboratory for Particle Physics. The aim of the project was to provide a uniform way in which information could be shared over wide-area networks. A paper by T. J. Berners-Lee et al. [4] describes how most of the information at CERN, for example technical reports, data from experiments etc., was already available on-line. However the ability to create references to this material required a reasonable knowledge of host names, terminals, passwords and the CERN network itself. As a consequence it

was very difficult, if not impossible, to create and then "jump" to these references.

The WWW overcame these problems by defining three new platform-independent and network-neutral components. These were:

1. The *Universal Resource Locator* (URL) addressing scheme. This consists of a simple string, which contains the type of protocol to use, e.g. FTP, HTTP etc., the name of the computer on the internet and the actual name of the document to be retrieved. For example "http://www.idiscover.co.uk/linux/linuxdoc/index-.html".

2. The *HyperText Transfer Protocol* (HTTP). This protocol was designed so that information could be efficiently retrieved, for the purpose of making hypertext "jumps"; see **Section 2.6.1** for a description of hypertext.

3. The *HyperText Markup Language* (HTML) is used to markup, see **Section 2.5.2**, documents that will be used on the WWW. This process enables authors to prepare and format their documents using a mixture of text, pictures, sound, video and *hypermedia links*. See **Section 2.6.1** for a description of hypermedia. Users can click on these links to "jump" to related items. By using this language, authors ensure that their documents will look the same on different WWW *browsers*. A WWW browser or client is an application that renders an HTML document into a form that can be displayed on the user's screen. Netscape's Navigator and Microsoft's Internet Explorer are two of the most common browsers.

These three components form the core architecture of the WWW. Originally the WWW was only designed to be used on wide-area networks (WANs). With the growth of the Internet however, see **Section 3.1**, WWW browsers are now being used to read and retrieve information off the internet. As a result the WWW has steadily grown in size (the number of web "pages" being created) and popularity.

The first Web browsers could only be used to display text and pictures. Hypertext links were displayed in a different colour and users could "click" on these, with a mouse, to follow the link. With the development of more sophisticated browsers, however, users could click on an "audio" link, which would make the browser download the file to the users machine. The user could then use another application to play the file. Some browsers, for example Netscape's Navigator, could also execute a program to play the

file once it had been downloaded. The process of downloading a file, however, could take a long time especially if the file was quite large and / or the network connection was very poor.

To overcome this problem, several companies, such as RealNetworks, Macromedia, Xing Technology etc., have created *streaming audio* servers and players. A streaming audio link is represented on a web page in exactly the same way as an audio link. This link will contain information about the audio server and the audio file to be streamed. When a user clicks upon this link, three events occur and they are:

1. The browser activates the relevant streaming audio player. This player will then send a message, containing the name of the audio file to be streamed, to the audio server.

2. The audio server splits the audio file into smaller packets and transmits them to the player.

3. The player buffers these packets until enough have been received, so that they can be played.

On a reasonably fast network connection the audio file is played almost immediately, giving the impression that the audio file is stored on the local machine. Otherwise it could take a long time for enough information to be received so that it can be played.

The current internet protocol, which the WWW uses, was not designed to handle streaming audio information. This protocol is called TCP/IP and is described in more detail in **Section 3.1**. To overcome this problem several protocols have been developed and they are the *Real-time Transport Protocol* (RTP) and the *Real Time Streaming Protocol* (RTSP). These protocols are discussed in more detail in **Sections 3.2** and **3.3**, respectively.

Overall the WWW uses audio as just another medium to link to. Anchors can not be created within the audio stream or file and therefore links can not be created from an anchor within the audio. Links can be created to a time within the audio file, but this is not a true anchor, it is usually just a subrange of the entire file.

2.5 Existing Standards

Several international standards have been created in the hypermedia and multimedia community. These are described in more detail in the following sections.

2.5.1 The Multimedia and Hypermedia Experts Group

The Multimedia and Hypermedia Experts Group[4] (MHEG) was formed by a subcommittee[5] of the International Standards Organisation (ISO) to address the problems of trying to design a software-neutral interactive multimedia presentation tool. There are several parts to MHEG and a paper by Rodriguez et al. [5] gives a brief description of MHEG parts 1 to 5.

Boudnik and Effelsberg [6] describe MHEG-1, which was the initial specification for robust multimedia data structures using ISO's own Abstract Syntax Notation (ASN.1). This notation describes all of the data structures (or MHEG classes) in an abstract syntax, which ensures that applications conforming to this standard will be able to communicate. An interactive presentation is formed by creating instances of these MHEG classes or *MHEG-Objects* and forming interrelationships between these objects. MHEG-1 defines several classes, of which four could be used for inserting links into streaming audio. These are:

- The *MH-Object* class which is the root class inherited by all of the other classes.
- The *Content* class which contains the data that the user sees and / or hears.
- The *Link* and *Action* classes which describes what actions are performed when a user activates a particular link object.
- The *Multiplexed Content* class which is derived from the content class and either contains or refers to the multiplexed stream data. It also assists in inter-stream synchronisation, such as lip synchronisation.

Each object when it is created can contain extra information about its original size and its play-out duration. This information is stored as *virtual co-ordinates* from a generic space and a *virtual timeline*, respectively. At run-time this extra information is

[4] Also known as the Multimedia and Hypermedia information coding Experts Group.
[5] "Coding of Audio, Picture, Multimedia and Hypermedia Information," ISO/IEC JTC1/SC29.

converted into real-time requirements, such as screen co-ordinates and a particular type of timer, depending on the type of hardware being used.

MHEG-2 is exactly the same as MHEG-1, except that the classes are defined in SGML (Standard Generic Markup Language) instead of ASN.1. SGML is described in more detail in **Section 2.5.2**.

A paper by Rutledge et al. [7] gives an overview of the MHEG-3 project which is an extension to MHEG-1. This part of the standard was created to increase the interactivity between multimedia objects and the environments that they run in. This is achieved by using :

1. *Scripting languages*[6] - a script is a program containing a set of procedures that can be used to monitor events that are created by particular objects, for example a user clicking on a button will cause an event. As a result of these events being generated, the script will execute certain actions, possibly on other objects. This allows objects to interact with each other.

2. *A Virtual Machine* - this is used to create a mapping table between the run-time services provided by a particular platform / environment and the interface of the scripting language, used to generate the script. This allows interactivity between the scripts and the platform being used for the presentation.

Overall MHEG-3 increases the functionality of MHEG-1, by allowing users to create presentations using external programs.

The fourth part of the MHEG standard (MHEG-4) was a simple extension and is used to register objects and formats supported by MHEG, e.g. MPEG, JPEG etc.

Joseph and Rosengren [8] give an overview of the MHEG-5 standard, which was designed to extend the class hierarchy of the initial MHEG-1 specification. This extended hierarchy contains a set of new classes that can be used to develop client / server multimedia applications across platforms with limited resources. This ensures that MHEG-5 conformant applications will run on conformant terminals.

MHEG-5 defines five classes that are relevant to the streaming audio and open hypermedia research domain. These are:

[6] These are also known as *Scriptware*.

- The *LinkEffect* and the *LinkCondition* classes. A LinkCondition is triggered by an event, which if it matches certain conditions will run a LinkEffect. The LinkEffect contains a list of elementary actions to be carried out.
- The *Stream* class, which provides the functionality to multiplex audio and video objects, present them in synchronisation and create links from MHEG objects to specific points in the stream (stream-events). Links can also be created to specific user-defined events.
- The *Audio* class, which encapsulates audio objects and can be used with streams.
- The *HyperText* class, which allows users to associate objects with a link to, for instance, another page. Objects can be words, groups of words, pictures etc.

An example of an MHEG-5 application is the "*GlassWWWay*" tool developed by Geyer et al. [9]. The original system, known as the "*GLobally Accessible ServiceS*" (GLASS), used MHEG-1 to provide several different services such as video on demand, interactive TV or online shopping in an easy-to-use interface. However, due to the problems of inferior browser technology and that the developers thought that MHEG-1 was too abstract, they decided to develop a new system based on the MHEG-5 standard. By re-defining the MHEG-5 classes in *Java*, they designed a Java applet which could be downloaded into any Java-enabled World Wide Web (WWW) browser. This also provided better communication between the users and the GLASS server.
This paper concentrates mainly on the benefits and drawbacks of using Java to implement the MHEG-5 engine and so the audio domain is not mentioned. It is not known whether this system provides the functionality to stream audio or to create links from certain points within an audio stream, which MHEG-5 can. The system does provide basic sound support for interactive television and the other services mentioned above. At this stage, however, Java has very limited sound capabilities and therefore the functionality needed to support sound would have to be supplied by external functions[7].

The MHEG standard contains many sections of which part five seems to provide enough information to develop a tool to create links to and from the streaming audio domain. MHEG at the moment, however, is used as a tool to develop visual, easy-to-use, interoperable client / server systems, of which audio is just a small part of this.

[7] These are also called *native functions* and are usually specific to a particular platform.

2.5.2 The Hypermedia/Time-based Structuring Language

The Hypermedia / Time-based Structuring Language (HyTime) became an ISO standard (ISO/IEC 10744:1992) in 1992 and uses the *Standard Generic Markup Language* (SGML) to describe document architectures. HyTime evolved from the work of the ANSI X3V1.8M committee on the development of a *Standard Music Description Language* (SMDL) and to understand the concepts of HyTime, a brief description of SMDL and SGML will also be given in this section.

A paper by Carr et al. [10] describes traditional markup and the equivalent logical and physical markup techniques in use today. Markup can be described as the process of inserting specific commands or codes into the original text which the compositor program (or traditionally a human) uses to compose the final document. Physical markup requires the user to explicitly place certain commands, such as 'bold' and 'centre', around the text to be rendered[8]; whereas logical markup requires the program to interpret and render abstract commands, such as "this text is a heading", inserted by the user.

This paper describes the Standard Generic Markup Language as taking logical markup to the extreme. SGML does not define default markup commands or *tags*; the user creates these by using SGML's logical *elements* and physical *entities*. In ArborText's SGML White Paper [11], logical elements are described as pieces of data that may contain either text or other subelements such as chapters and paragraphs etc. Each element also contains their own generic identifier (GI), which is used for the markup of documents. Physical entities are described as self-contained pieces of data, e.g. a separate text file, a separate graphic file etc., that can be referenced as a unit.

These elements and entities are then stored in a *Document Type Definition* (DTD) file, which can be used as a template for the structure of other documents. The syntax of DTDs is very strict which ensures that they will be understood by other SGML-compliant applications. It is upto the application, however, to interpret the 'meaning' of the document structure.

Mounce [12] describes in his paper, how the *Standard Music Description Language* (SMDL) was initially defined using a SGML DTD in 1988. SMDL is defined[9] as "an

[8] The process of turning non-graphical information into a form that can be represented in a graphical way e.g. a printer renders information into a form that can be printed.

[9] ISO/IEC Draft International Standard 10743.

architecture for the representation of music information, either alone, or in conjunction with text, graphics or other information needed for publishing or business purposes." Basically, SMDL divides a musical work into four domains and these are:

1. *The Logical Domain or cantus.* This will contain all of the logical information about a piece of music. The cantus can be described as an abstract timeline on which all events can be scheduled e.g. Eliens et al. [13] suggest that the cantus can contain information to do with automated lighting.

2. *The Visual Domain.* This graphically represents the music in some form e.g. a link to an image file (JPEG, GIF etc.) or to a coded music file.

3. *The Gestural Domain.* This can contain one or more links to the actual performance of the musical work e.g. a link to a MIDI file.

4. *The Analytical Domain.* This contains commentaries or theoretical analyses of all of the other domains.

A file written in SMDL would traditionally contain a cantus, links to one or more graphical representations of the musical work and one or more links, again, to the actual performance of the cantus. This initially provided enough information to create simple presentations, but with the development of digital audio and multimedia, there was a need to extend the functionality of the original SMDL document type definition (DTD). With each subsequent development of the DTD, more and more information was submitted until eventually a separate standards activity was initiated in 1989. This research produced the *Hypermedia/Time-based Structuring Language* (HyTime) which extended SGML and became a full International Standard in April 1992[10].

Newcomb et al. [14] describe how HyTime was originally defined using a SGML DTD, which contained the HyTime-specific generic identifiers (GIs). The standards committee realised, however, that these identifiers could not be changed because HyTime compliant applications needed them to recognise HyTime documents. This reduced the expressiveness of SGML and as a result, the next draft of the HyTime standard replaced the GIs with SGML *architectural forms*. In Newcomb's [15] paper, an architectural form defines elements with a standard meaning, e.g. independent hyperlink, and syntax for

[10]ISO/IEC 10744:1992.

its associated data. An architectural form also defines an attribute type for the element, which is used for identification purposes. Multimedia systems would then use this identifier to access only the relevant elements and hence parts of HyTime that they require. HyTime usually provides a standard set of forms to build hypermedia / multimedia documents and therefore, is often referred to as a *meta-DTD*.

One of the main problems of the hypermedia / multimedia industry is its inability to store, describe and transport media objects in an application-neutral manner. Newcomb [16] discusses how the HyTime / SGML paradigm addresses this problem, by encapsulating the information using abstract semantics. This ensures that the information can co-exist in a variety of applications and contexts.

HyTime consists of six main modules. A paper by Newcomb [15] and Newcomb et al. [14] give a thorough description of each one. They are:

1. *The Base Module* which includes facilities to manage documents, handle name collision between HyTime-specific and user-defined identifiers and an ability to track certain activities e.g. the creation, modification and deletion of links between objects. This module also contains SGML itself.

2. *The Location Address Module*. SGML uses an unique 'identifier' and an 'identifier reference', `#ID` and `#IDREF` respectively, to refer to particular elements within a document. SGML can also create simple links to particular entities e.g. a graphic file. These identifiers and links, however, can only be used within the local scope of the document that defined them. HyTime, again, extends this by providing location address, i.e. a pointer, architectural forms. These forms include methods to address locations by name and by position using dimension(s) and axis, e.g. a substring within a string, a word in a sentence, a node within a tree etc. HyTime uses this address information to "resolve" the address and hence recover the information at that location. This can be within the local document or in another document.

3. *The Hyperlinks Module* which consists of several methods to create active references (or hyperlinks) to other documents and / or objects within those documents. This includes the general purpose *independent link* (ilink) which can have any number of link ends and ways to activate a traversal of a link, e.g. a push of a

button and the *contextual link* (clink) which always has two link ends, with one of them being the clink's own location e.g. a footnote.

4. *The Measurement Module.* This defines a way to address document objects using abstract measurable domains such as space and time.

5. *The Scheduling Module* which uses *finite co-ordinate spaces* (FCSs) to define any number of axes. These axes can represent anything that can be measured or counted e.g. time, money, temperature etc. Objects within FCSs are called *events* and these are used by the rendition module.

6. *The Rendition Module.* This module uses two constructs, the `proscope` and `modscope`. The `proscope` can be used to project certain parts of events onto another FCS e.g. show only this section of a map, whereas the `modscope` can be used to modify an event e.g. change the colour. This module calls a schedule of `proscopes` a *baton* and a schedule of `modscopes` a *wand*. A wand and then a baton can be applied to a particular event, e.g. take this picture of a crowd, change the colour and then only project a particular section of the crowd. Generally the resulting FCS is in a form that the user can see and / or hear.

2.6 Hypermedia Systems

To fully understand how the audio domain has been used in Hypermedia Systems (HSs), a brief overview of the origins of *hypertext* and *hypermedia* has to be given. This section also discusses how hypertext and hypermedia have been used in first and second generation systems, respectively. Finally this section describes how the audio domain has been supported traditionally in second generation systems and more recently in "open" hypermedia systems.

2.6.1 Hypertext and Hypermedia

The term "hypertext" has been used, over the last 30 years, to describe an extension to the traditional form of "flat" or linear text. For example, a book can be described as being linear because it is usually read from the beginning to the end. Recent developments in computer systems, however, have allowed programmers to develop new ways

in which traditional text can be viewed. Conklin [17] describes how these systems allow references to be created between different chunks of text, which can be in the same or another document. This type of text is called *nonlinear text* or *hypertext* because the path through the document can branch-off to other documents via these references.

Three of the main contributors to the area of hypertext were Vannevar Bush, Douglas Engelbart and Theodore Nelson. Conklin [17] discusses each of their hypertext systems and they were:

1. Bush's *Memex* system. In 1945 Vannevar Bush[11] predicted a rapid growth in the amount of scientific literature and the need to create a way in which this large body of information should be browsed. In his article, see Bush [18], Bush describes how the human mind works by associating related pieces of information. He applied this concept to a machine, called the *Memex*, which allowed the user to tie two relevant pieces of information, from two separate documents, together. This idea of association is credited as being the first attempt to describe hypertext.

2. Engelbart's *oN Line System* (NLS/Augment). In 1963, Engelbart described a computer system that would augment man's intellect, by allowing the user to interact with the system using special cooperative devices[12]. As a result the amount of information that a user could manipulate and understand would steadily increase, effectively "amplifying" the native intelligence of the user. The NLS system was implemented five years later at the Stanford Research Institute. It allowed users to create any number of links between elements within a document and between the documents themselves. See Engelbart and English [19] for more details.

3. Nelson's *Xanadu* System. During the development of the NLS, Ted Nelson was also developing his own ideas about augmentation. Nelson's system would only allow the storage of documents in their original format and any modifications made to these documents, e.g. a different paragraph etc. By using links between these modifications and the original documents, previous versions could be easily reconstructed. New links could easily be created between different bodies of text

[11] President Roosevelt's science advisor.
[12] One of the devices he invented was the mouse.

and therefore new pathways could be formed through the material. It was from this system of linking large bodies of text together that Ted Nelson created the term "hypertext". Nelson's book, see Nelson [20], describes his ideas in more detail.

The term hypermedia can be described as an extension to hypertext. Hypertext systems allow users to author, edit and follow links between different bodies of text. Hypermedia systems, however, are similar to hypertext systems, except that the user can use other forms of media as well. For example, the authoring of links between an audio file and a body of text.

Halasz [21] describes how Engelbart's NLS/Augment system can be called a *first generation system* because it used workstations with little or no graphics capabilities and it focused primarily on text. An overview of these systems can be found in Conklin [17]. Halasz goes on to say that in the early 1980's, second generation systems began to emerge, which used workstations with more advanced user interfaces and graphics. As a result these new systems would allow users to create references between different types of media, e.g. text, pictures etc., and hence they were called hypermedia systems. Example hypermedia systems are NoteCards [21], KMS [22] and Intermedia [23].

These second generation hypermedia systems originally used proprietary document formats to store the data. The links themselves were embedded into these documents, which made them considerably easier to transport. However this approach can cause several problems, especially with networks and distributed systems. For example, when a document is moved from one computer on the network to another, all links pointing to this document will have to be updated. Otherwise users will not be able to follow links to this document[13]. Similar problems will occur if documents are deleted and the links are not updated or removed. The use of embedded links also made it very difficult to extend these systems, to support other types of media. External programs, that were not fully integrated into these systems, would have to be used. As a result of these problems, the second generation systems were called "*closed*" hypermedia systems. Goose [24], Beitner [25] and Halasz [21] describe these systems in more detail.

[13] This is also known as the "dangling link" problem.

2.6.2 "Open" Hypermedia Systems

Goose [24] describes that in 1987, at a international hypertext conference, researchers started to express their concerns about the problems mentioned in the previous section. Several ideas were discussed, see Halasz [21]; for example new search and query mechanisms, management of dynamic information, more integration of existing applications etc. As a result of these discussions, several American research groups defined, in 1989, a reference model for hypermedia. It was called the *Dexter Hypertext Reference Model*[14], see Halasz and Schwartz [26] and it was designed to:

1. Define both formally and informally the common abstractions found in a range of existing hypertext systems, e.g. NoteCards, Intermedia, KMS etc.

2. Serve as a standard, so that the functionality and characteristics of existing hypertext (and non-hypertext) systems could be compared.

3. Serve as a template, for the development of standards. These would assist in the interoperability and interchange between different hypertext systems.

The Dexter reference model is widely regarded as being one of the most important developments in hypermedia research.

A paper by Malcolm et al. [27] describes how hypermedia could be used in industry to integrate large amounts of data from specialist tools and applications. Malcolm, however, describes how the current (second) generation of hypermedia systems were incompatible with each other, as well as the tools and applications used in industry. As a result, Malcolm et al. defined several issues that needed to be addressed, e.g the ability to access and link across different platforms (interoperability), templates for common hypermedia structures and interaction with operating systems and networks etc.

Goose [24] describes that in 1991, at another hypertext conference, Halasz revisited his original ideas, see [21], that he placed before the hypermedia community. Halasz reviewed the progress that had already been made and he also discussed the contribution made by Malcolm et al. [27]. As a result of these discussions Halasz presented several new areas of research, which focused primarily on the the development of "open" systems with independent communicating processes and the way in which large amounts

[14] It is usually referred to as Dexter.

of information could be managed and visualised on workstation screens. With the development of more "open" hypermedia systems, researchers would be able to overcome some of the problems associated with the second generation of hypermedia systems, see **Section 2.6.1**.

In [24], Goose gives a detailed overview of several hypermedia systems, that have embraced some of the concepts of "open" hypermedia. These include the World Wide Web, see **Section 2.4**, Hyper-G, Intermedia and Multicard. The following section describes why the audio domain has been neglected in hypermedia systems and how several "open" hypermedia systems have managed to overcome this.

2.6.3 Hypermedia Systems and Audio

Traditionally, the audio domain has been neglected in the development of hypermedia systems. There are several reasons for this and they include:

- *The lack of technology.* The first generation of hypermedia systems did not have the computer technology to manipulate audio data. Engelbart's NLS/Augment system used a computer with a small amount of memory and a very simple display. Over time this technology has improved, e.g. the second generation systems could manipulate pictures and text using advanced workstations. The audio domain however, is quite complex and it requires more advanced technology, which is still being developed.

- *The problem of visualising audio information.* Audio, by its very nature, can not be seen. As a result, the ability to develop an intuitive graphical user interface, that will allow users to manipulate audio information, is quite difficult.

- *The dominance of the visual sense.* Vision has always been regarded as the primary sense for normally sighted people. **Section 2.1** describes this in more detail.

- *The problems of the audio file formats.* Currently there are several file formats which can be used to store audio data. For example Microsoft's WAV format, MPEG-1 layer 3 (MP3's), Sun's AU format etc. Each format has its own advantages and disadvantages. With each format, however, the file size tends to increase as the quality of the audio recording increases. As a result, high quality recordings

can rapidly consume large amounts of disk space. The amount of memory and the processing power of a computer can also affect the size of an audio file.

These problems have caused the majority of hypermedia systems to concentrate mainly on the authoring of links between text, images and video; the *visual* domain. With the development of more powerful computers, sound cards and "open" hypermedia however, it has become possible to develop applications that can be used in conjunction with these systems, to manipulate the audio domain.

Audio tools have been developed for two hypermedia systems and they are:

1. The *SoundViewer* tool for Microcosm. Microcosm[15], see [28, 29, 30, 31], is an open hypermedia system which was primarily developed to investigate new areas of hypermedia research. Goose and Hall [32] describe it as being a set of communicating processes which supplement the facilities of the native operating system. Users interact with Microcosm using *viewers* which display different types of media, e.g. text and video, and allow users to author and follow links to and from this media. Viewers can be either specifically created for the system or existing third-party programs that have had their functionality extended; for example by using Microsoft Word macros.

   When a user performs an action in the viewer, messages are passed from this viewer to Microcosm which then dispatches them through a chain of *filters*. These filters process the messages and then decide what sort of information should be returned to the user. The filters might return a set of links, which are stored in separate link databases or *linkbases*, that the user can then follow.

   Originally Microcosm had little support for sound. When a user traversed a link to an audio file, a native application would be invoked to play this sample from the beginning to the end. Users could not create links from the application or to an area within the audio sample. As a result of this problem the SoundViewer tool was developed. This tool is discussed in more detail in **Section 4.1**.

2. The *Sound (FT) module* for MAVIS. MAVIS, the Microcosm Architecture for Video, Image and Sound, is an extension to Microcosm and it was also developed

[15] Developed by the Multimedia Research Group at the University of Southampton.

at the University of Southampton by the Multimedia Research Group. Lewis et al. [33] describe the system as being a tool that allows users to author *generic* links[16] between text and non-text based media. MAVIS achieves this by using the *content* of the non-text media as the key to navigation and retrieval of related information.

To navigate and retrieve information related to text is relatively straight forward because algorithms already exist that can match strings or sentences. With Microcosm, a user selects a piece of text within a document and then tells the system that this selection is a generic link. The system will then create a representation of this text (a key) and then store this key in a linkbase. When users want to retrieve information relating to the original selected text, this key will be used with the string matching algorithms to find other occurrences of the text.

With MAVIS, however, exact matching with non-text media is clearly impossible; the similarities between non-text media, for example images, have to be measured instead and these values can then be used to determine the closest match[17]. To achieve this, MAVIS uses *signatures* to represent the content of a selection. Several signatures are often used since there are several different ways in which non-text objects can be matched, e.g. colour, texture etc. Each description of a signature is stored in a module and users can created their own signature modules, which MAVIS can then use.

MAVIS has a prototype signature module that supports links in sound. It is called the *Sound FT* module and it uses the Fourier Transform (FT) on a selected portion of a digital sound. The resulting sample can then be used as the key for matching. Lewis et al. [33] describe how single words can be articulated into the MAVIS system, which then displays the word as a sound wave. A section of this sound wave, representing the word, can then be chosen as a source anchor of a generic link. The Sound FT module will process this selection and generate the key. This key can then be used to navigate and / or retrieve information, e.g. text, images etc., related to the original audio sample.

[16] Microcosm supports the concept of generic links – links that can be followed from any point within any document.

[17] This is known as fuzzy matching.

Both of these tools can be used to create links to and from the audio domain and they have managed to overcome some of problems mentioned at the beginning of this section, see **Section 2.6.3**.

2.7 The Audio-Linker Tool

The Audio-Linker tool was developed[18] for a demonstration to show that links could be created from audio to text and vice-versa. Originally only the Soundviewer tool and Microcosm were going to be used, see **Sections 4.1** and **2.6.3** respectively. However it was decided that a simpler audio tool, to show how the links could be created without having to use an underlying Open Hypermedia System e.g. Microcosm, should also be developed. The design and implementation of this simple tool are discussed in more detail in the following sections.

2.7.1 The Design

The requirements for the design of the tool were, again, very simple. It needed to be created quickly, due to the limited amount of time before the actual demonstration and so a *Rapid Application Development* (RAD) software tool had to be used, such as Visual Basic, Borland C++ Builder, Visual Café etc.

The graphical user interface (GUI) of the tool had to have components to display:

1. The audio sample. This component would also be used by the user to select a portion of the audio to link to and from.

2. The audio controls, which would allow the user to play, rewind, forward and stop the audio file. In other words, buttons similar to the audio controls on a tape deck or CD player.

3. The text file. This component would also be used to select a portion of the text to link to and from.

The components to display the audio sample and the audio controls would have to interact with each other, so that when the "Play" control / button is pressed, the audio sample display would change to show the current position in the file. The components

[18] Developed by Lee Oades and Neil Ridgway.

to display the audio sample and the text file would also have to interact, so that any links between the two could be displayed and followed, e.g. a highlighted piece of text would represent a link to a particular part of the audio sample and if the user clicked on this text, the link would be followed and the audio sample would be played from the relevant position.

2.7.2 The Implementation

The tool was implemented using Visual Basic because it was easy to learn and use. Other RAD tools were not really considered because the Audio-Linker application had to be developed quickly, with the resources at hand. Visual Basic provided an environment where components, called OCXs[19], can easily be made to interact with each other. By inserting several of these components into a Visual Basic *form*, which represents the main application window, the Audio-Linker tool could be rapidly developed.

Several OCX controls were used for the application and they were:

- A 'Slider' control, which was used to graphically represent the audio sample. This control consists of a horizontal bar which represents the length of the audio file and a vertical slider which was programmed to represent the current position in the audio file. By moving the slider, the user can move to different areas within the audio file. By holding down a particular keyboard key and sliding the vertical bar, a user could easily select a portion of the audio file to be linked to or from.

- A 'MMControl' control. This is a multimedia MCI[20] control that consists of a row of buttons, i.e. play, stop, pause etc., which allows the user to manipulate visual and audio media. In the Audio Linker, this control was set to play WAV sound files. This control, however, has enough functionality to support MIDI, CD Audio, CD-ROM drives and AVI (Microsoft's proprietary audio/video format) files.

- A 'RichTextBox' control, which was used to display a text file, in either plain ASCII or rich text. This control was used to create links from text to audio.

[19] OCX is an extension to Microsoft's Object Linking and Embedding (OLE) which allows users to embed objects e.g. Word documents, file dialog boxes, text boxes etc., into other documents.

[20] Media Control Interface. This is a set of high-level, device-independent commands that control audio and visual peripherals.

- A 'Timer' control, which was run whenever the play button was pressed. The timer would then poll a certain function after a few seconds, which would update the vertical bar in the 'Slider' control.

- A 'FileDialog' control, which was used to open project files.

When the application is executed, an "Open" file dialog box is displayed. This is used to load a project file, which contains the name of the WAV audio file, the name of the text file and the links, into the application. The file does not have to contain any links since they can be created in the application and then saved in the file. When the project file has been loaded, the 'MMControl' control buttons are activated and the text is displayed in the 'RichTextBox' box. By clicking on the "Play" button the audio file is played and if the current position of the audio file is in between the start and end points of an audio link, the link is followed and the relevant text is highlighted. This is shown in **Fig. 2.4**. An array in the program is used to store the links.

To test the application a simple project file was created, using paragraphs of text from a book and an audio transcript of this text. Links were then created from each paragraph of the text, to the relevant position in the audio file. When the audio file was played back, a sentence of text from each paragraph would be highlighted, when the equivalent position was reached in the audio file. If the user moved the vertical slider and hence changed the current position in the audio file, again a sentence of text would be highlighted in the relevant paragraph. These two methods are used to follow links between audio and text. By clicking the right mouse button anywhere within a paragraph, the position in the audio file and hence the slider would move to the start of the audio version of that paragraph. This method is used to follow links between text and audio.

In conclusion, this tool has shown that links can easily be created from audio to text and vice-versa, without the need of the Open Hypermedia System (OHS), such as Microcosm. The Audio-Linker was also considerably faster than Microcosm in the creation and resolution of the links. This tool, however, does not have all of the extra functionality, see **Section 2.6.3**, that Microcosm does and therefore it can not be directly compared to this OHS. With modifications and enhancements it could be used in conjunction with an OHS, so that links could be easily created from audio to other

types of media and vice-versa.

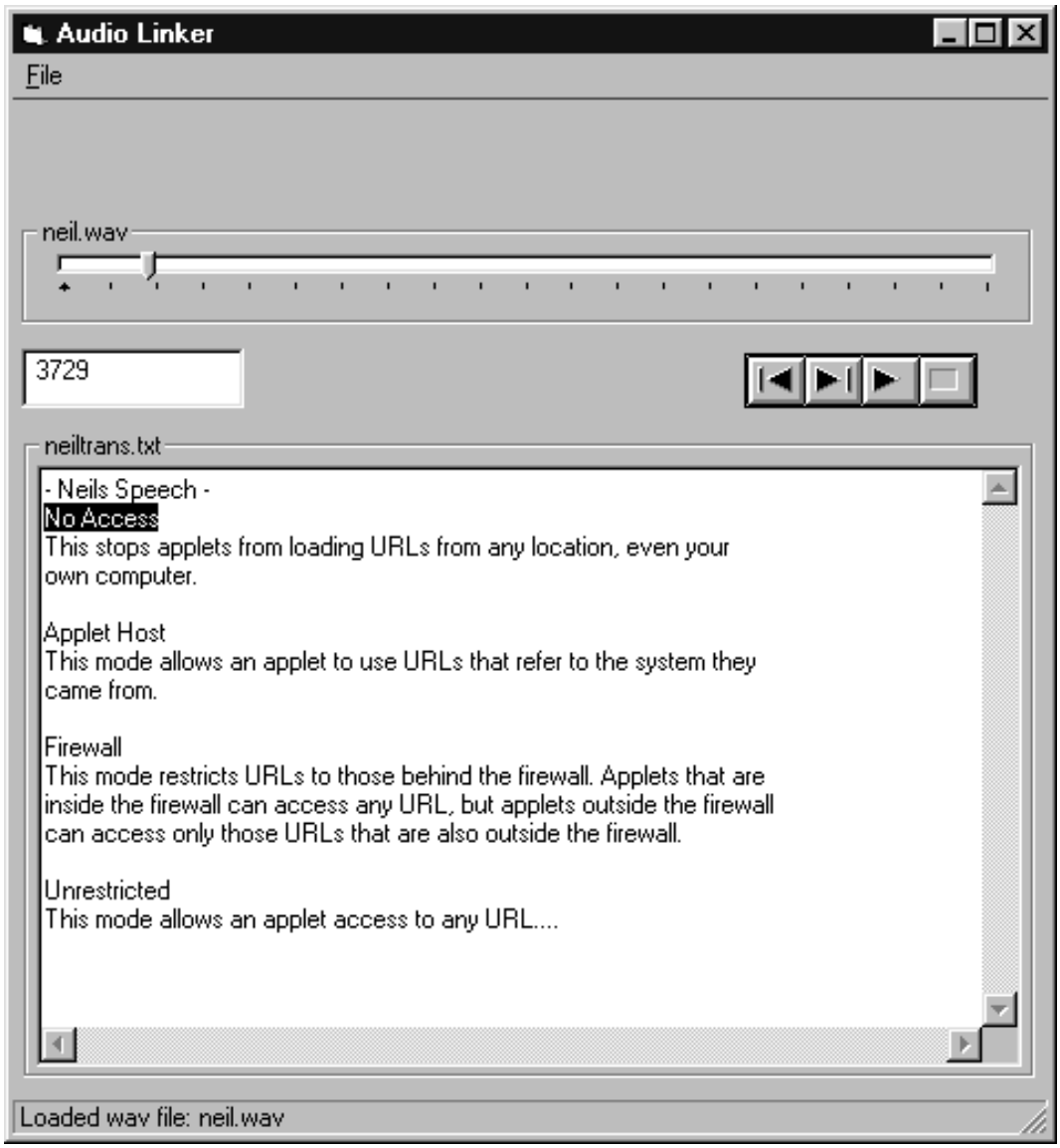

Figure 2.4: The Audio Linker Application

Chapter 3

Streaming Media Protocols

Chapter 3 describes a family of protocols that are being used to support the real-time delivery of multimedia data. It specifically concentrates on the *Real Time Streaming Protocol* which is designed to be an open standard. This chapter also gives an overview of streaming audio players and the software tools they use to create streaming multimedia presentations.

3.1 The Traditional Protocols - TCP/IP

The *Transmission Control Protocol / Internet Protocol* is the main protocol suite used with the *Internet*. Halsall [34] describes how originally *Local Area Networks* (LANs) were used to connect different computers on a local site, e.g. computers within an office or building. This allows users to share and distribute information across the local network. Large enterprises, however, usually have several sites situated in different areas of the same country and more recently, different countries as well. To connect sites that are situated within the same country, companies lease transmission lines between those sites, from the public carriers such as British Telecom. This forms a *Wide Area Network* (WAN). To connect sites that are situated in other countries, companies use different types of communication, for example satellites, optic fibres across land and / or sea etc. The networks that are formed from this type of communication are called *Internetworks* or just *Internets*.

TCP/IP consists of a suite or a layered *stack* of two core protocols and they are the *internet* and *transport* protocols. The Internet Protocol (IP) provides a number of core

functions that assist the process of internetworking across dissimilar networks. These are:

1. Addressing. There are three types of address used with the current version of the internet protocol (IPv4), *unicast*, *broadcast* and *multicast*. Unicast addressing is used when a packet of information or *datagram* is to be sent to a single destination. Broadcast addressing is used when a message is to be delivered to every host on a destination LAN. A multicast address is used to deliver a datagram to a specific set of hosts, called a multicast group. This type of addressing is called *IP Multicast*. Hosts can join a multicast group at anytime and receive the datagrams that are sent to the group.

2. Fragmentation and reassembly. If the datagrams sent by a host are larger than the packet sizes used by a particular part of the internet, the datagrams will have to be fragmented into smaller chunks so that they can be transmitted. When these smaller packets are received, they have to be reassembled into the original sized packet, so that they can be used.

3. Routing. This is used to determine which subnets, within the internet, the datagrams must travel to get to the destination host. This could involve travelling over several different LANs or WANs.

4. Error reporting. This consists of several functions that will detect errors, for example the process of reassembly could cause several packets to be discarded, and report them back to the IP in the source host. Halsall [34] describes these functions in more detail.

The transport protocols are designed to sit on "top of" the internet protocol mentioned above. They provide two modes of operation, *connection-oriented* or *connectionless*. A connection-oriented protocol creates a connection between the transmitter and the receiver before the data is actually transmitted. This is also known as a reliable transport service since the data is guaranteed to get to the destination. The *Transmission Control Protocol* (TCP) is an example of this type of service.
A connectionless protocol, as the name suggests, does not form a connection and therefore can not guarantee that the data will be delivered. This mode of operation reduces

the overhead associated with each message transfer because no network connection is established prior to the transmission. TCP/IP provides a connectionless protocol called the *User Datagram Protocol* (UDP).

When users want to transmit information over the internet, using an internet-aware application, the information is first passed to the transport protocol layer. This layer will determine the type of delivery mechanism, e.g. TCP or UDP, to use. The information is then passed to the internet protocol layer, which attaches extra information, for example the destination host address etc.

When a host receives information from the internet, the internet protocol will carry out several tests on the packets of information received. For example error reporting, which could result in the re-transmission of particular packets, reassembly of packets etc. The information is then passed to the transport protocol layer which strips off any information to do with the delivery mechanism. The resulting information can be used by the internet-aware application. A more thorough description of this can be found in Halsall [34].

3.2 The Real-time Transport Protocol (RTP)

The Real-time Transport Protocol was developed by the "Audio-Video Transport Working Group"[1] and has recently become an internet standard. RTP is described in the IETF's RFC 1889 [35] specification as being a protocol providing end-to-end delivery services, such as payload type identification, timestamping and sequence numbering, for data with real-time characteristics, e.g. interactive audio and video. It can be used over unicast or multicast networks. RTP itself however, does not provide all of the functionality required for the transport of data and therefore applications usually run it "on top" of a transport protocol such as UDP[2], which is discussed in more detail in the previous section.

RTP usually works in conjunction with another protocol called the Real Time Control Protocol (RTCP)[3], which provides minimal control over the delivery and quality of the data. It performs four main functions and these are:

[1] Formed by the Internet Engineering Task Force (IETF).
[2] RTP can also be used with other transport or underlying network protocols.
[3] It is also known as the RTP Control Protocol.

1. *Feedback Information.* This is used to check the quality of the data distribution. During an RTP session, RTCP control packets are periodically sent by each participant to all the other participants. These packets contain information such as the number of RTP packets sent, the number of packets lost etc., which the receiving application or any other third party program can use to monitor network problems. The application might then change the transmission rate of the RTP packets to help reduce any problems.

2. *Transport-level identification.* This is used to keep track of each of the participants in a session. It is also used to associate multiple data streams from a given participant in a set of related RTP sessions, e.g. the synchronisation of audio and video.

3. *Transmission Interval Control.* This ensures that the control traffic will not overwhelm network resources. Control traffic is limited to at most 5% of the overall session traffic.

4. *Minimal Session Control.* This is an optional function which can be used to convey a minimal amount of information to all session participants, e.g. to display the name of a new user joining an informal session.

When an RTP session is initiated, an application defines one network address and two ports for RTP and RTCP. If there are several media formats such as video and audio, a separate RTP session with its own RTCP packets is required for each one. Other participants can then decide which particular session and hence medium they want to receive.

Overall RTP provides a way in which real-time information can be transmitted over existing transport and underlying network protocols. With the use of a control protocol, RTCP, it provides a minimal amount of control over the delivery of the data. To ensure however, that the real-time data will be delivered on-time, if at all, RTP must be used in conjunction with other mechanisms and / or protocols that will provide a reliable service.

3.3 The Real Time Streaming Protocol (RTSP)

The *Real Time Streaming Protocol* is a proposed Internet standard which is being jointly developed by Netscape Communications, RealNetworks[4] and Columbia University. The current RFC [36] describes RTSP as being an "*application-level protocol*", which controls the delivery of streaming media with real-time properties. This media can be streamed over unicast or multicast networks. RTSP itself does not actually deliver the media data; this is handled by a separate protocol and therefore RTSP can be described as a "network remote control" to the server that is streaming the media.

The underlying protocol, that is used to control the delivery of the media, is determined by the scheme used in the RTSP *Uniform Resource Locator* (URL)[5]. The schemes that are supported on the internet are "`rtsp:`" which requires that the commands are delivered using a reliable protocol, e.g. TCP, "`rtspu:`" which identifies an unreliable protocol such as UDP and "`rtsps:`" which requires a TCP connection secured by the *Transport Layer Security* (TLS) [37] protocol. Therefore, a valid RTSP URL could be "rtspu://foo.bar.com:5150", which requests that the commands be delivered by an unreliable protocol to the server "foo.bar.com", on port 5150. A more detailed description of the URLs used in RTSP is given in Section 3.2 of the RFC [36].

RTSP is intentionally similar in syntax and operation to the latest version of the *HyperText Transfer Protocol* (HTTP/1.1) [38], which will soon become an internet standard. There are several reasons for this and they include:

- Any future extensions to HTTP/1.1 can also be added to RTSP, with little or no modification.
- RTSP can be easily parsed by standard HTTP or MIME parsers.
- It can adopt HTTP's work on web security mechanisms, caches and proxies.

Pizzi and Church [39], however, describe one of the fundamental reasons why RTSP is based upon HTTP/1.1; the previous version of HTTP (v1.0), which is currently being used by most web servers and browsers, was not designed to cope with the transmission of real-time data over the internet. What limited capabilities HTTP/1.0 has in this

[4] Formerly known as Progressive Networks.

[5] This is a formatted string that identifies via a name, location, or any other characteristic a resource, e.g. an audio file, on the internet.

area have already been exhausted. By making RTSP similar in operation and syntax to HTTP/1.1 the designers have essentially provided HTTP-level services to real-time streaming data.

Although RTSP is similar to HTTP/1.1, there are several areas in which it differs. One of these areas is RTSP's need to maintain state in almost all situations. In RTSP, as mentioned previously, the data is streamed via a separate protocol which is independent of the control channel. For example, TCP could be used for the control of the stream, whilst UDP is used for the actual delivery[6]. Thus the data will still be delivered by the media server, even if it receives no RTSP control commands. Since most servers are designed to handle more than one user at a time, the server needs to be able to maintain "session state", i.e. whether it is setting up a session, the "SETUP" state; playing a stream, the "PLAY" state etc. This will allow it to correlate RTSP requests with the relevant stream. HTTP/1.1, however, is a stateless protocol; there is no need to save the state of each client.

Another area in which HTTP/1.1 and RTSP differ is in the way the client and the server interact. With HTTP/1.1 the interaction is one-way; the client issues a request for a document and the server responds. With RTSP both the client *and* the server can issue requests.

RTSP is really more of a protocol *framework* rather than a protocol itself because it provides:

- A way to control the delivery of multiple data streams.
- A means for choosing the actual delivery channels such as UDP, multicast UDP and TCP.
- A way to choose a delivery mechanism such as RTP.

RTSP URLs, which are described earlier in this section, are used to control the delivery of the multiple data streams. The type of delivery channel and delivery mechanism, however, are usually implementation specific, e.g. coded into the implementation of the RTSP server. Early implementations of these servers support UDP, multicast UDP, TCP and RTP.

[6]RTP over UDP can also be used, although RTSP is not tied to RTP.

To create a presentation which could be a live videoconference or the simple transmission of stored data, a *presentation description* is used. This contains a common time axis and information about one or more media streams, e.g. RTSP URLs, duration, start time etc. A simple presentation description could, therefore, contain just one audio stream, whilst a more complex example could contain an audio, video and text stream running in parallel.

RealNetworks have developed the *RealMedia* architecture, that can be used to create presentations. It consists of several components of which one of them is a scripting language called the *Real Time Session Language* (RTSL). This language is used to define the content of a RealMedia presentation and therefore is used to create the presentation description.

The World Wide Web Consortium (W3C) have also developed a scripting language called the *Synchronised Multimedia Integration Language* (SMIL). Bugaj et al. [40] describe SMIL as a language that allows the integration of independent multimedia objects into a synchronised multimedia presentation. SMIL is very similar to RTSL and it can be used in the same way.

Chapter 4

The Sound Viewer Tool

This chapter covers two main areas. The first is an overview of the original Sound Viewer application and how it has been used. The second is the design and implementation of an extension to the Sound Viewer, which will allow the creation and traversal of links to and from *streaming* audio. A brief discussion of how this new tool can be used is also given.

4.1 The original Sound Viewer Tool

4.1.1 An Overview

The Sound Viewer tool was developed by Stuart Goose at the University of Southampton, to provide "a generic and meaningful visual representation of audio within a hypermedia context". A paper by Goose and Hall [32] describes the development of the Sound Viewer tool and discusses some of the unique properties of the audio domain, which is described in more detail in **Section 2.2**.

This paper discusses how audio has helped in many applications; for example audio confirmation of particular types of action possibly reduces the number of errors and non-speech sounds have been successfully used in the navigation of a screen interface for blind users. The human voice, however, is the main medium in which we communicate with each other and so it is natural to assume that it would form an integral part of multimedia and hypermedia systems. As shown in **Sections 2.3, 2.4** and **2.6** most systems support a variety of media such as text, animations, video and pictures. Audio

is supported but with many systems it is just associated with a particular event e.g. a button being clicked upon, the system shutting down etc.

The main reason why audio has not attracted as much attention as other media is due to its lack of *visual identity*. Visual media such as text, pictures and videos all have specific graphical applications to create and modify them e.g. word processors, graphics programs etc. With the Sound Viewer a suitable graphical metaphor for sound had to be found and so existing methods were reviewed. They are:

1. *The Scrollbar method.* This uses a graphical scrollbar, to represent the length of an audio file. By moving the position indicator, the user can seek to a different position in the file. Microsoft's Media Player and the simple AudioLinker tool, see **Section 2.7**, use the scrollbar approach.

2. *The Waveform method.* Several samples of the audio file are taken at a given rate and the results are then drawn, forming a waveform, onto the screen. This seems to be one of the most popular ways of representing sound. Microsoft's Sound Recorder uses this method.

3. *The Piano Roll.* This displays a single octave of a piano keyboard, drawn vertically, on the left hand side of the display. A single line representing the "play" head scrolls from left to right, at regular time intervals, over markers representing a key depression. This method is most commonly used in programs that manipulate Musical Instrument Digital Interface (MIDI) files e.g. Cakewalk's Pro Audio.

4. *The Manuscript / "Score" method.* This represents audio using a musical score, e.g. five horizontal parallel lines (the stave) represent pitch and musical notes drawn on this stave produce the music. Again this is used with MIDI files.

5. *No visualisation (Sound engineers).* The majority of sound editing by studio engineers is performed manually without any visual tools. The equipment used provides fine control over the start and stop positions within the audio recording and playback facilities for further refinements of these positions. Time and position indicators are used for noting and re-locating specific sequences.

Different sound formats have different ways in which they can be rendered on a display. For example MIDI can be expressed well using the manuscript and piano roll method,

whilst it can not be displayed using waveforms. Also MIDI can not be used for speech. Microsoft's proprietary WAV format is usually displayed using waveforms and can be used for both music and speech. However, the waveform itself is of little use to the average user who can not relate the sound to its corresponding waveform. The scrollbar method is really just a means to move within the audio file itself and therefore, it can be used for all of the sound formats e.g. WAV, CD-Audio, MIDI etc.

Each of these methods have their own advantages and disadvantages and the Sound Viewer tool was designed to represent WAV, MIDI and CD-Audio in an uniform way. It would be very confusing if the tool displayed a waveform for a WAV file and something completely different for a MIDI file. Therefore it was decided that more priority would be given to the visual authoring of the anchors, rather than how the audio was displayed. Goose and Hall [32] describe several other objectives required for the manipulation of audio by a hypermedia author / user and they are:

- To have full control over the audio device.
- To discern the current position and duration of the sound sequence.
- To have the ability to select a portion of the sequence, playback and refine that selection in preparation for creating an anchor.
- To identify any links present in the sequence.
- To traverse links to and from the sequence in an intuitive manner.

The host hypermedia system that manages the link information created to and from the Sound Viewer, is called *Microcosm*. This "open" system is discussed in more detail in **Section 2.6.3**.

4.1.2 The Interface

The user interface of the Sound Viewer consists of several components and these are the *audio controls*, the *windows* that represent the audio and are used to view the links, two *position counters* and two *selection indicators*. Each of these components are shown in **Fig. 4.1**.

The audio controls are similar to the ones used with cassette decks and compact disc machines and consist of buttons with the labels "Play", "Pause" and "Repeat". There

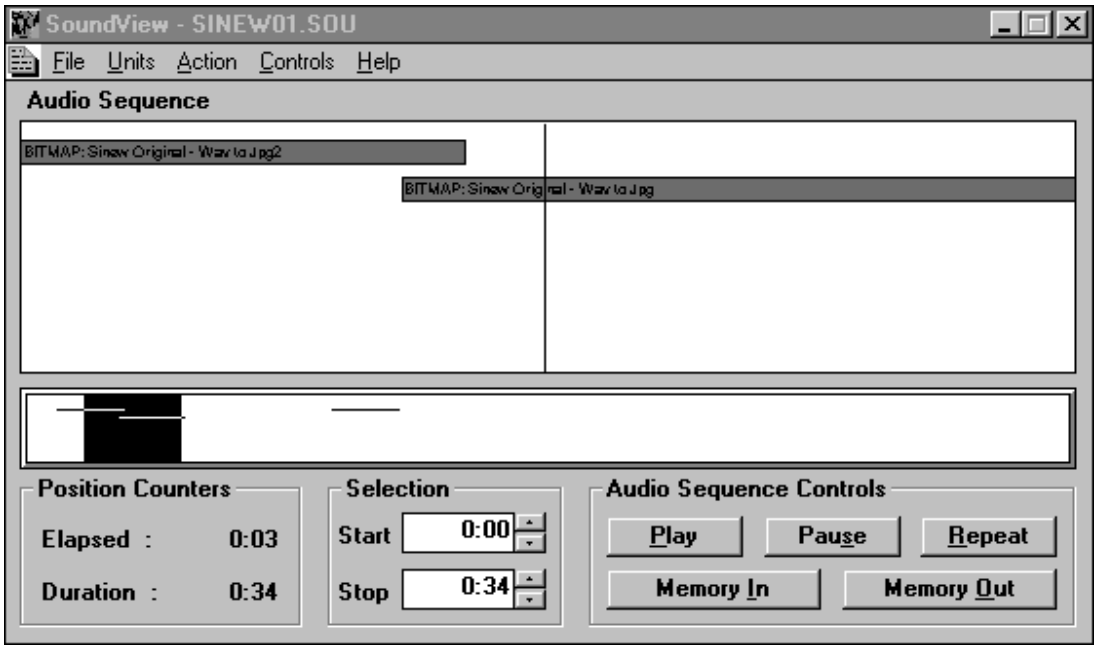

Figure 4.1: The Sound Viewer Interface.

are also two extra buttons; the "Memory In" which stores the current position / time within the file and the "Memory Out" button which, if pressed, displays a list box of all the stored positions. The user can double-click on any of these, to move to that position in the file.

The Sound Viewer has two white windows which span the width of the main application window. The larger of the two windows is called the *detail* window and the smaller, the *overview* window. The overview window represents the length of the audio file and within this window is a highlighted rectangle, which represents the exploded view seen in the detail window above. Users can change the width of this rectangle, so that the smaller the rectangle the higher the zoom factor and vice versa. The middle of this rectangle is represented in the detail window as a thin vertical line.

When an audio file is played, the highlighted rectangle moves from left to right, which gives a visual clue to the current position in the audio sequence. The vertical line also moves to the middle of the detail window and when the end of the audio sample has been reached, it will scroll off to the right. If the highlighted rectangle passes over any links, the links scroll horizontally across the detail window as well. The links are displayed in the overview window as horizontal lines and in the detail window as shaded

rectangles with simple text descriptions of the link. The overview window is really just an enhanced scrollbar and by moving the rectangle or the *position indicator* the user can seek to a position within the file.

There are two position counters and two selection indicators underneath these windows. The two counters represent the duration of the file and the elapsed time since the start of the track. The indicators are used to select a portion of the sound file which is conventionally highlighted in black. This selection can then be used to create an anchor.

When the Sound Viewer is executed, outside of the Microcosm environment, all of the components mentioned above are shown, apart from the detail window. This is displayed when an audio file is loaded. The position counters and selection indicators are all set to zero and the audio sequence controls are dimmed, since they can not be used at this stage. To load an audio file, the Sound Viewer uses a *project* file which has a ".SOU" extension. This file contains the type of audio format e.g. MIDI, WAV etc., the audio sample's filename, the start and stop time of the sample and a simple description of the file. There is also an option to automatically play the file when it is loaded. After a project file has been parsed and the audio file has been loaded, the detail window is shown and the position counters are updated. The audio sequence controls are also updated e.g. the "Play" button is undimmed so that the file can be played.
Within the Microcosm environment, the application is usually executed when the user follows a link to a ".SOU" file or when the user explicitly uses Microcosm to open an ".SOU" file. In both cases the Sound Viewer automatically loads the project file.

To create and traverse links to and from the Sound Viewer tool, the Microcosm system which manages all of this link information, must be running in the background. With this system running, a user must follow several steps to create a link and these are:

1. Using the selection indicators, a user selects a portion of an audio file to be the start point of a link.

2. The user selects the "Start Link" option from the "Action" menu. This starts the creation of a link and a "Start Link" dialog box is displayed.

3. The user then selects "End Link" from the "Action" menu of a Microcosm-aware application or Microcosm's own universal viewer. This ends the link and the "End

Link" dialog box is displayed.

4. The user then clicks on the "Complete..." button, in either of the two dialog boxes. This brings up the "Linker" window which allows the user to type in a brief text description of the link and choose the type of link to be created. At the moment the Sound Viewer only supports specific links (from one point to another) because intensive sound processing would be required for the generation of generic or local links.

5. By clicking on the "Ok" button, in the "Linker" window, the link is forged and a message is sent by Microcosm to the Sound Viewer to show the link information in its windows, e.g. a horizontal line in the overview window and a shaded rectangle, with the text description of the link, in the detail window.

To follow / traverse a link, from the Sound Viewer, a user can either double-click on the shaded rectangle (the link) in the detail window or press the "play" button. By pressing this button, the highlighted rectangle in the overview window will move from left to right, which will cause the shaded rectangles (the links) in the detail window to do the same. As these links pass under the vertical line, in the detail window, they are automatically traversed. This is shown in **Fig. 4.3**. In this picture an audio file is being played and it contains two links; one to a window containing a picture and the other to a text window. These links are immediately followed as the vertical line passes over them.

4.1.3 The Implementation

The original Sound Viewer was implemented using Microsoft's Visual C++ for Windows v1.52. The graphical interface to the viewer was created using one of the components of this application; the *Resource Editor*. This editor allows the user to create dialog boxes, icons, fonts, menus and other resources, which can then be used in the main application. For the Sound Viewer, this editor was used to create a dialog box with a menu, two text boxes for the counters, two list boxes for the selection indicators and five buttons for the audio controls. The detail and overview window were created using specific functions in the Visual C++ language. The dialog box was then used as the graphical user interface (GUI) to the program, see **Fig. 4.1**.

Goose and Hall [32] describe in their paper how the Sound Viewer tool was divided into a number of modules. For example one of the modules would handle the graphics, whilst another would handle the audio (see Figure 6 in [32]). This helped to reduce the complexity of the programming task.

This paper also describes how the audio module consists of two layers. The first layer is a suite of audio device independent functions, that separate the user from the lower layer audio device controls. The second layer consists of a suite of functions for each of the audio formats supported by the Sound Viewer. These are specific audio functions that are called by the first layer and rely heavily on Microsoft's Media Control Interface (MCI). MCI provides several functions that allow users to control different types of media, e.g. opening, playing, stopping and closing a WAV audio or AVI video file.

A simple visual representation of these two layers is shown in **Fig. 4.2**. Each of the "*?*" in this diagram can be replaced with a specific audio control command, e.g. "Open", "Close", "Play", "Stop" etc. Therefore if a user clicks on the "Play" button, the "PlayAudioDevice" function is called. This function will then call "GetAudioDeviceType", which will return the format of the audio file. If the file is a WAV file, the "PlayWAV" function is called, which in turn calls the relevant MCI function.

The modular design of the Sound Viewer and the creation of an abstract layer, between the user interface and the audio controls (MCI), ensures that any new and emerging audio formats can be easily supported.

4.1.4 The original Sound Viewer case study

To test the original Sound Viewer tool, a simple demonstration had to be developed. It was decided that certain items from the Churchill archives, stored at the University of Southampton, could be used. The archives contain several movie clips and text transcripts of Winston Churchill's Sinews of Peace speech at Westminster College. For the demonstration it was decided that the audio from the movie clips would be used, in conjunction with pictures and the text transcript of the speech.

To create the demonstration, links were created between the relevant pieces of the media, e.g. between the text transcript and the relevant portion of the audio sample, between the audio and JPEG pictures etc. These links were created using Microcosm, see **Sections 2.6.3** and **4.1.2**.

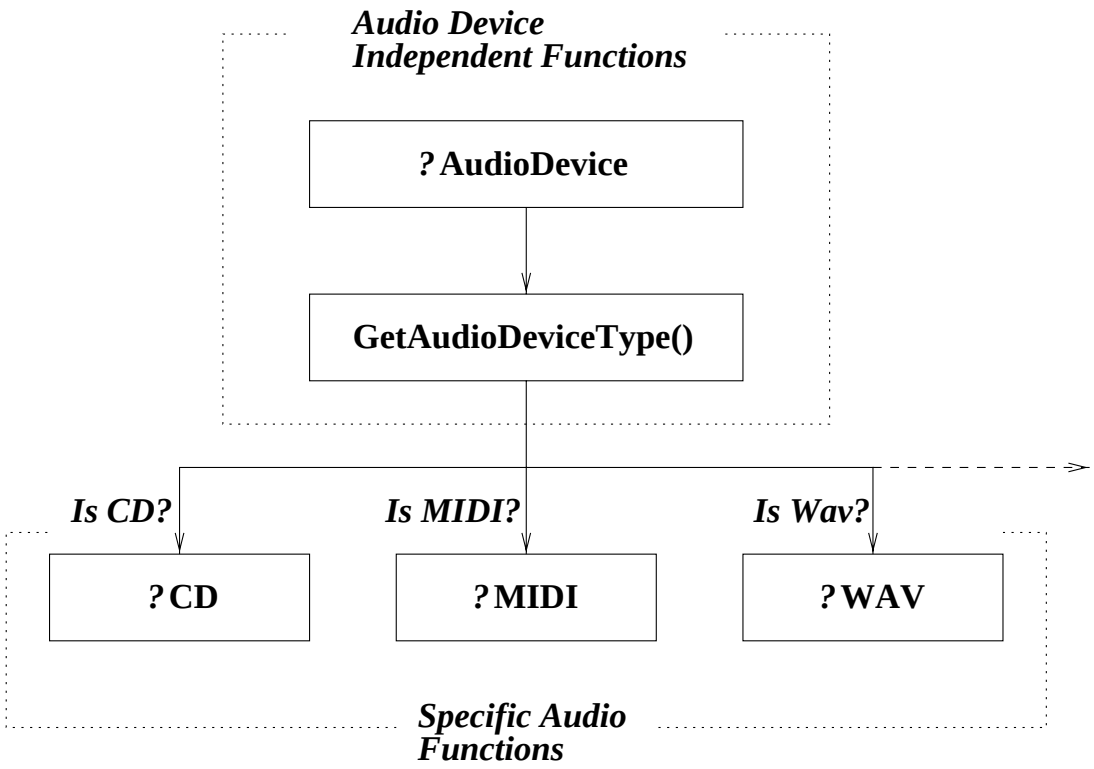

Figure 4.2: The Audio Device Layers.

Once the links had been created between each of the media objects, the demonstration could then be used. A user could bring up the text transcript of the speech and click on a highlighted link. This would cause the Sound Viewer tool to be displayed and the user could then play the audio sample. **Fig. 4.3** shows a small sample of a demonstration.

In this screen-shot, the Sound Viewer tool has been activated. Within the viewer's detail window there are two links (represented as two rectangles). When the user presses the "Play" button, the audio sample is played and the highlighted rectangle in the overview window moves from left to right. This causes the links to move and pass under the thin black line in the detail window above. As the links pass under this line they are automatically followed and in the diagram, **Fig. 4.3**, one of the links displays a picture of Winston Churchill and the other, the text transcript of his speech.

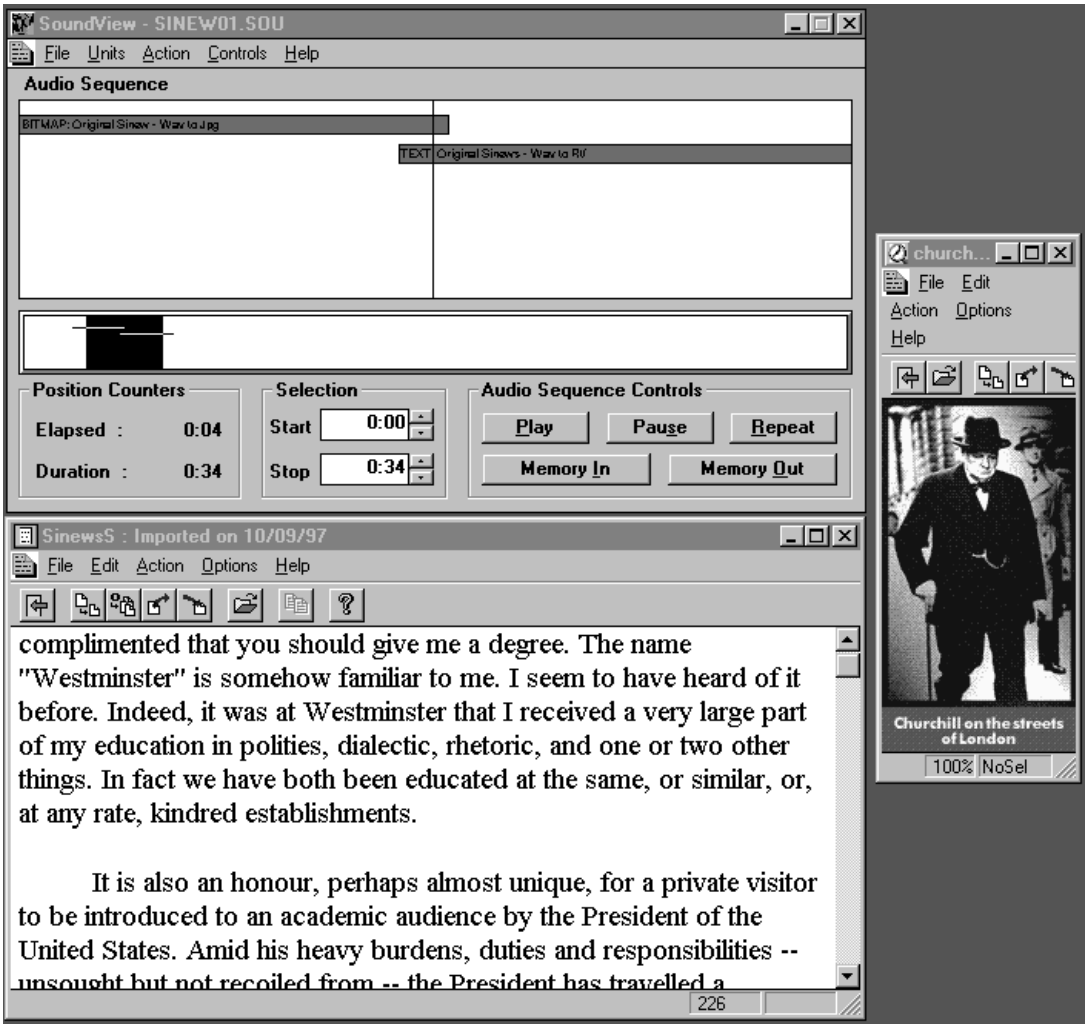

Figure 4.3: The original Sound Viewer demonstration.

4.2 The streaming Sound Viewer

4.2.1 An Overview

Over the last few years the World Wide Web, see **Section 2.4**, has steadily grown from being a simple hypertext system to a more sophisticated *hypermedia* system, that supports moving pictures, audio and video. Traditionally when a user clicks on a link to a page that contains a sound and / or a video sequence, the user had to wait until the entire media file had been downloaded, before it could be played. If the network connection was slow and / or the file was very large, this could take a considerable amount of time. Therefore software and hardware manufacturers started to develop

protocols that could *stream* the media over the networks. These are discussed in more detail in **Chapter 3**.

If a user clicks on a link to a streaming audio file, a request is sent to a streaming audio server which splits the appropriate file into smaller *packets*. These packets are then sent to the client (user's computer) using the new protocols. A client-side streaming media application then buffers the incoming packets until a certain amount has been received. It can then play the contents of the buffer. If the network connection is good, there is no perceivable delay in the delivery of the data. Otherwise there is a slight delay as the application waits for the packets.

The Soundviewer tool was developed as a hypermedia application to visually create and traverse links to and from the audio domain. Audio files can be loaded into this tool and then the Microcosm link creation mechanism can be used to create the links. These audio files, however, can be quite large e.g. 31.5 seconds of stereo sound can generate a 347.7 kilobyte WAV file. Audio and video sequences, which contain far more information, can consume copious amounts of hard disk space. With the streaming audio techniques mentioned previously in this section, it is possible to store this information on a separate streaming media server. Users can then access the audio by using a streaming media protocol, which would save valuable amounts of disk space.

To implement this concept, the functionality of the Soundviewer was extended so that streaming audio could be supported. This was achieved by changing the tool from a simple application to a streaming audio client, that could communicate with the media server. Several protocols exist that provide suitable methods for a media server to communicate with a client and vice-versa. **Chapter 3** discusses two of these protocols and they are the Real-time Transport Protocol (RTP) and the Real Time Streaming Protocol (RTSP). RTP is an internet standard whilst RTSP is a proposed internet standard. Before it can become a standard, however, it has to go through several refinements and at least two basic implementations of the client and the server have to be written. This is a requirement of the Internet Engineering Task Force (IETF), which is a part of the internet standards committee. The RTSP drafts and implementations are publicly available and allow prospective users to comment on any area of the protocol. This ensures that the protocol remains an "open" standard.

Since RTSP provides most of the functionality required to create a streaming version

of the Sound Viewer and an initial implementation of the protocol is already available for public use, it was decided that this protocol would be used with the Sound Viewer. The following sections describe the design and implementation of this streaming audio tool.

4.2.2 The Design

As described in the previous section, **Section 4.2.1**, IETF requires at least two implementations of a proposed internet standard before it can become an actual standard. As a result the developers of RTSP, see **Section 3.3**, have created a simple client and a server, that will stream WAV audio files over the internet. This code is available for Microsoft Windows (both 95 and NT) and UNIX machines. By combining a modified version of this client with the original Sound Viewer, a streaming Sound Viewer client was created. The RTSP server, however, was not modified since its functionality does not need to be changed.

To integrate the RTSP client into the Sound Viewer, the viewer's audio module was extended. The audio module consists of two layers, see **Section 4.1.3**; an abstract audio device independent layer and a suite of lower-layer functions for each of the supported audio formats. For RTSP, a new set of these functions were created. A small modification was also made to the abstract layer so that it could detect a new audio device type, e.g. RTSP. A simple diagram showing how RTSP is integrated into these layers is shown in **Fig. 4.4**.

Each of the "*?*" in this diagram can be replaced with a specific audio control command, such as "Open", "Play", "Stop" etc. If RTSP is the audio format being used and the user clicks on the "Play" button, the "PlayRTSP" function will be called.

One of the fundamental design decisions, for the streaming Sound Viewer, was that the interface would look exactly the same as the original tool. As far as the end user is concerned the viewer looks the same and can be used in exactly the same way. A few changes were made to the project file, which is described in **Section 4.1.2**, but apart from this, however, all of the underlying network connections to the RTSP server and the actual delivery of the audio data will be transparent. For example a fast network connection gave the impression that the sample being streamed from the server was actually on the local machine. Network congestion, however, can cause the viewer to

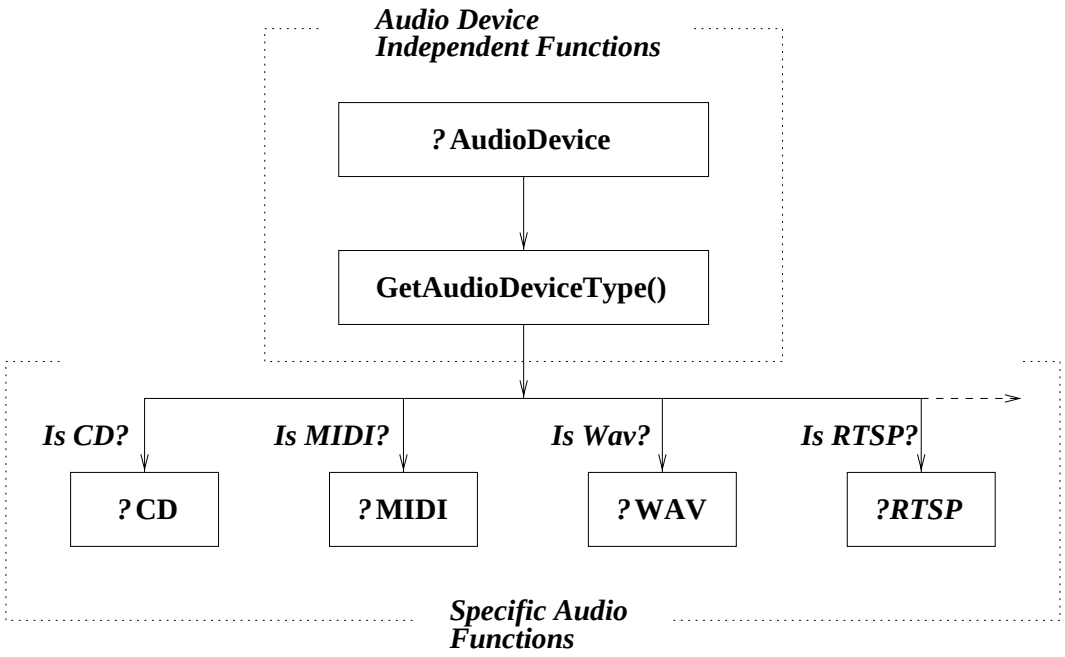

Figure 4.4: The Audio Device Layers with RTSP.

pause playback until enough data had been received.

With the original Sound Viewer, a project file is used to store information about the audio sample, e.g. the filename, start and stop times etc. This file is then loaded into the tool. The contents of this file for the streaming version of the viewer, however, had to be changed. For example:

- A new audio format type would need to be supported, e.g. RTSP. The original viewer only recognised three types of audio in the project file: CD, MIDI and WAV.
- The location of the audio file would have to be changed from a local path to an RTSP URL. This URL would be used to locate the media server.
- The start and stop times of the audio sample would have to be removed because the file is stored on a separate media server. The length of the file would only be known when this information is retrieved from the server.

When this new project file is loaded into the streaming viewer, the URL (which includes the filename of the audio sample on the server) is read in and a connection is then formed

between the viewer and the media server. At this stage the Sound Viewer has effectively become an RTSP client. **Fig. 4.5** shows the interaction between the streaming Sound Viewer and the media server, when the viewer requests a connection to be formed.

The connection that is formed between the viewer and the server is called the *control channel*. The type of channel used will depend on the RTSP URL, see **Section 3.3**. The viewer uses this channel to send commands to the server. Initially a "get" command is sent, with the filename of the audio sample, to the server. **Figures 4.6** and **4.7** show the interaction between the viewer and the server, when the RTSP "get" command is issued.

The media server processes this command and returns information about that audio file, e.g. the file length in bytes, the duration in milliseconds etc., to the viewer. This information is used to update the duration counter and initialise certain components within the Sound Viewer tool.

When the "Play" button is pressed in the viewer, the "p" command is sent to the server. This command can also contain a range to play, which is described in more detail in the RTSP standards track, see [36]. The server will process this command, divide the audio sample into packets and then stream these to the viewer. The delivery mechanism for this audio data is separate from the control channel and it might be reliable or unreliable[1], e.g. TCP or UDP respectively. The developers initial implementation of the RTSP server uses UDP and therefore, there is no guarantee that the packets will be delivered to the viewer. **Fig. 4.8** shows the interaction between the streaming Sound Viewer and the media server, when the play command is sent to the server.

With the original Sound Viewer two events occur when the audio file is played and they are:

1. The highlighted rectangle in the overview window moves from left to right. This graphically represents the current position in the audio file.

2. The "Elapsed" position counter also changes to display the current position. This is usually in milliseconds or minutes and seconds, depending on the user's preferences.

Both of these components use a Media Control Interface (MCI) function to obtain this

[1]This is usually implementation specific.

position. MCI, however, will only work with files stored on the local machine. With the streaming Sound Viewer, the audio sample is stored on a media server which is another machine connected to the internet. Therefore this function can not be used with the streaming viewer. To overcome this problem a *virtual time unit* (VTU) was created. This calculates the current position within the file, based on the number of packets received from the RTSP server, the length of the file in bytes and the duration of the file in milliseconds. The VTU was then used to move the highlighted rectangle and change the "Elapsed" position counter.

As mentioned previously, MCI can only be used with media files stored on the local machine. Therefore MCI will not work with RTSP, since the audio files are stored on other machines (media servers). The developers of the RTSP client, however, overcame this problem by using Microsoft's lower-level audio specific functions. These functions were incorporated into the lower-layer of the Sound Viewer's audio module. There was no need to modify the MCI controls for the other supported audio formats, since their functionality has not changed.

4.2.3 The Implementation and case study

The main problem that was encountered, in the implementation of the streaming Sound Viewer tool, was that the original viewer was designed for a 16 bit open hypermedia system (Microcosm), see **Sections 2.6.3** and **4.1.1**. Therefore the original Sound Viewer code had to be compiled using a 16 bit compiler, e.g. Microsoft Visual C++ for Windows 1.52. Both the RTSP client and server, however, were designed to be *32 bit* applications; requiring a 32 bit compiler, for example Microsoft Visual C++ v4 for Windows 95 or NT. As a result, the streaming viewer could not be created by simply "inserting" the relevant RTSP client code into the original Sound Viewer code. The resulting program would contain 16 and 32 bit code, which would not compile.

To overcome this problem, the RTSP client was converted into a 16 bit application. This new code was then integrated into the original Sound Viewer code to form the streaming viewer application. For example the functions in the RTSP client that handle the streaming audio were inserted into the lower layer of the audio module, see **Sections 4.1.3**, **4.2.2** and **Fig. 4.2**.

Another possible solution would of been to convert the Sound Viewer into a 32 bit

application. However, there is no 32 bit implementation of Microcosm and so the functionality required to create, modify and follow links would have been lost; the tool would have effectively become a standalone application that could play local and streaming audio files.

Overall the process of integrating the RTSP client into the original Sound Viewer tool was relatively straight-forward. New code had to be created to handle the virtual time unit (VTU), see **Section 4.2.2** and the interface between the RTSP client and the Sound Viewer's graphical user interface (GUI). This new code, to handle the interface, was designed to ensure that all of the network connections to the RTSP server were transparent to the user.

To test the streaming Sound Viewer, a demonstration similar to the original case study, see **Section 4.1.4**, was created. Again the Churchill archives, at the University of Southampton, were used. Links were created between the audio samples (which are now stored on a separate media server) and the relevant pieces of the archive. This is shown in the diagram **Fig. 4.9**.

In this screenshot, when the "Play" button has been pressed, a command is sent to the media server to stream the audio to the Sound Viewer. When the Sound Viewer receives enough information, the audio is played and the virtual time unit is updated. As the VTU is updated, the highlighted rectangle in the overview window moves from left to right, which causes the links to pass under the thin black line in the detail window. This results in the links being automatically followed.

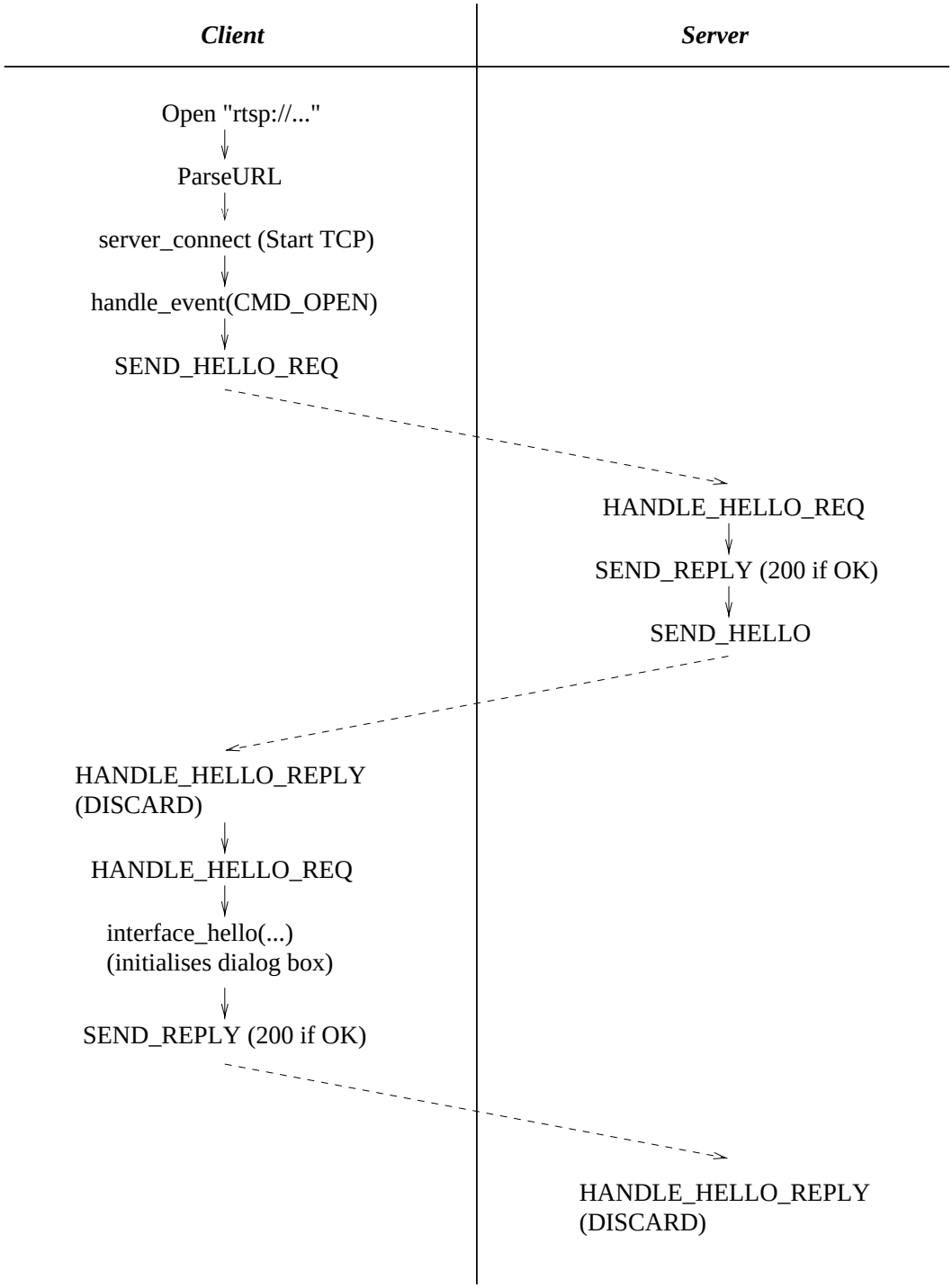

Figure 4.5: Client / Server interaction for the RTSP “open” function

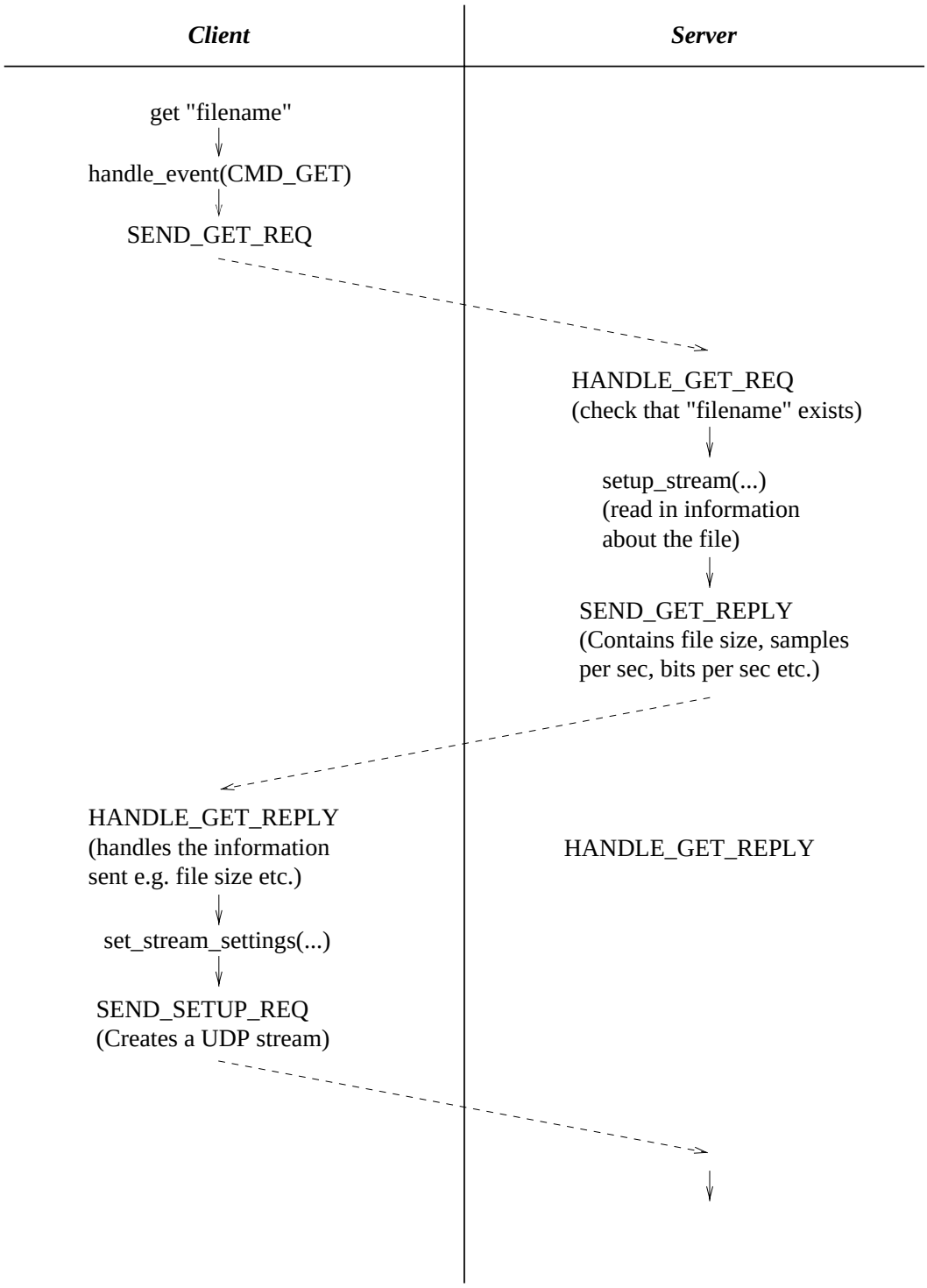

Figure 4.6: Client / Server interaction for the RTSP "get" function

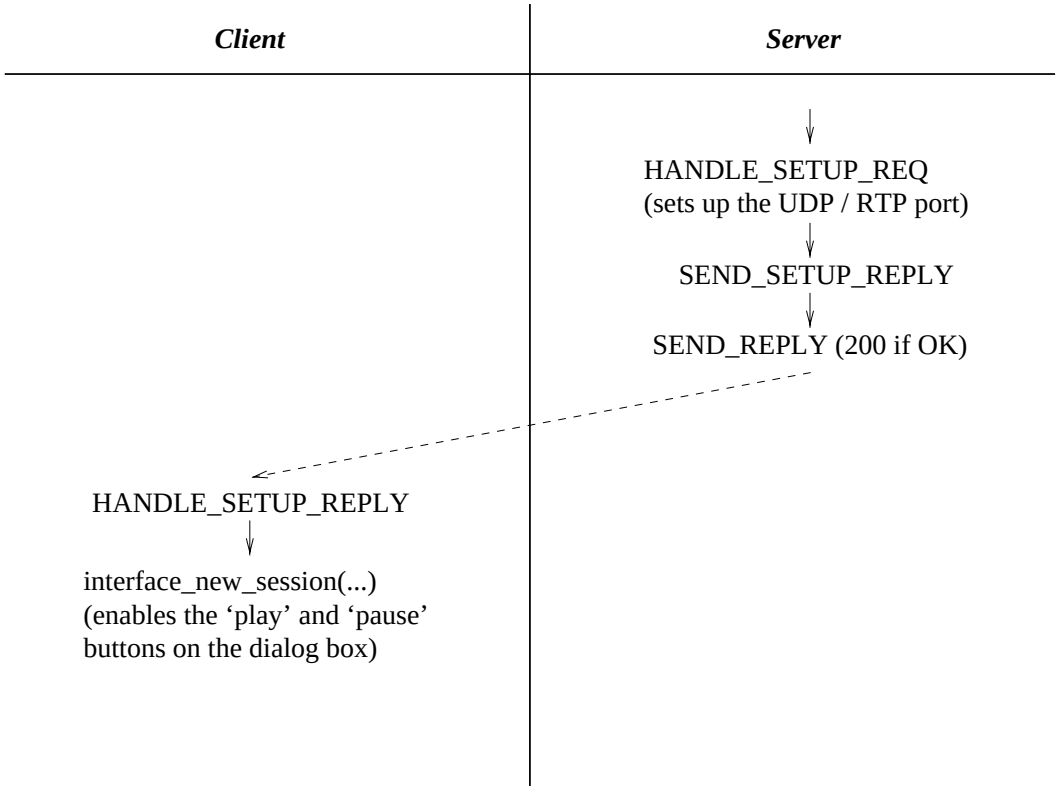

Figure 4.7: Client / Server interaction for the RTSP "get" function (cont'd)

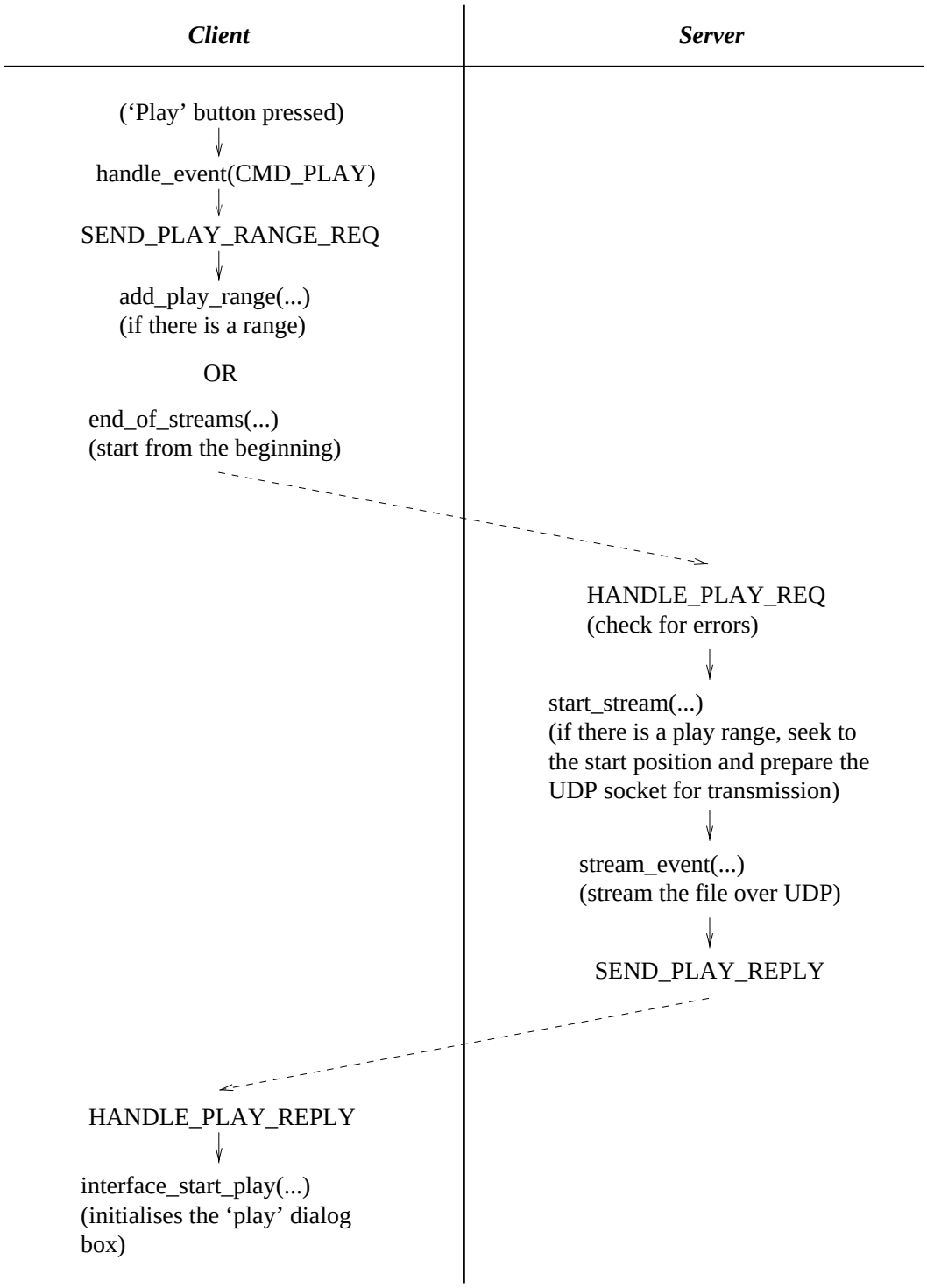

Figure 4.8: Client / Server interaction for the RTSP "play" function

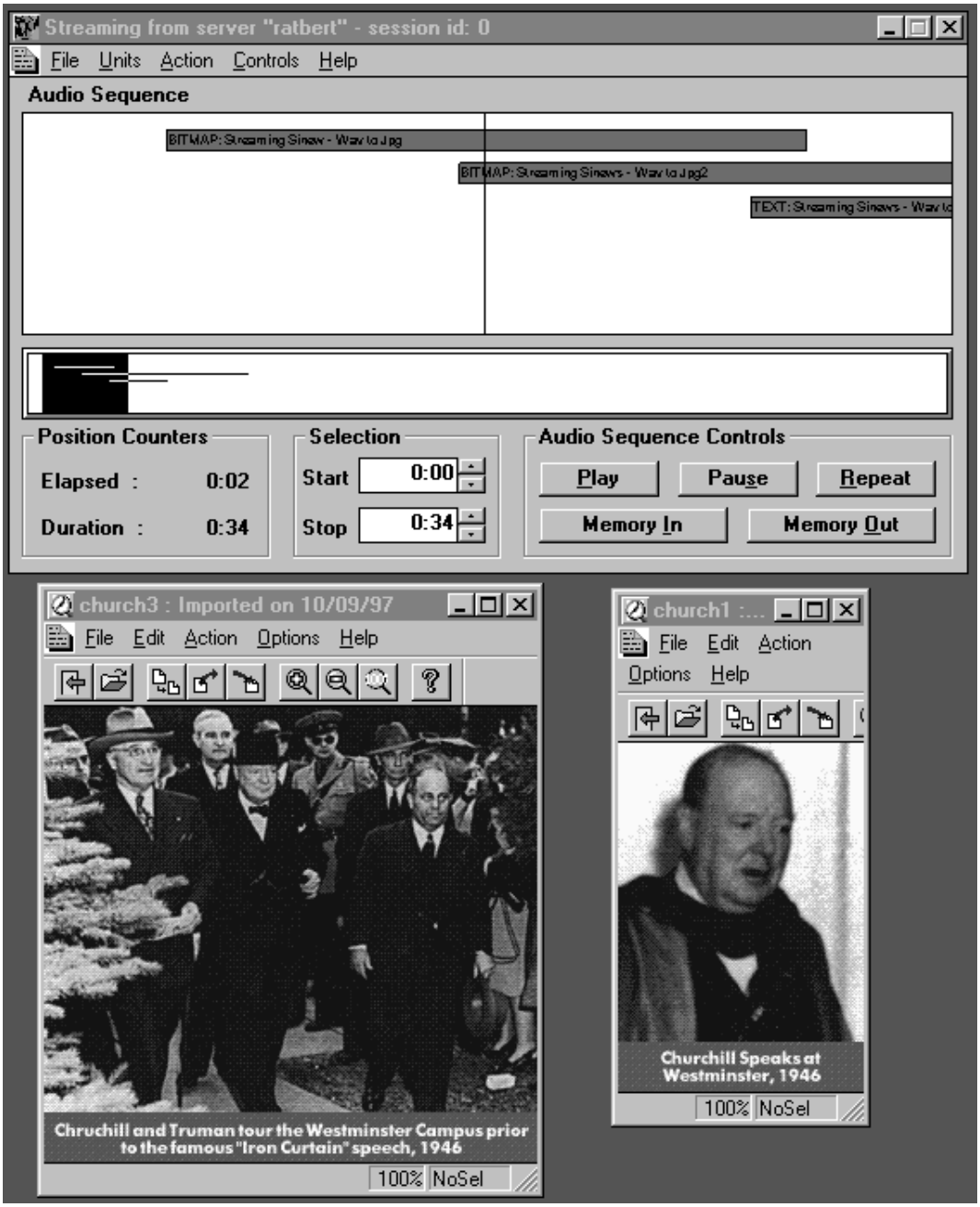

Figure 4.9: The streaming Sound Viewer.

Chapter 5

Conclusion

This report has discussed how audio has been used in a variety of multimedia, hypermedia and "open" hypermedia systems. With most of these systems however, audio is used as just another part of a presentation or as a side-effect when the user clicks on a button / link. This is not synonymous to the way sound is used in everyday situations. Humans generally use a mixture of the five senses to interact with the outside world, whilst the majority of the systems mentioned above use the sense of sight as the primary means to convey information. This is mainly due to the complexity of the other senses.

The aim of this research was to develop a tool that allows users to create links to *and from* streaming audio. There are several existing standards, MHEG and HyTime, that could have been used to do this. MHEG and HyTime however, are quite complex and have primarily been designed to help create easy-to-use client/server systems and well-structured documents, respectively. The audio domain has been supported by earlier versions of HyTime (SMDL), however these standards have focused specifically on the visual domain.

Streaming audio is being used quite extensively on the World Wide Web (WWW). On older web browsers, a user would click on an "audio" link to play an audio file. The browser would then download the entire file to the user's PC and then activate a local sound tool to play the sample. This process of downloading the file could take a considerable amount of time especially if the audio file was large and / or the network connection was poor. Streaming the audio can take considerably less time because the file does not have to be downloaded first. The WWW however, is known as a

"closed" hypermedia system because all of the link information is embedded within the documents. Also, there are no mechanisms within the HyperText Markup Language (HTML), the language used to markup documents for the WWW, to create links from within an audio file or stream.

Multimedia authoring tools and the WWW can be used to create links to audio. These systems however, can not create links from within an audio file or stream. Therefore to overcome this problem existing hypermedia systems were examined. Originally these systems only allowed the creation of links to and from text. Over time, as technology improved, new systems were developed that could support different types of media as well as text. These systems however, used proprietary document formats with embedded links, which made them very difficult to extend for different media types. At this time, these systems had limited support for audio.

In 1987 researchers realised that they needed to develop more "open" systems, which could share data and store link information separately from the actual content. Several "open" hypermedia systems have been developed and they include Intermedia, Microcosm and Hyper-G. The majority of these systems have, again, used the sense of sight as the primary means to convey information.

In 1994, however, an audio tool was developed for the Microcosm system. It is called the SoundViewer tool and it allows users to easily create links, from within an audio file, to different types of media. Microcosm also allows the creation of links to this tool and hence the audio, as well. The SoundViewer, however, only allows users to load files that are stored on the local machine. To create links to and from *streaming* audio, the tool had to be changed to a streaming audio client, so that it could communicate with a streaming audio server.

To create this new streaming SoundViewer client, the functionality of the tool had to be enhanced using a new streaming protocol; the Real Time Streaming Protocol (RTSP). RTSP has just recently become a proposed standard and it is described as being a "network remote control", e.g. it allows users to play, pause, stop or move to any position within the audio stream. This protocol is used by the tool to communicate with the server and buffer any incoming audio streams. The streaming SoundViewer can still be used with local audio files.

To conclude, the sense of sight has always been regarded as the primary means to

convey information, for normally sighted people. As a result the other senses have not really been used; although this is slowly changing as technology improves. This report has shown that it is possible to use the audio domain and recent developments in audio technology, in ways that have traditionally been used for the visual domain.

5.1 Future Work

This section describes several further areas of research and it also gives a brief outline of the schedule required to cover the main aspects of this work. Several areas are going to be investigated and they are:

1. Content Based Navigation (CBN) using speech recognition software and RTSP as the delivery mechanism.

2. The next generation of internet protocol, IPv6. The current protocol (IPv4) was not originally designed to handle streaming media. IPv6 however, has several new mechanisms that can handle this type of media.

3. Distributed linkbases. The Microcosm system stores link information in linkbases on the local machine. The streaming SoundViewer could be enhanced if distributed linkbases were also supported; possibly by using the Distributed Link Server (DLS).

The schedule for this work is given below.

- **March - April**
  - Research into speech recognition software and the speech API's.
  - Investigation into how the streaming audio can be fed into a speech recognition engine.
  - Familiarization of speech software development kits (SDKs).
- **May - July**
  - Research into the IPv6 "flow" control mechanism and incorporate IPv6 into the RTSP code.
  - General RTSP improvements.

- Incorporating RTSP into the DLS.
- Testing and evaluation.

- **August - October**
  - Design of speech recognition tool to be used with RTSP and the DLS.
  - Testing and evaluation.

- **November - December**
  - Write up of the thesis.

Bibliography

[1] A. Dix, J. Finlay, G. Abowd and R. Beale, *Human-Computer Interaction*. Prentice Hall Europe, 2nd ed., 1998.

[2] Brian C. J. Moore, *An Introduction to the Psychology of Hearing*. Academic Press, 4th ed., 1997.

[3] P. Georghiades and G. Jacobs, "Mixed Media," *Personal Computer World*, vol. 20, pp. 225–242, November 1997.

[4] T. J. Berners-Lee, R. Cailliau, J. F. Groff and B. Pollerman, "World Wide Web: An Information Infrastructure for High-Energy Physics," in *Proceedings of the Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics*, (La Londe-les-Maures, France), CERN, World Scientific, January 1992.

[5] Arturo A. Rodriguez, Martin Fisher and Brian Markey, "Scripting Languages Emerge in Standards Bodies," *IEEE Multimedia*, pp. 88–92, Winter 1995.

[6] Thomas Meyer-Boudnik and Wolfgang Effelsberg, "MHEG Explained," *IEEE Multimedia*, pp. 26–38, Spring 1995.

[7] Lloyd Rutledge, John Buford and Roger Price, "Mobile Objects and the Hyoctane Distributed Hyperdocument Server," *Computers & Graphics*, vol. 20, no. 5, pp. 633–639, 1996.

[8] Robert Joseph, PhD and Jörgen Rosengren, MSc, "MHEG-5: An Overview," *http://www.fokus.gmd.de/ovma/mug/archives/documents/mheg-reader/... rd1206.html*, December 1995.

[9] Lars Geyer, Michael Baentsch, Lothar Baum, Georg Molter, Steffen Rothkugel, Peter Sturm, "MHEG in Java - Integrating a Multimedia Standard into the Web," *http://thunder.informatik.uni-kl.de:8080/MHEG5Engine/*, 1996.

[10] L. A. Carr, D. W. Barron, H. C. Davis and W. Hall, "Why use HyTime?," *Electronic Publishing*, vol. 7, pp. 163–178, September 1994.

[11] An ArborText Inc. SGML White Paper, "SGML: Getting Started," tech. rep., ArborText, Inc., http://www.arbortext.com/wp.html, ©1992, 1995.

[12] Stephen R. Mounce, "A Brief Discussion of the Standard Music Description Language," *http://www.techno.com/SMDL.html*, 1990.

[13] Anton Eliens, Martijn Van Welie, Jacco van Ossenbruggen and Bastiaan Schonhage, "Jamming (on) the Web," in *The 6th International WWW Conference Proceedings*, (Santa Clara, California, USA), W3C, April 1997.

[14] Steven R. Newcomb, Neill A. Kipp and Victoria T. Newcomb, "The "HyTime" Hypermedia / Time-based Document Structuring Language," *Communications of the ACM*, vol. 34, pp. 67–83, November 1991.

[15] Steven R. Newcomb, "Multimedia Interchange Using SGML / HyTime (Part 1: Structures)," *IEEE Multimedia*, pp. 86–89, Summer 1995.

[16] Steven R. Newcomb, "Multimedia Interchange Using SGML / HyTime (Part 2: Principles and Applications)," *IEEE Multimedia*, pp. 60–64, Fall 1995.

[17] J. Conklin, "Hypertext: An Introduction and Survey," *IEEE Computer*, vol. 1(9), pp. 17–40, September 1987.

[18] V. Bush, "As we may think," in *Atlantic Monthly*, pp. 101–108, July 1945.

[19] D. C. Engelbart and W. K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Conference Proceedings*, vol. 33, no. 1, 1968.

[20] T. H. Nelson, "Literary Machines," *Mindfull Press*, 1987.

[21] F. G. Halasz, "Reflections on NoteCards: Seven issues for the next generation of Hypermedia Systems," *Communications of the ACM*, vol. 31, pp. 836–852, July 1988.

[22] R. M. Akscyn, D. L. McCracken and E. A. Yoder, "KMS: A distributed hypermedia system for managing knowledge in organisations," *Communications of the ACM*, vol. 31, pp. 820–835, July 1988.

[23] N. Yankelovich, B. J. Haan, N. K. Meyrowitz and S. M. Drucker, "Intermedia: The Concept and the Construction of a Seamless Information Environment," *IEEE Computer*, vol. 1, pp. 81–96, January 1988.

[24] Stuart Goose, *A Framework for Distributed Open Hypermedia.* PhD thesis, University of Southampton, Faculty of Engineering and Applied Science, Department of Electronics and Computer Science, June 1997.

[25] Nechemia Daniel Beitner, *Microcosm++: the development of a loosely coupled object based architecture for open hypermedia systems.* PhD thesis, University of Southampton, Faculty of Engineering and Applied Science, Department of Electronics and Computer Science, September 1995.

[26] Frank Halasz and Mayer Schwartz, "The Dexter Hypertext Reference Model," *Communications of the ACM*, vol. 37, pp. 30–39, February 1994.

[27] K. C. Malcolm, S. E. Poltrock and D. Schuler, "Industrial Strength Hypermedia: The Requirements for a Large Engineering Enterprise," in *Proceedings of the ACM Hypertext '91 Conference*, (San Antonio, Texas, USA), pp. 13–25, December 1991.

[28] A. M. Fountain, W. Hall, I. Heath and H. C. Davis, "MICROCOSM: An Open Model for Hypermedia with Dynamic Linking," in *Hypertext: Concepts, Systems and Applications*, INRIA, Cambridge University Press, November 1990.

[29] H. C. Davis, W. Hall, I. Heath, G. J. Hill and R. J. Wilkins, "Towards an Integrated Environment with Open Hypermedia Systems," in *Proceedings of the ACM Conference on Hypertext*, (Milan, Italy), pp. 181–190, ACM, December 1992.

[30] W. Hall, I. Heath, G. J. Hill, H. C. Davis and R. J. Wilkins, "The Design and Implementation of an Open Hypermedia System," Computer Science Technical Report 92-19, University of Southampton, Department of Electronics and Computer Science, UK, 1992.

[31] Wendy Hall, "Ending the Tyranny of the Button," *IEEE Multimedia*, vol. 1, pp. 60–68, Spring 1994.

[32] Stuart Goose and Wendy Hall, "The Development of a Sound Viewer for an Open Hypermedia System," in *The New Review of Hypermedia and Multimedia*, vol. 1, pp. 213–231, 1995.

[33] Paul H. Lewis, Hugh C. Davis, Steve R. Griffiths, Wendy Hall and Rob J. Wilkins, "Media-based Navigation with Generic Links," in *Hypertext '96*, (Washington DC, USA), pp. 215–223, 16^{th}-20^{th} March 1996.

[34] Fred Halsall, *Data Communications, Computer Networks and Open Systems*. Addison Wesley, 3rd ed., 1992.

[35] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications (RFC: 1889)," tech. rep., Network Working Group, http://src.doc.ic.ac.uk/-packages/rfc/rfc1889.txt, January 1996.

[36] H. Schulzrinne, A. Rao and R. Lanphier, "The Real Time Streaming Protocol (RFC 2326)," tech. rep., Internet Engineering Task Force (IETF), http://-src.doc.ic.ac.uk/packages/rfc/rfc2326.txt, April 1998.

[37] Tim Dierks and Christopher Allen, "The TLS Protocol Version 1.0," tech. rep., Internet Engineering Task Force (IETF), ftp://ftp.nordu.net/internet-drafts/draft-ietf-tls-protocol-05.txt, November 1997.

[38] R. Fielding, UC. Irvine, J. Gettys, J. Mogul, DEC, H. Frystyk, T. Berners-Lee, MIT/LCS, "The Hypertext Transfer Protocol – HTTP/1.1 (RFC:2068)," tech. rep., Network Working Group, http://www.w3.org/Protocols/rfc2068/rfc2068, January 1997.

[39] Skip Pizzi and Steve Church, "Audio Webcasting Demystified," *Web Techniques*, pp. 55–60, August 1997.

[40] S. Bugaj, D. Bulterman, L. Hardman, J. Jansen, R. Lanphier, N. Layaida, J. Marsh, A. Rao, W. ten Kate, J. van Ossenbruggen, M. Vernick and J. Yu, "Synchronized

Multimedia Integration Language," tech. rep., The World Wide Web Consortium, http://www.w3c.org/TR/WD-smil, November 1997.