UNIVERSITY OF SOUTHAMPTON

Open Hypermedia and Temporal Linking with Audio Streams

by
Charles Neil Harwick Ridgway

A thesis submitted for the degree of
Doctor of Philosophy

in the
Faculty of Engineering and Applied Science,
Department of Electronics and Computer Science

January 2001

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE

ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

Open Hypermedia and Temporal Linking with Audio Streams

by Charles Neil Harwick Ridgway

Advances in inexpensive computer systems and the development of the World Wide Web (WWW) have revolutionised communication and information dissemination around the world. This has resulted in a rapid increase in the amount of published information, to the extent that many users find it difficult to navigate and retrieve this information over the Web.

Research communities have recognised this problem and have developed Open Hypermedia Systems (OHSs), to help manage and organise this information. These systems however have focused primarily on the visual domain, e.g. text and images, which is easier to manipulate. Most OHSs also store the link data in proprietary linkbase formats, which makes interoperability between systems difficult.

The WWW allows users to share and download different types of media, including text, graphics, audio and video. Streaming protocols have also been developed, to deliver media in real-time. This thesis describes the modifications that were made to an audio tool, used in an open hypermedia system, to support streams.

The thesis examines this streaming protocol in more detail and describes how it was extended to support temporal linking with audio streams, using the Fundamental Open Hypermedia Model (FOHM). The system developed is the first application of FOHM to stream media.

Contents

List of Figures

List of Tables

Acknowledgements

Trust in the Lord with all your heart,
And lean not on your own understanding;
In all your ways acknowledge Him,
And He shall direct your paths.

Proverbs 3:5-6, NKJV

I would like to thank my supervisor Professor David DeRoure, for his help in resolving technical matters, his time and his guidance, especially when writing my thesis! I would also like to thank Professor Wendy Hall for allowing me to work within the Intelligence, Agents and Multimedia research group, Dr Paul Lewis for being my internal examiner and Jon Ibbotson.

Thanks are also due to the guys in the lab for their friendship, encouragement and frequent coffee breaks / discussions about hypermedia! I would especially like to thank Stephen Blackburn, David Millard and John Griffiths for proof-reading sections of my thesis and to Kevin Page, Mark Thompson and Danius Michaelides for their input.

I would also like to thank my friends at Portswood Church and Navigators for their prayers, patience and general support; especially Gerrard and Alison Perry, Pieter and Suzanne van Leeuwen, Steve Redstone, John White and Paul Dennis.

I must also thank my family for their support, love and encouragement during the last few months of my thesis write-up.

Finally I would like to dedicate this thesis to my Lord and Saviour Jesus Christ, for His strength and guidance He has given me, over the years.

Chapter 1

Introduction

Towards the end of the twentieth century the development of advanced computer systems and distributed heterogeneous networks has steadily increased. It has become possible for the average user to obtain inexpensive *Personal Computers* (PCs) with powerful graphical interfaces and network connections. These systems can be used to create, store, manipulate and transmit large quantities of information to other users around the world; for example users can now communicate with other users, in different countries, using a mixture of video and audio. This is known as *video conferencing* and these systems use small video cameras and microphones to transmit images of the users and their voices to other machines.

The ability to obtain this type of equipment and to communicate with practically anyone around the world has resulted in the rapid adoption and use of these systems. This can be clearly seen in the growth of the *Internet* and more specifically the *World Wide Web* (WWW), which uses open standards for communication and data representation. The WWW allows users to author and publish their own web pages, to find information and to transmit and receive different types of media such as audio and video. However as the popularity of the WWW increases, the amount of information that is being published is also increasing, at a significant rate. It is possible to be "lost in hyperspace", as users can find it hard to find information and navigate their way around the Web. Search engines such as Alta Vista and Google have been developed to help alleviate this problem, but users can still be easily confused and overwhelmed by the large amounts of available information.

In 1945 a scientist named Vannevar Bush predicted that as time progressed, the

amount of scientific information would steadily increase. His landmark paper "As We May Think" [11] described a system that would allow users to easily browse and associate interrelated pieces of information. Approximately 18 years later Douglas Englebart created the *NLS/Augment* system to demonstrate this, see Engelbart and English [22]. As a result of this demonstration and as interest grew in this concept of handling information, several research communities developed applications, to handle large amounts of information in this way. These applications were called *hypertext systems* and they ran on text-based computer terminals. As more advanced computers, that could handle graphics and sound, were developed these hypertext systems were extended to support different types of media. They were called *hypermedia systems*, see Conklin [15] and Halasz [56].

These hypermedia systems allowed users to create and traverse associations or *links* between related pieces of information. However Fountain et al. [4], Dale [76] and Malcolm et al. [83] describe the limitations of these systems, including embedding the links within documents and proprietary document formats. As a result, these systems were known as "closed" systems because users could only create and follow links between documents within the same environment; interoperability between systems was difficult. To overcome these problems the research community developed *Open Hypermedia Systems* (OHSs), such as Microcosm and HyperWave, which stored the link information in separate files or *linkbases*.

As mentioned previously the first hypertext systems used text because of the limitations of the computer terminals. As more advanced computers were developed the research community extended the original text-based systems to form hypermedia systems. However the majority of the research into these systems, both open and closed, focused primarily on the visual domain; for example text, images and more recently video. There has been very little support for the audio domain because the developers found it easier to create graphical tools and applications to handle visual information.

Dix et al. [2] explain that the majority of interactive systems are completely visual in nature, with very limited support for audio. This is mainly due to the fact that the audio domain is considerably more complex and audio, by its very nature, does not have a unique visual representation. There are many ways in which users can visualize audio information; for example recording studios might use a waveform to represent the audio, while an operating system might use a scroll bar to represent the length and current position within an audio file. Goose and Hall [118] describe several techniques, that are

currently used, for visualising audio information in greater detail.

Originally the WWW had little support for audio. It was mainly a text-based system that could also support graphics, see Berners-Lee et al. [132]. However with the development of more powerful WWW applications or *Web Browsers*, users could download audio files and then activate an external application to playback the file. For small low-quality files this is fine, however as the quality of the audio increases so does the file size. The process of downloading a larger, higher-quality file can take a long time, especially if the network connection was poor.

To overcome this problem streaming media protocols, such as the *Real time Transport Protocol* (RTP) [8] and the *Real Time Streaming Protocol* (RTSP) [53], were developed. These protocols are used by streaming media players and servers for communication and data transfer. When a client-based player requests a media file, e.g. an audio or video file, the server splits the file into more manageable packets and then transmits them to the client. On receiving the packets the client will buffer enough of them, to ensure the quality of the playback is reasonable.

Pizzi and Church [119] explain that these streaming media protocols were also developed to provide real-time services for the current *Internet Protocol* (IPv4) [78] and the previous version of the WWW protocol, the *HyperText Transfer Protocol* (HTTP/1.0). HTTP/1.0 runs "on-top-of" IPv4, however both protocols were not designed to handle real-time data. The current HyperText Transfer Protocol (HTTP/1.1) does provide some of the functionality required for streaming media, however it still uses IPv4. To overcome this limitation, the next generation of Internet Protocol (IPv6) [116] has been designed, from the beginning, to support real-time services.

A relatively new standard called the *Synchronized Multimedia Integration Language* (SMIL) [128] is currently using RTP and RTSP, to create and stream multimedia presentations over the WWW. These presentations can stream audio, video, text and images. Both SMIL and the Web however embed the link information within their respective documents, which can result in the "dangling" link problem if the document is moved or deleted.

Several existing standards can also be used to create and stream multimedia presentations to clients. These include the *Multimedia and Hypermedia Experts Group* (MHEG) and the *Moving Pictures Expert Group* (MPEG). MHEG has been used for the creation and delivery of multimedia presentations, while MPEG has been specifically designed

to efficiently encode audio and video information for transmission. MHEG however is used to develop systems of which audio is just a small part and MPEG has only recently produced standards, that can manipulate the audio domain in more diverse ways, see Chiariglione [14] and Koenen [85, 86].

Open hypermedia systems, such as Microcosm and HyperWave, do have limited support for the audio domain. The audio samples are usually stored on the same machine as the OHS, which overcomes the problems of handling large media files. However as the quality and quantity of the samples increase, there is no guarantee that these machines will have enough space to store them.

One of the main problems of the current generation of OHSs, is their lack of interoperability, see Davis et al. [25] and Reich et al. [125]. The majority of these systems use proprietary protocols between their clients and link servers. As a result clients from different systems can not be used on different servers. Papers by Millard et al. [27] and Davis et al. [49] explain how the Open Hypermedia research community overcame this problem, by developing the *Open Hypermedia Protocol* (OHP).

Over time however OHP steadily grew in size until it was realised that a single protocol could not handle all of the functionality required. Therefore it was divided into smaller components, specifically designed for particular hypertext domains. Further research into these domains, by the Southampton members of this research community, produced the *Fundamental Open Hypermedia Model* (FOHM), see Millard et al. [27]. This data model concentrates on the interoperability between three of the most common hypertext domains: navigational, spatial and taxonomic hypertext.

1.1 Research Overview

The main aim of this research is to extend and enhance the Open Hypermedia paradigm to support recent developments in the audio domain; specifically *audio streams*. Currently authors can use SMIL to create multimedia presentations and then stream these presentations, to users on the Web, using streaming media protocols and dedicated servers. This technology however has not been used with the current generation of Open Hypermedia Systems (OHSs); only a few OHSs allow users to create links between audio and other media types. All of this media has to be stored on the same machine as the OHS.

OHSs on the other hand separate the link information from the actual data, while the World Wide Web and SMIL embed this link information within their documents. By combining the Open Hypermedia paradigm with audio streams, authors will be able to create temporal links *and* store them separately from the actual data; overcoming the problems associated with embedded links.

To obtain this research objective several existing standards and systems were investigated and they include:

- *Standards for content creation* – such as MHEG, MPEG and "off-the-shelf" multimedia authoring tools.
- *Standards for delivery* – such as the Internet Protocols (IPv4 and IPv6), RTP, RTSP and Digital Audio Broadcasting (DAB). MPEG also provides protocols for content delivery and these are examined as well.
- *Systems for handling Open Hypermedia* – such as Microcosm, HyperWave, OHP and FOHM.

1.2 Research Contribution

The work presented in this thesis contributes to the area of Open Hypermedia in three ways:

1. *By extending an existing Open Hypermedia tool to support audio streams* – this is achieved by modifying Microcosm's SoundViewer Tool, so that it can communicate with a streaming media server. The protocol used for this is RTSP and this new tool allows the creation of links to and from audio streams, stored on a separate machine.
2. *By extending a streaming media protocol to support Open Hypermedia* – this is achieved by extending the functionality of RTSP to support FOHM, so that authors can create, traverse and store temporal links. FOHM concentrates on the interoperability between three of the most common hypertext domains and therefore it can be described as an interoperable exchange format for open hypermedia. As mentioned previously however FOHM is just a data model; it does not provide a protocol for communication, between OHS clients and servers.

3. *By enhancing FOHM with a communication protocol* – this is achieved by combining FOHM with RTSP. This combination is mutually beneficial because FOHM provides the Open Hypermedia paradigm and RTSP provides the protocol for communication.

1.3 Thesis Structure

Chapter 2 describes the origins of hypertext and multimedia extensions to this form of non-linear text; hypermedia. It explains the reasons why the first and second generation of hypertext and hypermedia systems were known as "closed" and "monolithic" applications. This chapter gives an overview of several "open" hypermedia systems, that have supported the audio domain and existing hypermedia and multimedia standards. It closes by describing the development of a new high-level data structure, that can be used across three of the most common hypertext domains; navigational, spatial and taxonomic hypertext.

Chapter 3 talks about the audio domain and how it has been used in a number of applications and systems. It describes the dominance of the visual domain and how audio, when it is used in conjunction with the other four senses, can help in everyday situations. Existing standards for audio encoding and multimedia authoring tools are discussed. The chapter then explains how audio has been used in open hypermedia systems, including the World Wide Web.

Chapter 4 concentrates on streaming protocols for the Internet. It briefly describes the underlying Internet protocol suite, IPv4 and how these new streaming protocols interact with IPv4. RTP and a new framework or application-level protocol, RTSP, are discussed. Finally IPv6, the next generation of Internet protocol, is described.

Chapter 5 discusses the extensions that were made to an open hypermedia audio tool to support streams. This chapter describes the development of Microcosm's SoundViewer tool and a simple demonstration of this tool is then given. The design and implementation of the streaming SoundViewer is then discussed and another demonstration, of the new streaming functionality, is given.

Chapter 6 describes a new implementation of the RTSP protocol, that supports open hypermedia. It discusses the development of a new socket library, supporting both IPv4 and IPv6, which is used with this new implementation. This chapter then describes the

open hypermedia data model (FOHM), that is used to store and deliver link information. Modifications made to the original RTSP methods, for communication and playback of streams and the design of the new methods, for handling link creation and traversal, are then described.

Chapter 7 presents a simple demonstration of the Open Hypermedia Tool and it then examines how the work, carried out in this thesis, has achieved the original research objectives.

Chapter 8 concludes this thesis by bringing together the various threads of research. It also describes the possible future areas of research, that have arisen from the work discussed in this thesis.

1.4 Declaration

This thesis is based upon the work undertaken by the author within a collaborative research environment. It is all the original work of the author, except where explicitly stated otherwise.

Chapter 2

The "Open" Approach

2.1 Introduction

The ability to associate two or more related pieces of information is a common practice, that is used every day. Similarly the process of annotating and cross-referencing material, to help in its understanding, has been traced back as far[1] as Aristotle, who constantly referenced other authors' work. Supplementing material in this way has helped many authors and eventually the readers of their works, to more thoroughly understand the different aspects of their material.

This chapter briefly discusses several systems that have been used to help reference, annotate and access information. It describes the origins of hypertext and hypermedia and the development of "closed" hypermedia systems. It continues by describing several "open" hypermedia systems, that have developed tools to handle audio information. Existing hypermedia standards and the development of OHP and FOHM, a new associational structure for interoperability, are then discussed.

2.2 Hypertext and Hypermedia

The term "hypertext" has been used, over the last 40 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

[1] Approximately 384 B.C.

traditional text can be viewed. Conklin [15] 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. Their hypertext systems, discussed in Conklin [15], are outlined below:

1. Bush's *Memex* system. In 1945 Vannevar Bush, President Roosevelt's science advisor, predicted a rapid growth in the amount of scientific information and the need to automate the way in which this information should be browsed. In his article, Bush [11] 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[2]. As a result of this cooperation, 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, see Engelbart and English [22]. This system allowed users to create any number of associations or *links* between interrelated elements within a document and between the documents themselves.

3. Nelson's *Xanadu* System. During the development of the NLS, Ted Nelson was also developing his own ideas about augmentation. Nelson's system [131] would only allow the storage of documents in their original format and any modifications made to these documents, e.g. a different paragraph. By using links between these modifications and the original documents, previous versions could be easily reconstructed. New links could be easily created between different bodies of text and therefore new pathways could be formed through the material. It was from this

[2]One of the devices he invented was the mouse.

> system of linking large bodies of text together that Ted Nelson created the term "hypertext".

Hypertext systems allow users to author, edit and follow links between different bodies of text. With the introduction of more powerful digital computers, it has become possible to create links within and between documents containing different types of media, including text; for example, creating a link between a speech sample and the text representation of the speech. As Nelson [131] explains:

> It should have been obvious to anyone that general interactive media would appear and proliferate. Text, graphics, audio and video can now come alive in unified, responding, explorable new works that present facts and ideas; hypermedia. Unlimited new forms of connection and branching now offer the chance to explore ideas - to follow different lines of thought, different forms of exposition, different connections in a subject, in ways never before possible. The sequential writings and media of the past have given us only the dimmest precedents.

The term "hypermedia" can therefore be described as an extension to hypertext, to support multimedia information.

Halasz [56] 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 [15]. Halasz also explains that in the early 1980's, second generation systems began to emerge, which used workstations with more advanced user interfaces and graphics. These new hypermedia systems allowed users to create references between different types of media, e.g. text, pictures, video. Example hypermedia systems are NoteCards [56], Intermedia [101] and KMS [111].

Fountain et al. [4] and Dale [76] explain however, that these second generation hypermedia systems originally had a number of limitations including:

- *Proprietary document formats* – into which a document had to be converted before it could be used. Second generation hypermedia systems had their own proprietary document formats, which made it difficult to share information between systems. Once converted, a document could not be used by the original application that created it.

- *Embedded mark-up* – which involved inserting the start and end points of the links or *anchors* into the documents themselves. This made the documents considerably easier to transport. However this approach caused 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. This is known as the *dangling link* problem. Similar problems will also occur if documents are deleted and the links are not updated or removed.

- *Lack of extensibility* – in supporting other types of media, links and applications. External programs, that were not fully integrated into these systems, could be used to overcome this problem.

As a result of these problems, these second generation systems were known as "monolithic" or "closed" hypermedia applications. With these systems, users could only create and view hypermedia information, within the environment provided by the application. This information could not be shared with other hypermedia systems. Halasz [56], Beitner [99] and Goose [117] describe these systems in more detail.

2.3 "Open" Hypermedia Systems

Goose [117] explains that at the 1987 international hypertext conference, researchers started to express their concerns about the limitations of these second generation hypermedia systems, see **Section 2.2**. In his paper, Halasz [56] describes the ideas that were discussed at this conference, including new search and query mechanisms, management of dynamic information and more integration of existing applications. As a result of these discussions, several American research groups defined, in 1989, a reference model for hypermedia. It is called the *Dexter Hypertext Reference Model* [41] or usually just Dexter and it was designed to:

- Define both formally and informally the common abstractions found in a range of existing hypertext systems, e.g. NoteCards [56], Intermedia [101] and KMS [111].

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

- 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, see **Figure 2.1**, is widely regarded as being one of the most important developments in hypermedia research and it divides a hypermedia system into three *layers*:

1. The *Runtime Layer* – which provides a basic model for hypermedia presentation mechanisms and functionality to the user. This model will be used to access, view and manipulate the underlying hypermedia network structure.

2. The *Storage Layer* – which is the focus of the Dexter model. It describes the mechanisms used to form a hypermedia network structure, from nodes or *components* and the links used to connect them.

3. The *Within-component Layer* – which is concerned with the content and structure within the nodes and links of the hypermedia network. This layer of the model has been purposely left unspecified because there are many different ways to represent the content and structure of data.

In between the runtime and storage layer is the *Presentation Specification* interface. This is used to manage the presentation of components within the storage layer to the user. The form of this presentation can be a property of the component itself, a function of the hypermedia system or the path taken to reach the component. The *Anchoring* interface is between the storage and within-component layer. It is a mechanism that allows links to apply to a selection within a component, whilst ensuring that the two layers are kept separate.

A paper by Malcolm et al. [83] 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.

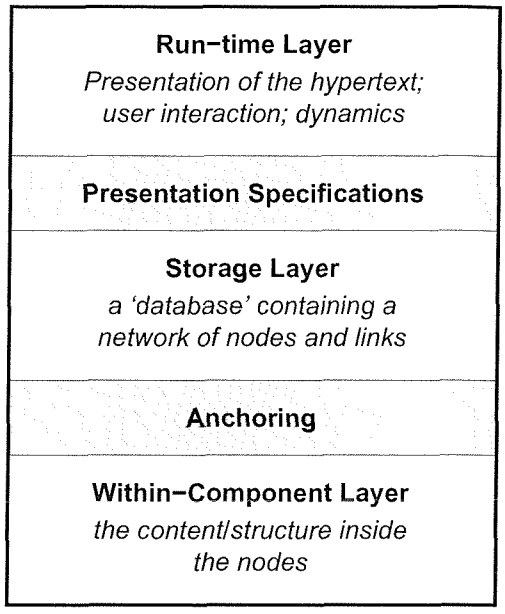

Figure 2.1: The layers of the Dexter Hypertext Reference Model.

Goose [117] describes that in 1991, at another hypertext conference, Halasz [56] revisited his original ideas 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. [83]. As a result of these discussions Halasz presented several new areas of research, which focused primarily on the development of "open" hypermedia systems with independent communicating processes and the way in which large amounts of information could be managed and visualised on workstation screens. With the development of these "open" systems, researchers would be able to overcome some of the problems associated with the second generation of hypermedia systems, see **Section 2.2**.

The following sections give an overview of several "open" hypermedia systems that can handle different types of media, including the audio domain. **Chapter 3** will discuss these systems' audio tools in more detail.

2.3.1 The World Wide Web

The World Wide Web or WWW was originally designed in 1990 by Tim Berners-Lee at CERN, the European Laboratory for Particle Physics. The aim of this project was

to provide a uniform way in which information could be shared over wide-area networks. Berners-Lee et al. [132] describe how most of the information at CERN, for example technical reports, data from experiments, 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 *Uniform Resource Identifiers* (URIs) [133] – which are short strings used to identify abstract or physical resources such as documents, images and files on the WWW. A URI is actually a generic term for all types of resources on the Web. It can be classified as a *Uniform Resource Locator* (URL), a *Uniform Resource Name* (URN) or both. A URL contains explicit instructions on how to access a specific resource on the Internet. A URN is a globally unique name that is given to a resource and it will remain unique, even when it ceases to exist.

   The generic format of a URI is:

   ```
   <protocol>://<internet host address>[:port]/<document path>
   ```

   The protocol, e.g. the *File Transfer Protocol* (FTP) and HTTP, is used to determine the type of communication between the user's machine and the Internet host address, which is usually a Web server. The host address is a unique value for a computer on the Internet. The port number is optional and it is used to determine to which port on the server, the user's computer connects. The document path is the location on the server to the required file. An example URI is "`http://www.foo.com/bar.html`".

2. The *HyperText Transfer Protocol* (HTTP) [108] – which is the protocol used for communication between WWW clients and servers. It is an Internet standard and it runs "on-top-of" the traditional protocols used for the Internet, see **Section 4.2**. This protocol was designed so that information could be efficiently retrieved, for the purpose of making hypertext "jumps"; see **Section 2.2** for a description of hypertext.

3. The *HyperText Markup Language* (HTML) [60] – which was originally defined using a SGML DTD, see **Section 2.4.2** and it is used to markup documents for the

WWW. The markup process allows authors to prepare and format their documents using a mixture of text, pictures, sound, video and *hypermedia links*. See **Section 2.2** 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.

2.3.2 Microcosm

Microcosm [5, 52, 143, 144] is an open hypermedia system that was developed by the *MultiMedia Research Group* (MMRG)[3] at the University of Southampton, to investigate new areas of hypermedia research. Goose and Hall [118] 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, that the user can then follow. In this system, links are stored in link databases or *linkbases*.

2.3.3 The Distributed Link Service

The *Distributed Link Service* (DLS) [48, 88, 89] was also developed by the MMRG at the University of Southampton. As mentioned previously in **Section 2.3.1**, the World Wide Web (WWW) uses open standards for the creation and delivery of hypermedia documents. To navigate around the WWW however, users must activate hypermedia links

[3]This group has recently been renamed to the Intelligence, Agents and Multimedia (IAM) research group.

which are embedded within the documents. These embedded links can cause problems because only the authors of the original documents can modify them.

The DLS overcomes this problem by providing link management and delivery services for the WWW. At the server side, a DLS *proxy* server is used in conjunction with a Web server. A proxy is an intermediate server that sits between a client and a Web server. It is allowed to modify the incoming and outgoing traffic and it is commonly used as a cache[4]. The DLS proxy server contains several linkbases; a default linkbase that is always used and additional linkbases, from which the user may choose. Each linkbase provides its own set of links or *contexts* and by selecting specific linkbases, users can tailor their linking activities to their current requirements.

The client interface is usually a Web browser, that has had its proxy set to the URI of a DLS-compliant server. Linkbases are set via a configuration document, that can be downloaded using a specific URI. Any other URIs that are typed into the browser will pass through the DLS to the intended WWW servers. The documents that are returned will be checked by the DLS for any relevant information. This information is obtained from the linkbases and if any is found, it is dynamically marked-up to form links. This new document is then displayed in the browser and users can activate these new links to obtain additional information.

The DLS also has another mode of operation, which allows users to statically compile the links into documents. With this operation it is possible to generate entire sets of Web pages, using the DLS's linkbases and these documents.

2.3.4 Mavis

The *Microcosm Architecture for Video, Image and Sound* (MAVIS), is an extension to Microcosm and it was also developed at the University of Southampton by the Multimedia Research Group. Lewis et al. [105] describe the system as being a tool that allows users to author *generic* links[5] 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.

[4]A cache is a data store, containing regularly visited documents. If a client requests a document, the cache is checked to see if it already holds a copy of this document. It it does the local copy is immediately delivered to the client; if it does not, then the cache passes the request onto the intended Web server. The returned document is then stored for possible future requests.

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

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 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. This is known as *fuzzy* matching. 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. Each description of a signature is stored in a module and users can create their own signature modules, which MAVIS can then use.

2.3.5 HyperWave

Kappe and Maurer [44] describe HyperWave, which was originally called Hyper-G, as being a large distributed open hypermedia system suitable for a wide range of applications. It was developed at the Graz University of Technology in Austria and it is a multi-user system, allowing native clients to access objects from information servers.

The information servers, which are also known as the *distributed link* servers, are complex object oriented databases. These databases contain descriptions of documents, links, anchors and other higher level structures, such as *collections* and *tours*. A collection is a hierarchy of related objects and when users visit a collection, they are given an overview of all the objects in this hierarchy. Users can then visit these objects. A tour is a collection that allows users to visit the substructures in a certain order; there is a set route through the information that the user has to follow.

HyperWave clients consist of several programs or *viewers*, which are used to author links and search and browse through the information on the server. At the heart of each viewer is a *session manager* which displays, in its main window, the structure of the data on the server. When a user traverses or follows a link, the session manager will start the relevant viewer to display any information related to this link. For example, a link to a video clip will start the "Film Player" viewer, which will then connect to the server and download the actual document. The session manager only provides information on

the documents / structures stored on the server; it is up to the viewer to download the document. More information on these viewers and the server can be found in [43, 44].

2.4 Hypermedia Standards

Currently there are two main international standards for the creation and delivery of multimedia and hypermedia presentations. These two standards are described in more detail in the following sections.

2.4.1 The Multimedia and Hypermedia Experts Group

The Multimedia and Hypermedia Experts Group[6] (MHEG) was formed in 1989 by the Joint ISO/IEC[7] Technical Committee (JTC 1) on Information Technology. MHEG is a working group (WG12 of SC29), with the mandate to develop international standards for the coded representation and interchange of multimedia and hypermedia presentations. This interchange usually occurs across heterogeneous systems and networks. There are currently eight parts to MHEG and these are discussed in more detail below.

MHEG-1 [65] is the first part of MHEG and it became an international standard in 1995. Rodriguez et al. [1] and Boudnik and Effelsberg [134] describe how it was the first technical specification for the definition of robust multimedia objects, the actions that can be applied to them, their behaviour and their interaction. MHEG-1 also defines the interchange format for these objects, using the *Abstract Syntax Notation One* (ASN.1) [72, 73] ISO standard. This notation is used to define software-neutral abstract syntaxes, which ensures that applications conforming to this standard will be able to communicate. For MHEG-1, it formally defines the syntax of all the multimedia data structures or *MHEG classes*. By creating *instances* of these classes, which are known as *MHEG objects* and by forming interrelationships between these objects, authors can easily develop and distribute interactive presentations.

MHEG-1 defines several classes and these include:

- The *MH-Object* class – which is the root class inherited by all the other classes. It contains a data structure for identification.

[6] Also known as the Multimedia and Hypermedia information coding Experts Group.
[7] International Organization for Standardization/International Electrotechnical Commission.

- The *Content* class – which contains either the audio-visual data itself or a reference to this data, depending on its size. Only a small amount of AV material, such as audio, video and text, can be included within this class. Referenced data, which could be stored on another machine on the network, will be retrieved at runtime.

- The *Link* and *Action* classes – which are used to describe the actions that will be performed on other objects, when a particular link object is activated. A link object is activated by an *event*, such as clicking on a mouse and pressing a key on a keyboard. A link object always defines a relationship between one source object and one or more target objects.

- 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 synchronization, such as lip synchronization.

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 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 using the *Standard Generic Markup Language* (SGML), see **Section 2.4.2**, instead of ASN.1. Due to a lack of resources however, the development of this part of the standard has been stopped.

MHEG-3 [66] became an international standard in 1996 and Rodriguez et al. [1] and Rutledge et al. [94] describe how it was designed to extend MHEG-1, by increasing the interactivity between the multimedia objects and the environments that they run in. This is achieved by using:

- *Scripting languages*[8] – which are programs containing procedures that can be used to monitor events generated by particular objects. For example, a user clicking on a button object will generate an event. When an event has been detected, these scripts will then execute certain actions, possibly on other objects. This allows objects to interact with each other.

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

- *A Virtual Machine* – which is used to create a mapping table between the run-time services, provided by a particular platform or environment, and the interface of the scripting language. This mapping table increases the interactivity between the scripts and the platform being used for the presentation.

MHEG-3 increases the functionality of MHEG-1 by allowing authors to develop presentations, using platform-neutral external programs or scripts.

MHEG-4 [63] is the fourth part of the MHEG standard and it is used to register objects and formats supported by MHEG; for example MPEG, see **Section 3.4.1**, JPEG[9]. It became an international standard in 1995.

The fifth part of the MHEG standard is MHEG-5 [67] and it became an international standard in 1996. Echiffre et al. [97] and Joseph and Rosengren [110] explain that it was developed to encode interactive multimedia applications into an interoperable format, that could be easily transmitted to terminals with limited resources. These terminals or *clients* are usually distributed across heterogeneous networks and they are simple devices, with a minimum amount of memory and processing power. Once created, MHEG-5 applications are stored on servers and when portions of an application are required, they are downloaded to the client. MHEG-5 clients contain an interpreter or *Runtime Engine* (RTE), which is used to interpret the most recently received portion of a presentation, present it to the user and then handle any local user interaction.

MHEG-5 defines several new classes, which are used to develop multimedia applications. These include:

- The *Root* class – which is inherited by all the other classes in MHEG-5. It provides a mechanism to identify objects and it also handles the semantics of object construction/destruction and activation/deactivation. This class is similar to the MH-Object class used in MHEG-1.

- The *Ingredient* class – which defines the common behaviour for all the objects that can be used in an Application or a Scene.

- The *Application* class – which consists of one or more Scene objects. This class can also contain objects derived from the Ingredient class, such as an Audio or a Stream object, that can be shared between these scenes. An instance of this

[9]The Joint Photographic Experts Group, an ISO standard for encoding still images.

class, an Application object, will define the entry point to the first scene within a multimedia presentation.

- The *Scene* class – which consists of objects derived from the Ingredient class that are temporally and spatially coordinated. A Scene object effectively describes a portion of the entire multimedia presentation.

- The *LinkEffect* and the *LinkCondition* classes – which are derived from the Ingredient class. 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 is derived from the Ingredient class. This class provides the functionality to multiplex audio and video objects, present them in synchronization and create links from MHEG objects to specific points in the stream. Links can also be created to specific user-defined events.

- The *Audio* and *Video* classes – which are used to encapsulate the audio and video information, respectively. They are subclasses of Ingredient and can be used with Stream objects.

- The *HyperText* class – which allows users to associate objects with a link to, for instance, another page. Objects can be words, groups of words and pictures.

At any time during the presentation, only one Application and Scene object can be active. When the presentation changes to a different scene, the current scene and hence its object is deactivated. The Ingredient objects used within this scene, if they are not used by another, are then deactivated and destroyed. When the presentation has finished, it is closed down and all of its objects are destroyed. A new application can then be started.

Bitzer and Hofrichter [54] describe how MHEG-5 also defines two encoding formats for the development and delivery of MHEG-5 applications. For the development of applications, an ASCII-based textual format is used. This is easier for authors to read and edit. For delivery, a binary format based on ASN.1 is used, which ensures interoperability across conformant terminals. This format also optimises MHEG-5 objects, so that they can be simply and efficiently parsed and validated by client engines. After the development phase of a MHEG-5 application, the textual format is usually translated into the binary format, for more efficient distribution.

MHEG-6 [68] became an international standard in 1997 and it extends MHEG-5 by defining an *Application Programming Interface* (API), written in the Java programming language and a *Java Virtual Machine* (JVM). This API which is also known as the *MHEG-5 API*, allows Java programs to access, manipulate and interact with MHEG-5 objects, within a multimedia application. It also allows these programs to access and control the basic services or functions of the MHEG-5 engine and the host terminal itself. The JVM is used in conjunction with the MHEG-5 engine, to interpret the Java programs.

Several organizations and consortiums are either using MHEG-5 or are considering MHEG-5 and its API (MHEG-6) for their systems. They include the *Digital Audio Visual Council* (DAVIC) [135], the *Digital Video Broadcasting* (DVB) Project [39, 138], the *Digital Television Group* (DTG) [136] and the *Digital Terrestrial Television Action Group* (DigiTAG) [137].

DAVIC was established in 1994 by an international consortium of manufacturers, companies and research organizations. Yasuda and Ryan [55] and Echiffre et al. [97] explain that the DAVIC specifications define the tools and the dynamic behaviour required, by interactive digital audio-visual systems, for end-to-end interoperability across different countries, applications and services. This is achieved by using existing international standards, such as MHEG-5 and its API, to define open interfaces and protocols, for the transfer of audio-visual information. The consortium has released several specifications, versions 1.0 to 1.5, of which the most complete is DAVIC Version 1.4.1 [30]. In 1998, the consortium also submitted DAVIC Version 1.3.1, as a Draft International Standard, to ISO/IEC JTC1. This specification, which is known as the DAVIC *Publicly Available Specification* or DAVIC PAS [69], became an international standard in 1999.

The DVB Project was established in 1993 by an international consortium of manufacturers, broadcasters, regulatory bodies and network operators, from a number of countries worldwide. Bitzer and Hofrichter [54] and Echiffre et al. [97] explain that the project's objective is to provide a complete solution for digital television and data broadcasting, across a range of delivery systems; for example networks, cable, satellite.

Evain [77] describes how this consortium is currently considering several APIs, including MHEG-5, for its *Multimedia Home Platform* (DVB-MHP) specification. This specification consists of the home terminal, its peripherals and the *In-Home Digital Network* (DVB-IHDN), which allows several terminals to communicate and share information within the consumer's home. Terminals can be computers, *set-top boxes* or televisions with an integrated set-top box. A set-top box is a device which is used to receive,

decode and then display digital broadcasts on a TV. Peripherals can be keyboards, joysticks and other external devices. A non-proprietary and open API, such as MHEG-5, will allow manufacturers to develop more advanced interactive multimedia applications, that can easily access the resources of the host terminal itself. Once created, these applications can then be used, by consumers, to view and interact with enhanced services such as the Internet, games, email, TV-banking and shopping.

The Digital Television Group and its sister organization, DigiTAG, were formed to establish *Digital Terrestrial Television* (DTT) within the United Kingdom and Europe, respectively. The DTG was founded in 1995 and consists of about 100 companies, whilst DigiTAG was established in 1996 and consists of a number of organizations from the broadcasting, networking and manufacturing domain. Both of these groups are using the DVB Projects' specifications and standards, to create advanced digital broadcasting systems. The specification for the DVB Multimedia Home Platform (DVB-MHP) however, is still under development and the API, for the home terminals, has not yet been finalised. Due to tight deadlines and other time constraints, both groups had to examine other APIs for their terminals and in 1997, they decided to use MHEG-5. They have also combined their research to develop and support a European version of MHEG-5, called *EuroMHEG* [17]. This extension is specifically designed to meet the requirements of the European digital community, e.g. by supporting European fonts and colour encodings. When the DVB-MHP specification finally becomes a standard, their existing APIs will be extended or modified, so that they will be fully compliant.

The final two parts of the MHEG standard are currently under development and they are MHEG-7 and MHEG-8. MHEG-7 will define a set of interoperability and conformance tests for MHEG-5 engines and applications. MHEG-8 will create a new encoding format for MHEG-5, using the *Extensible Markup Language* (XML), see **Section 3.5.5**.

2.4.2 The Hypermedia/Time-based Structuring Language

The Hypermedia / Time-based Structuring Language (HyTime) [70] became an International Standard in 1992 and uses the *Standard Generic Markup Language* (SGML) [61] 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) [7] 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. [87] 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; 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 [6], logical elements are described as pieces of data that may contain either text or other subelements such as chapters and paragraphs. 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 or a separate graphic file, 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 up to the application, however, to interpret the "meaning" of the document structure.

Mounce [120] describes in his paper, how the *Standard Music Description Language* (SMDL) was initially defined using a SGML DTD in 1988. SMDL is defined as

> An architecture for the representation of music information, either alone, or in conjunction with text, graphics or other information needed for publishing or business purposes.

SMDL basically divides a musical work into four domains and these are:

1. The *Logical* domain or *cantus* – which contains all 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. [3] suggest that the cantus can contain information to do with automated lighting.

2. The *Visual* domain – which graphically represents the music in some form, e.g. a link to an image file or to a coded music file.

3. The *Gestural* domain – which contains one or more links to the actual performance of the musical work, e.g. a link to a MIDI file.

4. The *Analytical* domain – which 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 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 became an International Standard in 1991.

Newcomb et al. [123] 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 [121] paper, an architectural form defines elements with a standard meaning, e.g. independent hyperlink, and syntax for 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 and multimedia documents and therefore, is often referred to as a *meta-DTD*.

One of the main problems of the hypermedia and multimedia industry is its inability to store, describe and transport media objects in an application-neutral manner. Newcomb [122] 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. Papers by Newcomb [121] and Newcomb et al. [123] give a thorough description of each module. 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 – which is an extension to the current method, used by SGML, to reference elements within documents. SGML uses two tags to reference these elements; a unique "identifier" (`#ID`) and an "identifier reference" (`#IDREF`). SGML can also create simple links to particular entities such as 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. 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 – which 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 such as time, money and temperature. Objects within FCSs are called *events* and these are used by the rendition module.

6. The *Rendition* module – which defines 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.

HyTime is a complex standard that is primarily used to express document architectures in SGML. It is not an application and as a result of this and its complexity, very few implementations of the HyTime standard actually exist.

2.5 The Open Hypermedia Protocol

Davis et al. [25] and Reich et al. [125] explain that in 1994, at the first International Workshop on Open Hypermedia Systems [140], the open hypermedia research community discussed the problems of the current generation of open hypermedia systems (OHSs). The community realised that these systems were "open" in the way that they handled the link information, see **Section 2.3** and how the client-side applications could use this information, e.g. creation, manipulation and traversal. They also explain that the majority of these systems stored the link information on separate *link servers* and that they used their own proprietary protocols, for communication, between these servers and the clients. As a result of this, different open hypermedia systems could not interoperate because the client-side tools, used in one system, would not be able to communicate with the link servers used in another.

The research community also noticed that the developers of these systems were spending a considerable amount of time re-implementing existing tools; to ensure that they could communicate with their own OHSs. To support new data formats, either the existing tools would have to be modified again, or new client-side tools would have to be created. The open hypermedia community came to the conclusion, that the developers were wasting too much time on these tools, instead of concentrating on the main areas of research, the link servers or *services*.

Millard et al. [27] and Davis et al. [49] describe how Antoine Rizk, at the first workshop in 1994, proposed that the open hypermedia community should concentrate on the development of a new lightweight message-based protocol, to overcome these problems. He described how each of the current generation of open hypermedia systems had similar features and functionality; for example the communication protocols, used in these

systems, were confidential and proprietary in nature, but they all performed similar tasks. With this in mind, he suggested that the most common features of these systems could be combined to form the *Open Hypermedia Protocol* (OHP). The OHP could then be used to create a standard set of interoperable client-side tools.

In 1996, at the Second Workshop on Open Hypermedia Systems [141], two important events occurred. The first was the formation of the *Open Hypermedia Systems Working Group* (OHSWG), which meets every 6 months to coordinate the efforts of the OHS research community. The second was the presentation of the first draft of the Open Hypermedia Protocol, by Davis et al. [49]. This draft introduced the concept of a protocol *shim*[10], which would convert messages, delivered by the OHP, to the native format used by the open hypermedia system. The protocol itself consists of text-based messages, containing ASCII tags and values. It was heavily influenced by the authors' own experience with hypermedia systems, e.g. Microcosm's message model, see **Section 2.3.2**. The client-side tools would communicate with the shim using the OHP, which would then communicate with the link servers, using the OHS's own proprietary protocol format. The link servers would not have to be modified because the developers would only have to create the shim. Once created any OHP-aware client would then be able to communicate with any server, via these shims.

After several refinements, the draft was ready to be used and the OHSWG concentrated on developing systems, to demonstrate interoperability. However, Millard et al. [26, 28] and Davis et al. [50] describe several of the problems that became evident, with the scope of the draft proposal and hence the protocol itself, during the development of these systems. These included:

- The protocols' complexity and size. Originally the protocol was designed to provide a standard interface, between the clients and the servers. As the work progressed however, the complexity and size of the protocol steadily increased. For example, the group could not agree on which type of on-the-wire protocol to use and so several were included. Aspects of resource location and naming also caused confusion. As a result, this "lightweight protocol" steadily grew to encompass several different aspects of the hypermedia domain.

- Problems with the text-based messages. The protocol consisted of a sequence of tag / value pairs, which proved difficult to parse. Inconsistencies in the naming of

[10] A "shim" is a thin piece of material, used in machinery, to make parts fit.

these tags and the handling of the objects, also caused problems.

The OHSWG soon realised that it was impossible for a single protocol to handle all of the different aspects of the open hypermedia domain and so they decided to split the protocol into more manageable sub-domains. OHP was renamed to the *OHP Navigational Interface* (OHP-Nav) and its functionality was reduced, so that it dealt with the more traditional form of navigational hypertext exclusively.

Millard et al. [24, 27] and Reich et al. [125] describe navigational or *associative* hypertext, as being the most widespread and common form of hypermedia in use today. The World Wide Web, for example, has implemented its own form of navigational hypermedia, see **Section 2.3.1**. Vannevar Bush [11] and several other early pioneers, see **Section 2.2**, originally defined the concept of navigational hypertext, as the process in which humans "link" or "associate" related pieces of information together. By recalling one item of information, it became possible for users to recall other related pieces of information; in other words recalling the item or items "linked" to.

By concentrating on this domain, it became possible for the OHSWG to define the main abstractions for this protocol as:

- A *Node* – which can be described as a "wrapper" for an arbitrary resource. It contains information about a *content specifier* (ContentSpec). The ContentSpec can contain URIs to data or the actual data itself, e.g. a picture and text.

- An *Anchor* – which contains an identifier and an optional *location specifier* (LocSpec), for the identified object. This LocSpec provides a handle to either the entire object or an area within the object, e.g. a byte offset into a text file and the coordinates for an area within an image.

- An *EndPoint* – which consists of an anchor identifier and a link identifier. It is used to bind exactly one anchor to one link. An anchor can be bound to more than one endpoint.

- A *Link* – which is a relationship between zero or more endpoints.

Figure 2.2 is an example of two links in OHP-Nav. `Link 1` contains two endpoints (`Endpoint 1` and `Endpoint 2`); both of which point into `Node 1`. With this link, one of the anchors could point to an entire document and the other, point to an area within

this document. `Link 2` again has two endpoints, however one is bound to `Node 1` and the other is bound to `Node 2`, which could be a URI to an image. Therefore if `Link 2` is followed, the user could go from an area within a document to an image.

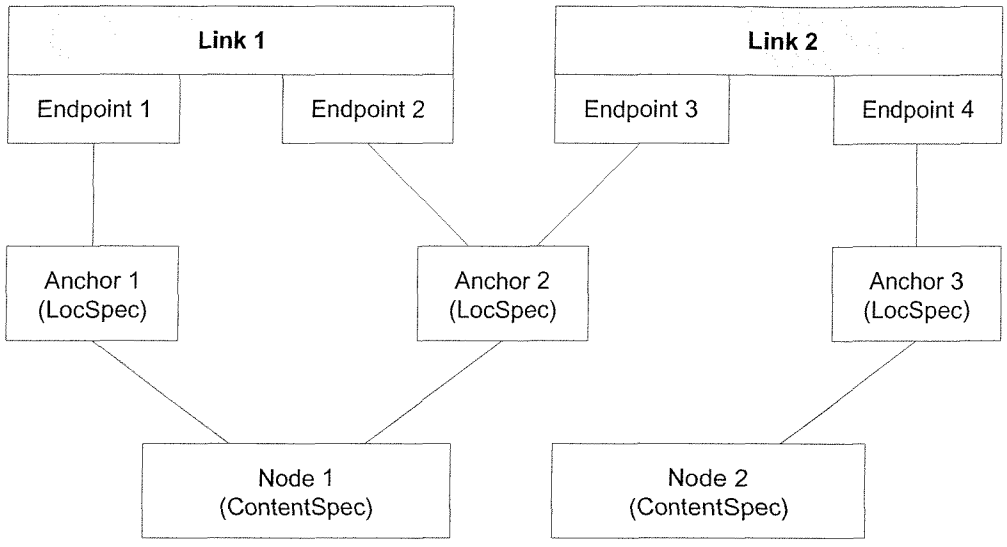

Figure 2.2: The OHP Node Link model.

Millard et al. [28] describe how in 1997, the interface definition for OHP-Nav [51] was considered complete enough, for developers to begin initial implementations. The proposal originally used the text-based message format for communication. However this format, as mentioned previously, is difficult to parse and therefore the group decided to use a new format. Two standards for handling this type of information were investigated and they are outlined below:

1. The *eXtensible Markup Language* (XML) [152] – which has recently become a standard. XML is derived from SGML, see **Section 2.4.2** and it has been used in a number of different tools and applications. For OHP-Nav, a XML *Document Type Definition* (DTD) was defined. This is used to efficiently parse and verify OHP-Nav objects, to ensure that they conform to the specification.

   When using XML clients that are OHP-Nav aware, pass object identifiers to the servers, which resolve them directly. These servers then send the relevant objects, encoded in XML, back to the client. For a more detailed description of XML and DTDs see **Section 3.5.5**.

2. The *Common Object Request Broker Architecture* (CORBA) [19] – which is a standard developed by the *Object Management Group* (OMG) [102]. The OMG [16] describes CORBA as being an open, vendor-independent architecture and infrastructure, that allows applications to interoperate over heterogeneous networks. By using CORBA and its protocols, vendors can communicate using different machines, operating systems and programming languages.

   For OHP-Nav, the OHSWG decided to create a second definition of OHP-Nav using the CORBA *Interface Definition Language* (IDL); so that it would be possible to transmit and receive OHP-Nav objects over different communication mechanisms. The IDL, as the name suggests, specifically defines the interface for each object being used. This interface contains strict declarations for each of the functions and their parameters, used by the objects.

The XML DTD for OHP-Nav could be used immediately, for initial prototype implementations. The CORBA interface definition however, raised a few issues. For example, each object that has an interface requires a name, which ensures that an application can access that object by simply using its name. At this time however, there had been very few discussions, within the OHSWG, about the naming of OHP-Nav objects. Other issues involved how the OHP-Nav messages would be passed between CORBA-aware clients and servers, e.g. as plain ASCII text or as IDL mapped methods in a new object. Millard et al. [28] describe how a simple CORBA implementation was eventually developed. The OHSWG decided however, that the issues mentioned above required further research.

Millard et al. [28] and Reich et al. [124, 125] describe three open hypermedia systems, that have implemented OHP-Nav:

1. The *Southampton CSF* system – which was developed by the *Intelligence, Agents and Multimedia* (IAM) research group[11] at the University of Southampton. This system consists of a picture viewer, a linkserver and the *Client Side Foo* (CSF) communication mechanism.

   The picture viewer is an application that supports link traversal and endpoint creation. The linkserver is a database that stores the link information and the CSF handles the communication between the viewer and the server. The CSF contains an engine that handles the OHP-Nav messages and it is designed to be easily modified, so that different networking protocols can be supported. Currently it uses

[11] Formally known as the MultiMedia Research Group (MMRG).

TCP/IP for communication, see **Section 4.2**.

2. The *Solent* system – which was again developed by the IAM research group at the University of Southampton. This system uses two servers, which provide navigational and content-based retrieval services. Content-based retrieval allows users to retrieve information, and possibly navigate to new pieces of information, based on the content stored within hypermedia documents. An application might support navigation, by finding "similar" images based on their characteristics, such as colour distribution, different shapes within the images.

   Solent has four client applications and they include a wrapper for a word processor, a picture-viewer and a "car stereo", which was used to play audio files.

3. The *Construct* system – which was developed at Aarhus and Aalborg Universities in Denmark. This system has three main servers, which provide navigational and spatial hypermedia services, as well as collaboration. Collaboration allows multiple clients or users to share and interact with a particular environment, e.g. a shared document, that allows different users to modify the content.

   Construct has four client applications and they include wrappers for a word processor, e.g. Microsoft's Word, and a Web browser.

At Hypertext 98, the Southampton CSF system and a simplified version of the Construct system were both demonstrated. They showed, for the first time, client interoperability between different OHSs. It was possible, for example, to create one endpoint in the picture viewer and then another in one of the Construct clients. These links could then be followed from either client. As a result of these demonstrations the amount of interest, in the work of the OHSWG, increased. The group decided therefore, to do another demonstration at the next Hypertext conference.

Millard et al. [26] explain, that for the next conference the group decided to demonstrate different aspects of the OHP-Nav protocol. At Hypertext 99, the Solent system and the current version of the Construct system, mentioned above, were demonstrated. After this conference however, the OHSWG realised that the different hypertext domains were not that different after all. The Solent system, for example, demonstrated a versatile storage component that held arbitrary data structures in XML. This component does not know anything about the data structures it stores; it only understands the structure of the XML. As a result of this and further discussions within the group, the Southampton

members of the OHSWG focused on defining the highest level of structure, that would work across all the domains.

To define this high level structure, three of the most common hypertext domains were investigated. One of these is navigational hypertext, which has already been described in this section. Millard et al. [24, 27] describe the other two domains and they are:

1. *Spatial* hypertext – which is also known as *Information Triage*. This domain allows users to express relationships between different pieces of information or nodes, using visual characteristics such as proximity, colour and shape. It is possible to combine similar nodes together, to form *collections* and to organise these collections using *Abstract Data Types* (ADTs), such as sets, lists and stacks. The visual representation of these node are known as *composites*.

   The relationships between these nodes and collections are implicit; defined by the visual characteristics. Therefore to ensure that this information can be queried, most spatial systems use a *spatial parser*, to convert these implicit relationships into explicit associations. This parser is also used to decide on an appropriate structure such as a set to store this information.

2. *Taxonomic* hypertext – which uses taxonomies to categorise items of information, called *artifacts*. Taxonomy is the science of classification and it has been used by biologists, for example, to classify living and extinct organisms. An artifact is similar to a node.

   Set theory is used to model taxonomic hypertext and users sort artifacts into *categories*, which are represented by sets, based on their characteristics. Different users might view the information in different ways and therefore, users can have different *perspectives* on the way the information is categorised. For example, one user might categorise three artifacts as being similar. Another user might categorise them completely differently and therefore, two different perspectives will be defined, by the users, for these artifacts. Users can navigate the taxonomy by moving between overlapping categories or perspectives.

All three domains have several similarities. For example, data is represented as a node in navigational hypertext, as a visual entity in spatial hypertext and as an artifact in taxonomic hypertext. However, each domain has its own unique features, e.g. navigational hypertext allows users to define anchors that link to areas within documents

and taxonomic hypertext allows users to define perspective objects, which diverge the relationships within the categories, according to different views.

To ensure that the new high level structure could work across all three domains, Millard et al. [27] decided to take the unique features of each one and create corresponding features or data structures in the others. They describe how spatial hypertext provides internal structure, using its ADTs. These structures could be used, in navigational hypertext, to organise the different parts of large documents and provide more structured movement through the information. For taxonomic hypertext, these structures would be another feature of a category, which would allow perspectives to split over structure as well as content.

Two other important areas are link traversal, the process of following a link and arrival, the process of viewing a structure. For spatial hypertext, traversal and arrival will depend on the ADT being used. For example, arrival in a set will display all of the elements in that set, while traversal will display all of the elements except the starting endpoint. A more thorough description, of the interoperability between domains, is given by Millard et al. [27].

This work resulted in the creation of an abstract data model to represent the relationships between these domains. This data model was called the *Fundamental Open Hypermedia Model* (FOHM) and it is used to map from one domain to another. This ensures interoperability. FOHM is defined using an XML DTD, see **Section 3.5.5** and it introduces four new objects:

1. A *Data* object – which is a simple wrapper for information. It could represent a document, file or any other type of information.

2. A *DataRef* object – which contains a *feature vector.* A feature vector is used to determine how a dataref maps to the *feature space.* The dataref also contains an optional LocSpec, which has been described previously.

3. A *Binding* object – which binds a dataref to an association, using the feature vector.

4. An *Association* object – which contains a list of features, known as a *feature space.* All objects in the association must map to this feature space. It also contains a set of bindings.

Figure 2.3 is an example of a FOHM structure, from a navigational link to a spatial

list. The feature space for both associations is shown, in brackets, after the `Association` tag. For links, it is `direction` and for spatial lists, it is `position`. The feature vector is shown, again in brackets, in the `Binding` box. For navigational links, there are three types; a source (`src`), a destination (`dest`) and a bi-directional (`bi`), which is not shown in the figure. For spatial lists, the feature vector is just a number, representing the position in the list. If the navigational link is followed, the user will go from an area within a document to the spatial list. On arrival, all of the elements within this list will be displayed.

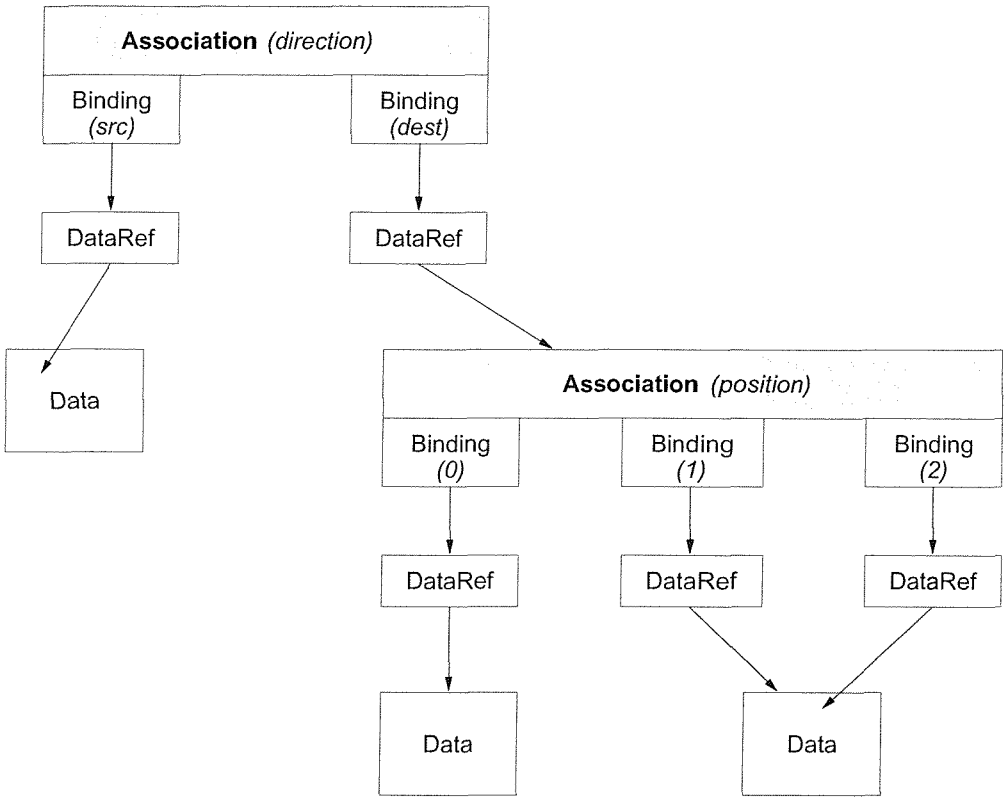

Figure 2.3: A Navigational Link to a Spatial List.

FOHM has been designed to support a number of different domains, not just the three mentioned above. It is possible to extend FOHM with new data structures. However at this stage, it will not be able to "understand" them. As a result, clients will not be able to edit the associations because they will not understand the implications of their modifications.

The development of FOHM has formed a new high-level, associational structure for interoperability. FOHM-aware applications, for the *SoFAR* [93] agent framework, have

already been developed, to demonstrate spatial and navigational browsing of the same structures. The focus of the group however, has changed from a single all-encompassing protocol, to the actual data structures themselves. As a result of this, the infrastructure for a powerful open hypermedia framework, is now being developed.

2.6 Discussion

In the twentieth century there has been a phenomenal growth, in the amount of scientific and publicly available information. However systems that can be used to store and manage this information have only been developed towards the end of this century. This was mainly due to the lack of technology, both hardware and software, that could be used to handle this information in a suitable way.

Several research communities and standards' organisations have approached this problem, using different techniques and systems. The hypermedia research community was one of the earliest developers of hypertext systems, which could be used to augment the traditional form of linear text within documents. These systems allowed users to create, edit and manipulate links between related bodies of text; in effect mirroring the way the human mind associates related pieces of information. As more advanced digital computers were introduced this community developed hypermedia systems, which allowed the creation of links between documents containing different types of media, such as audio and video.

These systems however, were known as "closed" second generation systems because they had several limitations, including the use of proprietary document formats and the embedded markup of links. As a result these systems could not share information. To help overcome some of these problems the Dexter Hypertext Reference Model was defined. It was designed to be used as a template for future standards and to define several common layers to handle the presentation, storage and link management of hypermedia information. This model was an important development in hypermedia research however, it has been criticised. Certain parts of the model have incomplete specifications, such as the composites and it does not address the dangling link problem.

After several conferences, the hypermedia research community concentrated on new areas of research, specifically the development of Open Hypermedia Systems (OHSs). These OHSs were designed to overcome the problems of "closed" hypermedia systems and some of the most successful OHSs include:

- Microcosm – which was designed to extend the functionality of the native operating system by using a set of communicating processes and viewers. These viewers are tools that allow users to interact with the system and they are either specifically designed for Microcosm or are native applications, that have been made Microcosm-aware. As a result, this OHS has the ability to cope with a large number of documents (10,000 and above) and it separates the links for different users. When users log in, the linkbase(s) for each user are processed and the links will be displayed in the relevant documents.

  In Microcosm however, the editing and following of links can cause a few problems. Links are not embedded within the documents and therefore if the documents are modified, outside of the Microcosm environment, then the links may no longer be accurate. If the document is removed, then the dangling link problem will occur. The developers of the system have attempted to solve this problem, by attaching time stamps to each link and node. When these change, the system will attempt to repair itself or notify the user with a warning. When following a link to a large document, Microcosm-aware viewers will attempt to move the view to the exact position of the link endpoint. Unaware viewers however can not do this.

- MAVIS – which is an extension to Microcosm and it allows users to create links between text and non-text media. It achieves this by using fuzzy matching techniques with signatures, to determine the closest match between similar types of non-text media. These signatures are stored within files and represent the content of a selection; for example the values of the colours used within an image. MAVIS provides a good framework for testing different fuzzy matching algorithms, especially against each other. However at this stage of development, it can only handle images and text. It has limited support for audio and this is discussed in more detail in **Chapter 3**.

- Hyperwave – which was originally known as Hyper-G. It is described as being a large distributed open hypermedia system. This system consists of information servers which contain descriptions of higher-level structures such as documents, links, collections and tours. The Hyperwave clients consist of several purpose built viewers, to handle different types of media and a session manager. It has support for multiple languages, so that links can be made available regardless of the original language that was used for a selection. The design of the Hyperwave system meets many of the requirements of open hypermedia, however the servers are

not inherently extensible and therefore they are difficult to customise for particular users.

Another "open" hypermedia system is the World Wide Web (WWW), which has steadily grown from being a research project at CERN laboratories, to the most widely used mechanism, over the Internet, for information retrieval, delivery and exchange. The WWW can be described as being "open" because it uses open standards for the markup of documents (HTML) and the delivery of these documents, from the servers to the clients (HTTP). However, it violates some of the most common principles of open hypermedia; all documents have to be converted into HTML so that they can support embedded links, only unidirectional (one-way) navigation is supported and links can be broken when documents are moved, which results in the dangling link problem.

The Distributed Link Service (DLS) was developed to help alleviate some of these problems. It provides link management and delivery services for the WWW. A DLS proxy server is used in conjunction with a Web server to dynamically markup HTML documents, requested by the users, with extra link information. This information is provided by external linkbases, which are selected by the users via the proxy server. Carr et al. [89] explains however, that even though the DLS separates some of the link information from the Web pages, problems will occur with keeping track of the documents to which each link points. Also the original software assumed that the linkbases would be stored on the same machine that was running the server. Later versions of the software used URLs to access linkbases, that are stored anywhere on the Web. As a result it could be possible for the author to lose control of the source and destination documents, as well as the links themselves.

Whilst these Open Hypermedia Systems were being developed, the International Standards Organisation (ISO) was also developing two hypermedia standards; HyTime and MHEG. HyTime was designed to overcome the problems of storing, describing and transporting media objects in an application-neutral manner. It is usually described as a meta-DTD, that uses SGML to construct the architecture of documents. However it is a complex standard and as a result very few implementations actually exist. MHEG was designed for the development and delivery of multimedia and hypermedia presentations, across heterogeneous networks. It consists of eight parts of which the most popular are MHEG-5 and MHEG-6. These two parts of the standard are being used in a number of systems, to help in the delivery of digital television and data broadcasting. MHEG is far

simpler than HyTime and it can be used for applications that require real-time information interchange. It is primarily used however, as a tool to develop visual, easy-to-use, interoperable distributed systems, of which audio is just a small part.

The open hypermedia research community have discussed, at several conferences, the problems of Open Hypermedia Systems. They realised that the vast majority of these systems were "open" in the way that they handled the link information, at the client and the server. They also realised however, that these OHSs could not interoperate because each system used its own proprietary protocols, for communication between the clients and the server. As a result of this the Open Hypermedia Systems Working Group (OHSWG) started the development of the Open Hypermedia Protocol (OHP), which focused primarily on a protocol for interoperability.

As work continued, the protocol steadily grew in size until the group realised that a single protocol could not be used, to handle all of the aspects of open hypermedia. The functionality of OHP was reduced so that it handled the traditional form of navigational hypertext and it was renamed to OHP Navigational (OHP-Nav). OHP-Nav defines a protocol for communication and the data structures required to handle navigational hypertext. It was successfully demonstrated at two conferences.

After these conferences the Southampton members of the group focused on the development of a higher-level structure, that could work across the three main hypertext domains; navigational, spatial and taxonomic hypertext. This structure was called the Fundamental Open Hypermedia Model (FOHM). This model however, does not describe an interoperable communication protocol. It just describes an associational structure for interoperability. Therefore to use this model, over heterogeneous networks and distributed systems, an underlying transport protocol has to be used.

2.7 Summary

This chapter has discussed the origins of hypertext, hypermedia and the development of "closed" hypermedia systems. Dexter, a hypermedia reference model, is described and the concepts of "Open" Hypermedia Systems (OHSs) are discussed. Several OHSs that have developed tools to handle multimedia documents, including the audio domain, are described.

Existing hypermedia and multimedia standards, developed by the International Standards Organisation (ISO), are reviewed and they are MHEG and HyTime. The complexity of HyTime has resulted in very few systems actually been developed. MHEG on the other hand, is a far less complex standard and it can be used to create and deliver multimedia presentations, although the audio domain is only a small part of this.

Finally this chapter describes the development of the Open Hypermedia Protocol (OHP) and OHP-Navigational (OHP-Nav), which is a subset of OHP. The Fundamental Open Hypermedia Model (FOHM), which is a higher-level interoperable associational structure, is then discussed.

The next chapter describes the importance of the audio domain and how it has been used in a number of systems, including the "Open" Hypermedia Systems discussed in this chapter, multimedia authoring tools and existing International Standards.

Chapter 3

The Audio Domain

3.1 Introduction

This chapter describes the benefits of the audio domain and how it has been used in a number of systems. Traditionally the sense of sight has always been regarded as the primary sense for normally sighted people. It has been shown however, that sound is just as important as sight. The human ear for instance can detect different types of sound, their location and how far away they are.

Multimedia authoring tools use different types of media, including sound, to create multimedia applications and three of the most common visual environments, for creating these applications, are discussed in this chapter. Two existing standards for handling audio information are also discussed and they are MPEG and Digital Audio Broadcasting.

The previous chapter describes the WWW and "Open" Hypermedia Systems (OHSs), which are being used to handle large amounts of information in different ways. The WWW uses embedded links to allow users to navigate around this information. OHSs on the other hand, store the link information separately from the documents and allow users to create, edit and follow links in this information space.

This chapter describes how the audio domain has been used in "Open" Hypermedia Systems and the WWW. Finally the chapter draws to a close, by describing how a multimedia presentation can be created, using a new declarative language, for the Web, called SMIL.

3.2 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 sight, hearing, touch, taste and smell. This information is then processed and depending on the results, humans will react or respond 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 and therefore they do not receive as much attention. This is mainly due to their complexity.

Dix et al. [2] 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 channels could be used to relieve the pressure of the visual channel and thus reduce the information overload. With humans there are two types of channel, one for input and another for output. An input channel represents one of the five senses, whilst an output channel represents a response, e.g. moving a leg, walking, talking. 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 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. [2] 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.

To understand the audio domain an overview of the human auditory system is required. Dix et al. [2] and Moore [9] 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* – which 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* – which consists of the *cochlea.* This part of the ear has rigid bony walls and is filled with a special type of fluid. It 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 [9] 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. [2] describe the different characteristics of sound, such as the pitch and amplitude. If the pitch of the sound increases, then the frequency will also increase. The human ear can hear frequencies from about 20 Hz to 15 KHz. 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.

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 user's 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 listener's 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 1**.

3.3 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 and floppy disks. 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 to 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 [104]).

3.3.1 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 bird's 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. **Figure 3.1** 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.

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, such as a screen transition or a video started, 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.

3.3.2 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 and key presses,
- *actions* to be performed after an event such as a transition or a sound,
- *routines* to perform branches such as loops, decisions and interactions.

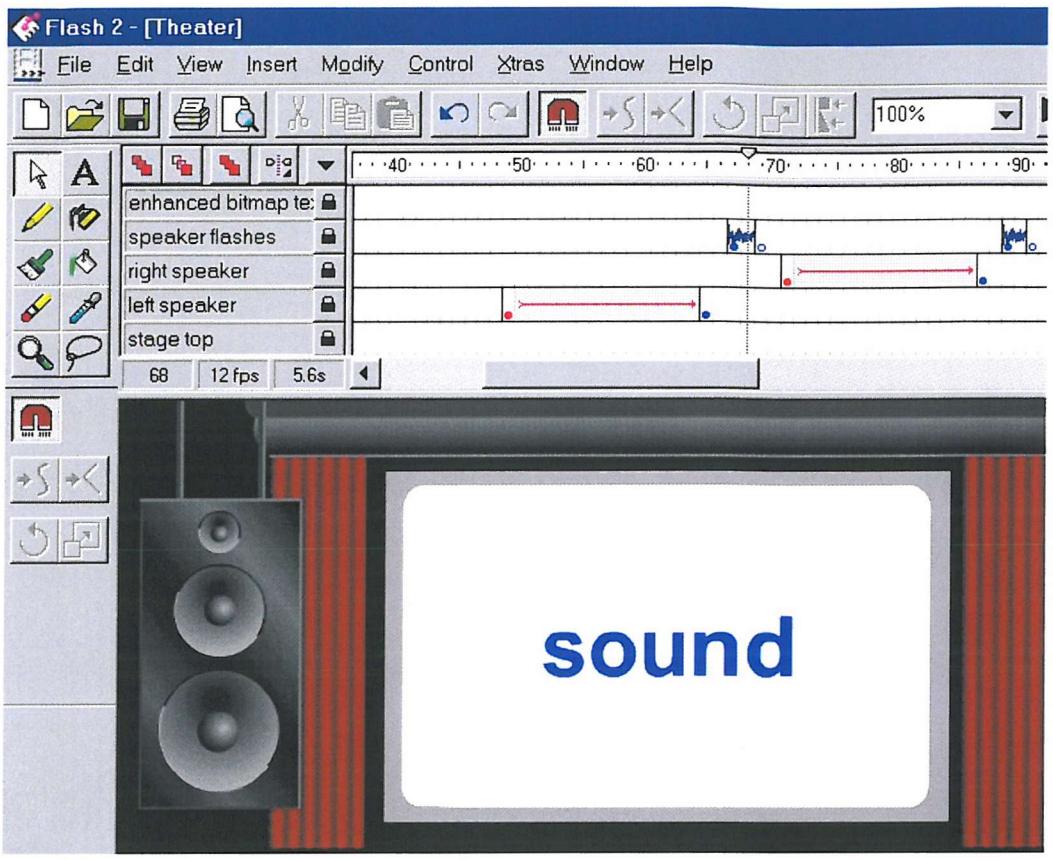

Figure 3.1: A screen-shot of a Macromedia Flash presentation.

The flow of control within the flowchart is usually from the top to the bottom. Branches such as a loop or a conditional branch, are allowed and users can also insert smaller flowcharts within the main flowchart. **Figure 3.2** is a screen-shot of Macromedia's Authorware program, displaying specific icons and decision branches.

Clicking on each icon usually brings up 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.

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.

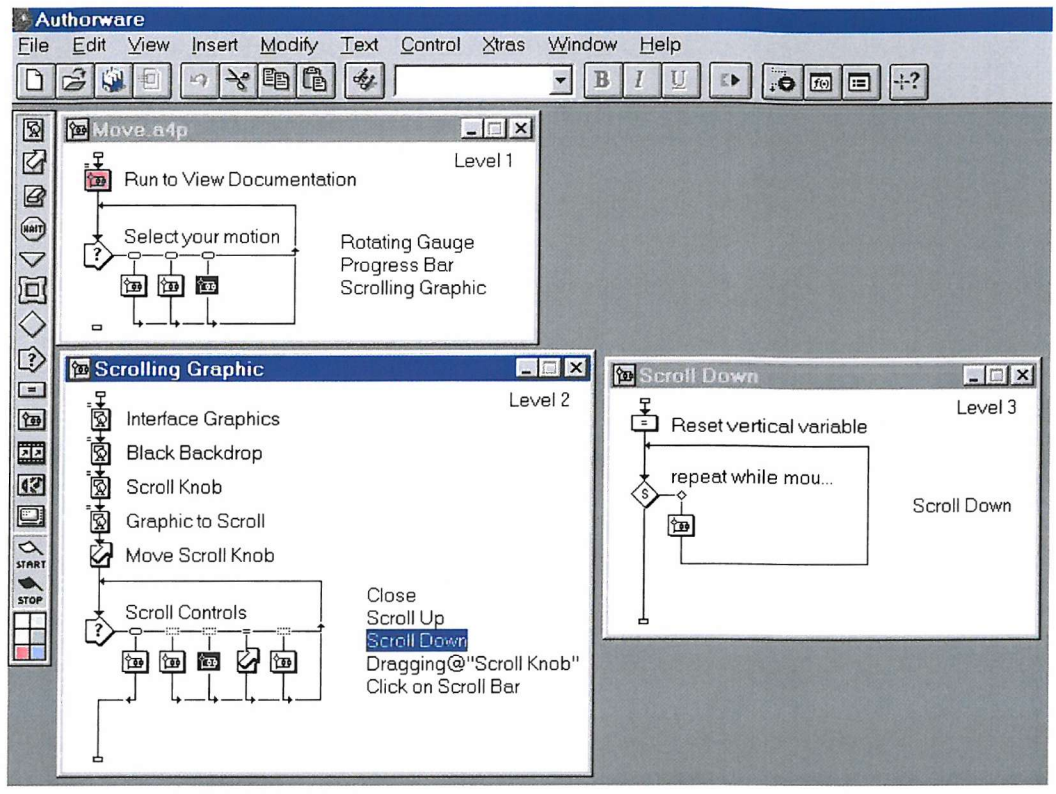

Figure 3.2: The design process for Macromedia's Authorware program.

3.3.3 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 and buttons. 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. to zoom text in and out and to cause a picture to flow onto the page. **Figure 3.3** 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 being just another entity or object used within the presentation. The majority of the time audio output is caused by an event, such as a button being pressed or the mouse cursor moving

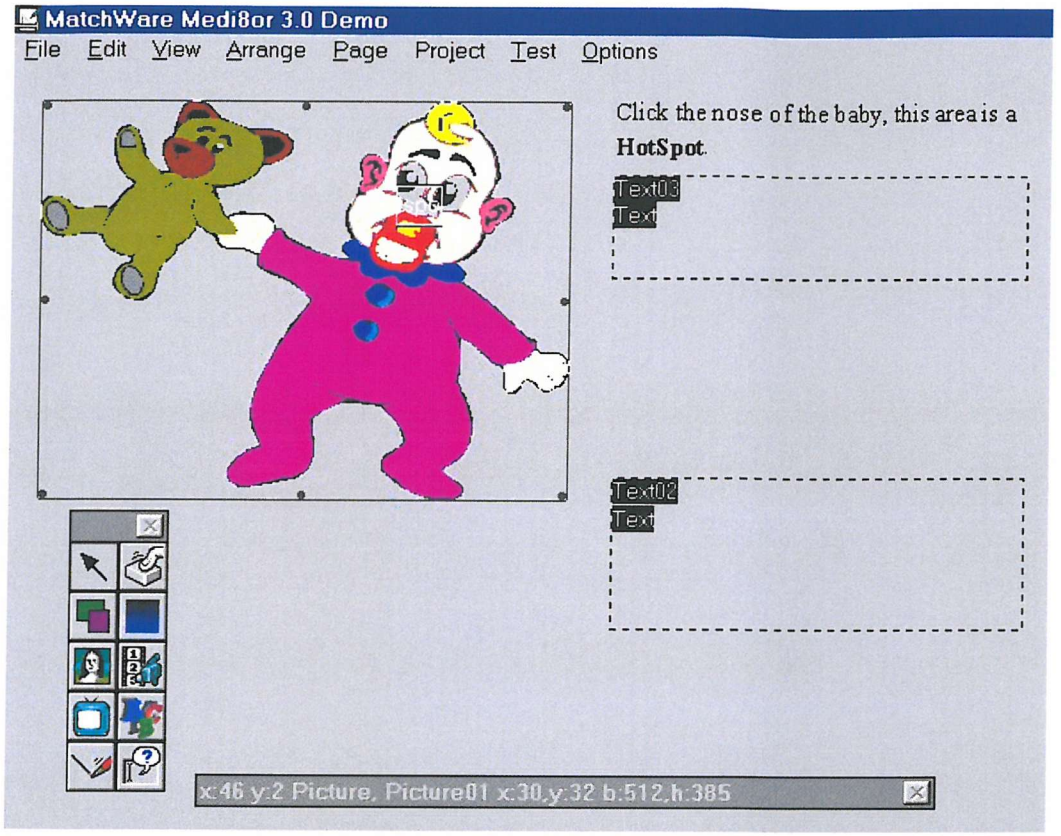

Figure 3.3: MatchWare's Medi8or design process.

over a hotspot.

3.4 Existing standards

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

3.4.1 The Moving Pictures Expert Group

The Moving Pictures Expert Group (MPEG)[1] was formed in 1988 by the ISO/IEC Joint Technical Committee (JTC 1) on Information Technology. Nack and Lindsay [45] describe MPEG as a working group (WG11 of SC29), with the mandate to:

[1] Also known as the Moving Picture coding Experts Group.

> "... develop international standards for the compression, decompression, processing and coded representation of moving pictures, audio and their combination."

There are five versions or *phases* of MPEG and these are described in more detail below.

MPEG-1 [62] is the first phase of MPEG and it became an international standard in 1993. Chiariglione [12] describes how it consists of five parts and they are:

1. *Systems* – which addresses the problems of combining one or more data streams, from the audio and video parts of the MPEG-1 standard, with timing information to form a single stream. Once formed, this single stream can then be used for digital storage or transmission.

2. *Video* – which addresses the algorithms required to compress video sequences into bitrates, that can be efficiently delivered over networks or played back from CD-ROMs. Originally CD-ROMs transferred information at about 1.2Mbps and the quality of a video at this bitrate, without sound, is equivalent to a VHS recording.

3. *Audio* – which addresses the algorithms required to compress audio sequences (both mono and stereo) into bitrates, that again can be efficiently delivered over networks or played back from CD-ROMS. One of the most popular audio formats currently used today is MPEG-1 layer 3 (MP3). MP3 files are relatively small in size and the quality of the encoded audio is comparable to compact discs. They can be easily stored on local machines, e.g. on CD-ROMs or hard-drives, or streamed over networks, see **Section 2.3.1**.

4. *Conformance Testing* – which addresses the design of tests to ensure that manufacturers comply with parts 1–3 of the MPEG-1 standard.

5. *Software simulation* – which is the initial software implementation of the first three parts of the MPEG-1 standard.

In his paper, Chiariglione [14] describes how MPEG-1 was the first integrated audio-visual coding standard ever produced. Traditionally, in industry and standard bodies alike, audio and video data were handled in separate departments with very little interaction between the two. With the development of this standard however, audio and video

data can be easily combined, into a form that can be stored or transmitted over the Internet.

The second version of MPEG is MPEG-2 [64] and papers by Nack and Lindsay [45] and Chiariglione [14] describe how it was designed to overcome the problems of the multiple standards used within the television broadcasting industry. Originally these standards used analogue signals for the transmission of the audio-visual information. With the development of MPEG-2 however digital techniques could be used instead, which improved the quality and performance of the television.

MPEG-2 became a standard in 1996 and originally consisted of nine parts. Chiariglione [13, 14], Thorn et al. [29], Nack and Lindsay[45] and Balabanian et al. [142] describe the different parts of the standard in reasonable detail. They are:

1. *Systems* – which extends MPEG-1 part 1 by creating two new types of stream, the *Program Stream* and the *Transport Stream.* The Program Stream combines one or more *Packetised Elementary Streams* (PESs)[2] with a common time base, into a single stream. It is designed for relatively error-free environments and consists of variable length stream packets. The Transport Stream however, uses one or more independent time bases and is designed for environments where errors are more likely to occur, e.g. networks. Transport Stream packets are 188 bytes long.

2. *Video* – which extends MPEG-1 by offering a wide variety of coding tools, higher resolutions, improved scalability and efficient interlacing.

3. *Audio* – which extends MPEG-1 by increasing the number of available audio channels from 2 (stereo) to 5 main channels and a low frequency enhancement (LFE) channel. Lower sampling rates and higher bitrates are also supported.

4. *Compliance Testing* – which corresponds to MPEG-1 conformance testing.

5. *Software Simulation* – which corresponds to MPEG-1 part 5.

6. *Extensions for Digital Storage Media Command and Control* (DSM-CC) – which is the specification for a set of protocols that provide a range of multimedia services over broadband networks. Clients and servers, in the DSM-CC network, are known

[2]A PES is a packetised version of an audio or video stream.

as *users* and there are two types of information transfer between these users; *user-to-user* (U-U) and *user-to-network* (U-N). U-U transfer is between the client and the server, whilst U-N transfer is between the network and the client or the server. User-to-network messages are exchanged via a U-N connection and are used to configure clients, control network resources and manage the sessions, created between users. User-to-user communication occurs via a separate connection and the information exchanged is used for stream control, file manipulation and directory access. A U-U connection is usually created for each of the resources, e.g. audio and video streams, used within a DSM-CC session. DSM-CC also contains a protocol to download a complete operating environment to the client, which results in low-cost client devices e.g. set-top boxes.

7. *Advanced Audio Coding* (AAC) – which uses a very high-quality audio coding algorithm with multichannel, multilingual and multiprogram capabilities. It can handle up to 48 channels with a wide variety of bit rates, from monophonic speech at 8 kbit/s to very high quality coding at 160kbits/s per channel. This algorithm encodes audio data into a form that is indistinguishable from the original source and it is not backwards compatible with MPEG-1 audio.

8. *Video Coding using 10 bit samples* – which was a technique for encoding video at 10 bits per input sample. However due to insufficient interest from industry, this part of the standard was discontinued.

9. *Extension for real time interface for systems decoders* – which is the specification of the Real-time Interface (RTI) to Transport Stream decoders.

MPEG-2 is being used by a number of organizations, including the *Digital Audio Visual Council* (DAVIC) and the *Digital Video Broadcasting* (DVB) Project, see **Section 2.4.1**, for the encoding and transmission of digital audio and video information. It is also being used to encode video information on *Digital Versatile Disks* (DVDs)[3].

MPEG-1 and MPEG-2 were developed to provide efficient algorithms and tools for the storage and transmission of high quality digital audio and video data. They are Emmy award-winning standards and they have been used in several different systems, from digital storage on CD-ROMs to digital television. In 1993, MPEG realised that these two standards alone would not be sufficient for the creation, manipulation and distribution of

[3]These are also known as *Digital Video Disks*.

multimedia content. Sikora and Chiariglione [127] explain that the MPEG group also realised that the telecommunication, computer and TV/film industries were rapidly converging and that a common core standard would have to be developed, to overcome the multiple standards used within these industries. With these requirements in mind, the group started to develop MPEG-4 which is the third phase of the MPEG standard.

Koenen [86] explains how MPEG-4 became an international standard in April 1999 (ISO/IEC 14496) and that it focuses on multimedia production, content-based access, user interaction and distribution. MPEG however, are currently extending this standard to support more advanced tools. MPEG-4 Version 2 will have multi-user interaction, a file format for storing content (MP4s) and a programming system using the Java language (MPEG-J).

Chiariglione [14] and Koenen [85, 86] describe how MPEG-4 introduces a new paradigm for the representation of audio and video data. In the two previous standards, this data is represented by tightly packaged "bitstreams", which are created by encoding audio and video signals from internal and/or external devices, e.g. soundcards, microphones and video recorders. MPEG-4 however, uses *Audio-Video Objects* (AVOs)[4] to encapsulate the audio and video data. These objects can be independent entities or they can be combined to form compound media objects.

In MPEG-4, *Video Objects* (VOs) are used to represent atomic units of visual information, e.g. an image or a video. There are two main types of VO and they are:

1. *Natural* video objects – which use advanced tools to efficiently create and manipulate images and video of arbitrary shape and size. The algorithms from the previous standards can also be used, to create natural VOs in their more traditional rectangular form.

2. *Synthetic* video objects – which are computer generated and can be used to create and animate realistic 2D objects. Koenen [85, 86] and Battista et al. [115] describe how MPEG-4 Version 1 provides tools to create 2D *meshes*, which are formed from partitioning 2D regions or images into triangular polygonal patches. These meshes can be static objects or they can be easily animated by combining them with geometry and motion information.

[4]These are also known as *Media Objects*.

It is possible to create *hybrid* visual objects, by combining both natural and synthetic VOs together. For example mapping an image of a flag onto an animated rippling mesh, will give the impression of the flag moving in the wind. This simple animation technique could be used to provide a more efficient method of transmitting moving images over the Internet. Only the image, the mesh and the direction of motion needs to be downloaded, which is considerably more efficient than downloading a video or the individual images. Videos and textures can also be mapped onto animated meshes, providing the ability to create more complex hybrid objects.

Originally the Moving Pictures Expert Group concentrated primarily on the visual domain; audio was regarded as being just another signal to be associated with video. With the development of MPEG-4, the group realised that audio was just as important as video and so they developed more advanced audio tools. Scheirer et al. [33], Koenen [85], Battista et al. [115] and Scheirer [126] describe how these tools can be used to create natural and synthetic *Audio Objects* (AOs).

Natural AOs in MPEG-4, are created by three audio coding tools; two for speech and one for *General Audio* (GA). The speech tools use advanced algorithms for the coding of toll-quality speech at 6kb/s to high-quality, highly compressed speech at 24kb/s. The GA coder is an enhanced version of the MPEG-2 advanced audio coding (AAC) algorithm and creates audio objects of very high quality.

To create synthetic audio objects, two types of audio coding tools are used; the *Text-To-Speech* (TTS) interface and the *Structured Audio* (SA) tool. Thorn et al. [29] and Scheirer et al. [34] describe how TTS systems convert text input to speech using a sequence of *phonetic symbols* and *prosody*. A phonetic symbol or *phoneme* corresponds to one sound in human speech and prosody is used to add human characteristics, e.g. pitch, duration and timbre, to a phoneme sequence. The MPEG-4 TTS tool provides this functionality and it also supports various languages, as well as the International Phonetic Alphabet (IPA).

The Structured Audio tool is used to describe different methods of musical synthesis. It consists of two languages that create and control a musical score and they are:

1. The *Structured Audio Orchestra Language* (SAOL) – which is derived from the "*Music-V*" family of computer music languages. These languages create audio waveforms through functional construction or through dynamically reading sampled sounds, stored in *wavetables*. These sampled sounds can be combined or

used directly to generate the actual sounds. SAOL defines an orchestra of musical instruments and then downloads them to the user's terminal. The terminal will then generate the sounds based on the definitions of the instruments. Using the *Structured Audio Sound Bank Format* (SASBF) additional wavetables can also be downloaded. This will ensure that the synthesized sounds will be identical on each terminal.

2. The *Structured Audio Score Language* (SASL) – which uses a set of time-sequenced commands, to invoke various instruments at specific times. As a result, each instrument contributes to the overall composition of the music.

The Structured Audio tool also supports the popular *Musical Instrument Digital Interface* (MIDI) standard, as an alternative for the delivery of musical scores. MIDI can be described as a protocol that provides a method of communication between musical devices and the computers that control them. The control "stream" from the computers can be easily packaged into MIDI files, which provide an easy way to exchange musical content. These files can be used on terminals that contain soundcards or other MIDI devices. By supporting MIDI, sound designers can easily create new SAOL orchestras for their existing MIDI files. For more sophisticated content and tighter control, composers must use the structured audio score language.

Koenen [85, 86] describes how multimedia applications can be created in MPEG-4 by combining audio, video and compound media objects with spatial and temporal information. This is called *composition* and the result is an audiovisual "*scene*". Users can interact with these scenes and they can also be modified, possibly as a result of the interaction.

To describe and dynamically change scenes, MPEG-4 defined a new language for scene description; the *BInary Format for Scenes* (BIFS). Originally the MPEG group was going to use the Web3D Consortium's *Virtual Reality Modeling Language* (VRML) [71], which allows users to describe and interact with 3-D objects over the World Wide Web (WWW), see **Section 2.3.1**. VRML creates 3-D scenes using a text-based language, that must be downloaded in full before it can be used to render[5] a scene. This however, is not an efficient method of transmission and so the group developed BIFS to overcome this problem. BIFS is used to encode scene descriptions into a binary form which is easier

[5]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.

to compress, transmit and execute. It can also be used for real-time streaming, so that scenes can be created and animated on the fly.

BIFS provides the functionality to modify and then position video objects in 2-D or 3-D space. Modifications include rotation and scaling transformations, as well as changing the properties of a VO such as its colour. Audio objects however, are more complex than VOs and require more advanced composition techniques. Scheirer et al. [33, 126] explain how the *AudioBIFS* part of the BIFS standard uses these techniques to control the composition of "*sound*" scenes. These scenes can then be used with other visual scenes to form a complete multimedia application.

AudioBIFS supports two main "modes of operation" and they are known as:

1. *Virtual Reality* compositing – which uses 3-D spatialization techniques to recreate, as accurately as possible, particular acoustic environments. For example, sounds can be positioned in a scene to give the impression that they are coming from behind the listener. Other techniques can also be used to give the impression of sounds moving around within a scene.

2. *Abstract-Effects* compositing – which provides authors with a rich set of sound processing tools, to create the right effects for a particular environment. For example, adding specific effects such as air absorption, propagation and reverberation to a piece of music, can give the impression that the listener is sitting in front of an orchestra, in a particular type of hall.

Audio-video objects are only defined once in BIFS and any changes made to the scene, for example a change in the viewing angle, a change in the acoustic properties of an object, are carried out locally at the user's terminal. This allows users to modify a scene reasonably quickly without having to wait for extra information to be downloaded first, over the network.

Koenen [85, 86] describes how MPEG-4 has a new paradigm for the preparation of multimedia content for transmission or storage. This involves creating separate *Elementary Streams* (ESs) for:

1. *Media Objects* – which usually have only one ES, for a particular type of object in a scene. Other scalable objects however will have several; one for basic-quality information plus one or more ESs for different types of enhancements, e.g. higher

quality audio or video with finer detail. When an object has multiple ESs, the one that is actually used depends on the Quality of Service (QoS) and the available network bandwidth.

2. *Scene Descriptions* – which ensures that the scenes can be modified, without having to change the objects and that the objects can be easily reused in other scenes. Again, multiple ESs might be used for a single description, depending upon the bandwidth and the QoS.

3. *Object Descriptors* (ODs) – which consist of pointers to elementary streams. These are used to inform encoding or decoding systems as to which elementary streams belong to a certain object.

Once the elementary streams have been formed, they are split into smaller "*packets*" ready for transmission or storage. Timing information is then added to the payload of these packets, so that multimedia applications can easily synchronize and buffer the compressed media.

The *Delivery Multimedia Integration Framework* (DMIF) protocol is then used to send the packetised streams to the remote application. Huard et al. [81], Koenen [86] and Chiariglione [90] describe DMIF as a protocol framework that uses existing protocol standards to manage the delivery of multimedia streams. Existing protocols from the Internet, satellite and cable industries are used for transmission, whilst CD and DVD technology is used for storage. DMIF however, was designed to separate the development of multimedia applications from the delivery details. This allows users to develop a single multimedia application for different types of "delivery technology" and it will also ensure that applications will not need to be modified if any new or enhanced protocols are developed.

DMIF defines two interfaces for this "layer of abstraction" and they are the *DMIF Application Interface* (DAI) and the *DMIF Network Interface* (DNI). DAI provides a common interface between the multimedia software and the delivery mechanism. It also handles the QoS requirements. The DNI handles the communication between the DMIF peers and the network itself, by mapping its own semantics to the relevant protocol standard, used on the network.

MPEG-4 is the first international standard to provide a set of advanced tools for the creation and delivery of digital multimedia content. It is designed to replace the proprietary formats already in use for audio, video and multimedia. It can also use existing protocol standards for the delivery of content over the Internet and cable. Authors can easily create interactive scenes with high-quality audio and video and then deliver it to users, who can interact with the scene in real-time. MPEG-4 however, has only just become an international standard and only a few encoding and decoding systems actually exist. Koenen [85] describes how audio broadcasters and mobile equipment manufacturers have expressed an interest in the technology, but very few have actually implemented or used the standard.

With the development of the Internet, the World Wide Web and digital techniques for the creation, manipulation and storage of audio-visual information, it has become relatively easy to develop and exchange multimedia content. Traditionally users would create this information, e.g. videos, speech, 3-D models and still pictures, for specific applications. However more advanced processing systems, such as image recognition systems for surveillance, have also been created which automatically generate and exchange AV material in real-time. As a result, the amount of audio-visual information is steadily increasing, which makes it much more difficult for users to identify, find, retrieve and manage this AV data efficiently. Several tools already exist that allow users to manipulate text in this way, for example Web-based search engines such as Alta Vista, Excite and Google. However very few tools exist[6] that allow this type of manipulation on audio-visual content.

To overcome this problem, the MPEG Requirements Group [113, 114] in October 1996 decided to develop MPEG-7; the fourth phase of the MPEG standard. It is formally known as the "*Multimedia Content Description Interface*" (MCDI) and it provides a set of standardized tools for the description of multimedia content. These tools can be used to generate descriptions in real-time, whilst the content is being captured, or after. The descriptions and their associated content can be located within the same data stream or on the same storage device. MPEG-7 can also store the descriptions separately, which ensures that the relevant material can be easily retrieved, exchanged and re-used. When kept separately, tools can be used to form bi-directional links between the AV data and its description. Whichever method is used for storage, humans as well as machines will

[6]Specialised tools do exist to find and retrieve AV data in professional databases and digital libraries, although they are usually proprietary solutions.

be able to use the description and its content, to efficiently search for related material. MPEG-7 will become an international standard in September 2001.

Nack and Lindsay [46] and the MPEG Requirements Group [113, 114] describe how MPEG-7 will generate descriptions, by identifying the components of the audio-visual content. This is achieved by defining:

- A set of *Descriptors* (Ds) – which can be used to describe the different types of audio-visual information regardless of storage, transmission, coding and display. Descriptors define the syntax and semantics of *Features*, which represent the distinctive characteristics of the AV information. For example a feature could be the title of a movie, whilst the descriptor for this feature would be the text of the title. A feature can also have multiple descriptors, to represent different requirements e.g. a movie feature could have descriptors for motion and colour information.

- A set of *Description Schemes* (DSs) – which specify the structure and semantics of the relationships between pre-defined descriptors and possibly other description schemes. An example of a DS is a movie, consisting of several scene structures and textual descriptors for those scenes.

- A language called the *Description Definition Language* (DDL) – which can be used to create new description schemes and descriptors, as well as modify and extend existing descriptors and DSs. This language can also be used to express the relationships between the DSs and among the individual components of the scheme itself. The DDL is still under development and it will use a mixture of the *Extensible Markup Language* (XML) syntax, see **Section 3.5.5** and MPEG-7 terminology.

- One or more methods for *encoding* descriptors – which will be used for efficient compression, storage or transmission and fast access and retrieval.

The MCDI is still under development and MPEG is currently examining the different proposals, that have been contributed by the participating members of the standard. Nack and Lindsay [46] and the MPEG Requirements Group [114] describe these proposals[7] and they include tools and descriptors for audio and visual information and multimedia description schemes. Audio descriptors will describe instrument definitions, speech

[7]The details of these proposals have not been finalised and therefore they could change.

recognition and sound effects, whilst visual descriptors will handle motion activity, textures and shapes. The multimedia DSs will handle different types of descriptors and information, including the physical structures of an AV document or image, the descriptors that carry author-generated information and the descriptors that are specific to the storage media.

MPEG-7 will provide the tools to effectively describe the different types of audio-visual content used today, in multimedia applications. Nack and Lindsay [45] describe how this standard will eventually be used, for information retrieval and interaction, in various professional and consumer oriented applications. These include education, journalism, biomedical research and digital libraries.

Work on the fifth phase of the MPEG standard, MPEG-21, has only just begun. It will be used to created multimedia frameworks and at the moment the group are gathering information on this area.

3.4.2 Digital Audio Broadcasting

The *Digital Audio Broadcasting* (DAB) system was developed by the EUREKA-147 Project, which is a European consortium of several organizations from the broadcasting, networking and telecommunication domains. The project was established in 1987, to develop a digital radio system that could deliver high-quality digital audio programmes and data services to mobile, portable and fixed receivers. In 1997 the *European Broadcasting Union* (EBU), the *European Committee for Electrotechnical Standardization* (CENELEC) and the *European Telecommunications Standards Institute* (ETSI)[8], released DAB as a European Standard [35] and it has also been adopted by the *International Telecommunication Union* (ITU) as an International Recommendation [74, 75].

The WorldDAB Forum [149] explains that the system itself can be used with existing analogue Frequency Modulation (FM) and Amplitude Modulation (AM) radio broadcasting systems. Analogue receivers, however suffer from interference caused by the surrounding environment, e.g. hills, trees and buildings. Signals from analogue transmitters "bounce" of these objects and can be received at totally different times, distorting the main signal. This is known as *multipath* interference. DAB receivers are intelligent and use advanced digital techniques to sort through the multiple signals received, so that the

[8]More formally known as the EBU/CENELEC/ETSI Joint Technical Committee - Broadcast.

main signal which the user hears, is interference free. As a result these receivers produce CD-quality sound and eventually they will replace existing FM and AM radio receivers.

The EBU/CENELEC/ETSI JTC [35] and the EUREKA-147 Project [40] describe how the DAB transmission signal consists of several digital services multiplexed together, to form an *ensemble*. Multiplexing is the process of combining several signals into one for easier transmission. Once received, this signal is then passed through a de-multiplexer or *demux* to retrieve the individual signals. The overall bit-rate of an ensemble is about 1.5 Mb/s and it contains audio programmes, data related to these programmes and optionally other data services. Receivers usually decode several of these services simultaneously and to ensure that the multiplexed signal is decoded properly, DAB defines:

- *Flexible audio bit-rates* – from 8 kb/s to 384 kb/s using MPEG-1 Layer 2 and MPEG-2 Layer 2 audio compression techniques, see **Section 3.4.1**. This provides up to 6 high-quality stereo channels or up to 20 lower-quality mono channels.

- *Data services* – for the streamed or packet-based delivery of the actual services. This is achieved by using either the *stream* or *packet* mode of transport, respectively. The stream mode transparently delivers the data from the source to the destination. The packet mode splits the data into smaller addressable blocks, which it then delivers. By adding the address of the destination to these packets, transmitters can use the same broadcasting channel to deliver different services.

- *Programme Associated Data (PAD)* – for data that is associated with the programme, e.g. lyrics or phone-in telephone numbers. It is usually synchronized with the audio and therefore, it is embedded into the audio bit-stream. The use of PAD is optional.

- *Service Information (SI)* – for the control and operation of receivers. This provides information about the audio programmes and the data, including *programme service labels* e.g. the name of the station, *programme types* e.g. news, sports, classical, *programme language* based on the country and the ability to switch to traffic reports, news flashes and announcements on other ensembles, including FM and AM radio. To select a radio station, with traditional analogue systems, users have to either remember the frequency of the desired station or search the frequency range, until a strong enough signal is found. With DAB receivers however, users

select a station from a text-based menu of programme service labels. This menu is displayed on a small screen on the receiver.

- *Conditional Access (CA)* – for individual services or packets of data, if packet mode transmission is required. This is similar to "pay-per-view" systems, that allow users to view a movie or a specific TV channel, once they have subscribed to the service provider e.g. a TV broadcaster. With CA, services are scrambled so that unauthorised users can not listen to them until they have subscribed. Once subscribed, users receive (from the service provider) encrypted codes that can be used to de-scramble the service.

DAB also provides an advanced transmission protocol and interface, for the delivery and decoding of radio programmes containing other types of media; for example text, graphics and pictures. The EUREKA-147 Project [40] describes how the system uses the *Multimedia Object Transfer* (MOT) [38] protocol for the streamed or packet-based transmission of "multimedia objects" and the *Receiver Data Interface* (RDI) [37] for the interface. Each multimedia object consists of a file containing the data, such as the data for an image and extra information for object classification and presentation. These objects can be created using MHEG, see **Section 2.4.1**, or Java. The MOT protocol supports several formats for the data files including ASCII text, HTML see **Section 2.3.1** and MPEG-1, 2 and 4, see **Section 3.4.1**. The RDI is used to connect DAB receivers to dedicated decoders, e.g. a computer, so that the multimedia information can be decoded, processed and then displayed. The transfer rate between the receiver and the decoder is about 1.8 Mb/s.

Digital radio programmes, that have been supplemented with multimedia information, will be used to provide enhanced data services for listeners. This *Multimedia Radio* will include:

- *Traffic and Travel Information* (TTI) – which will provide up-to-date information about traffic problems, e.g. congestion, traffic navigation and route planning using digitised maps and general travel information, e.g. restaurant locations, hotel vacancies, room prices and pictures of rooms. TTI services will be language-independent and will allow international roaming, using portable or mobile receivers.

- *Picture Transmission* – which will be used for the transmission of still pictures. These pictures could be displayed in a slide-show, synchronized with the audio

programme, e.g. photographs of scenes or individuals during a news bulletin.

- *Text Transmission* – which will provide simple dynamic labels, limited to about 128 characters in length, for station names, programme types. This is similar to the text service supplied by the FM-*Radio Data System* (RDS) [36]. This system displays the same type of information, on simple alpha-numeric screens, over FM radio. A more sophisticated text service, which is known as the *Interactive Text Transmission System* (ITTS), could also be used. It can transmit textual information at different rates and it can process several streams of text simultaneously. ITTS supports different display formats, e.g. 12-character displays and colour displays.

- *TV Transmission* – which will be used to transmit digital television signals to portable and mobile receivers. Trial systems have shown that DAB is a suitable system for this. MPEG-1 and MPEG-2, see **Section 3.4.1**, will be used to compress and combine both audio and video information, into a DAB signal.

The EUREKA-147 Digital Audio Broadcasting system has gone through extensive trials in a number of countries, including Australia, the United Kingdom, Singapore and Canada. It has been adopted by the International Telecommunications Union as the de facto standard for digital radio and several countries are now using this system, to broadcast digital radio programmes and data services; from classical music and jazz, to news and weather information with pictures. Eventually DAB will replace traditional analogue systems.

3.5 "Open" 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 3.2** 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) and Sun's AU format. 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 several hypermedia systems and they are described in more detail below.

3.5.1 The SoundViewer Tool for Microcosm

Originally Microcosm, see **Section 2.3.2**, 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 and this tool is discussed in more detail in **Section 5.2**.

3.5.2 Mavis

MAVIS, see **Section 2.3.4**, 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. [105] 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 and images, related to the original audio sample.

3.5.3 The Harmony Audio Player for HyperWave

The HyperWave client consists of several viewers and Geiger [47] describes how the Unix version of this client, *Harmony*[9], has been extended to support digital audio. The tool is called the *Harmony Audio Player* and it can be used with HyperWave or it can be used as a standalone application. When used with HyperWave, the tool will be executed when a user selects an audio document from the session manager. This will create a connection to the server and download the audio file. The audio sample is then displayed in the viewer as a waveform, with a scrollbar representing the current position in the audio file; see **Section 5.2** for a description of each of these. The player contains the normal controls to playback, stop and pause the audio sample. Users can author links to other documents by selecting the sample or a portion of the sample. These areas will then be marked as links, which users can then follow by simply clicking on these areas.

3.5.4 Streaming Audio and 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 4.2**, WWW browsers are now being used to read and retrieve information off the Internet. As a result the WWW has steadily grown in size (due to the number of hosts and web pages) 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" or "video" link, which would make the browser download the file to the users' machine. Once downloaded some browsers, for example Netscape's Navigator, would automatically start an application to play the file. Older browsers would require the user to execute an external program. The process of downloading this file however, could take a long time especially if the file was large and / or the network connection was poor.

[9]The Window's version of this client is called Amadeus.

To overcome this problem several companies, such as RealNetworks, Microsoft and Xing Technology, have developed "*streaming media*" servers and players. The streaming servers, which are usually located on different machines on the Internet, store the actual audio and video information. The players are used to retrieve and play this information. A streaming media link is represented on a web page in exactly the same way as a normal link, however it will contain information about the server and the file to be streamed from the server. For example, if a user clicks upon a streaming audio link on a web page, three events occur and they are:

1. The browser activates the relevant streaming audio player e.g. RealNetwork's RealPlayer or Microsoft's MediaPlayer. 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 media. This protocol is called the *Transmission Control Protocol / Internet Protocol* (TCP/IP) and is described in more detail in **Section 4.2**. To overcome this problem several streaming 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 4.3** and **4.4**, respectively.

3.5.5 SMIL for the WWW

The Synchronized Multimedia Integration Language (SMIL)[10] was developed by the SYnchronized MultiMedia (SYMM) Working Group of the World Wide Web Consortium (W3C). This group was formed to develop a language that could combine independent media objects into a synchronized multimedia presentation, over the WWW. SMIL

[10]Pronounced "smile".

Version 1.0 [128] became a W3C Recommendation in June 1998 and it uses an *Extensible Markup Language* (XML) [152] DTD to define the syntax of SMIL documents. This section will describe SMIL and it will also give a brief description of XML.

The Hypertext Markup Language (HTML), see **Section 2.3.1**, is currently the de facto standard for document markup on the Web. It provides a set of generic markup tags, e.g. for headings, paragraphs and images, that can be used to define simple documents. Garshol [92] and Flynn [103] explain however, that this language only provides one way to describe information and that this information can not be described precisely. Different organizations require different types of markup, to describe their documents effectively. For example a chemical engineering firm will probably use specific markup tags, to describe chemical formulas in their documents. HTML was not designed to handle this type of information; it is a simple generic language. This limitation has caused many organizations to extend the language, with their own proprietary markup. In some cases this "new" markup will only work with a specific version of a specific browser, which defeats one of the main goals of HTML; to be an non-proprietary, interoperable language.

Ossenbruggen et al. [80] also explains that HTML has limited support for hypertext linking. HTML provides a very simple linking mechanism; links are created and stored within the documents themselves, using specific markup tags. These links are known as *embedded* links and **Section 2.2** describes the problems of using this type of link in hypertext systems. This section also describes Open Hypermedia Systems (OHSs), which have far more advanced tools for link creation, storage, manipulation and delivery.

Many developers and organizations, including the W3C, are aware of these problems and believe that HTML has reached the limits of its usefulness. In 1996 the W3C started work on a new standard, that would allow users to define their own markup for the Web. Initially they looked at SGML, see **Section 2.4.2**, the international standard for defining document markup. SGML however, is a complex language that is hard to implement and it also contains a lot of rarely used features. Eventually the W3C formed a new working group, the XML Working Group, to develop a new language for this standard and in February 1998 the Extensible Markup Language [152] became a W3C Recommendation.

Garshol [92] and Flynn [103] explain that XML is an abbreviated form of SGML, which makes it easier for users to define their own markup for different types of documents. Programmers will also find it easier to develop tools to process these documents. By themselves however, SGML and XML do not have any elements to define

links. With SGML, HyTime and other advanced standards can be used to add these elements to SGML DTDs, see **Section 2.4.2**. For XML two new languages, that are derived from HyTime and other similar standards, are being developed. These languages will be used in conjunction with XML to provide advanced linking capabilities and they are known as the *XML Pointer Language* (XPointer) [150] and the *XML Linking Language* (XLink) [151]. XPointer will be used to reference resources within XML documents, whilst XLink will be used to define the actual link elements themselves. These elements can then be used in XML documents to create and describe the links between resources.

Hoschka [57, 58] explains that in October 1996 the (W3C) also had a workshop on "Real-Time Multimedia and the Web". Representatives from different communities and organizations, including the streaming media and international standards communities, participated in this event. From the discussions and presentations during the workshop and the feedback received after the event, Hoschka observed that:

- The World Wide Web is rapidly becoming a multimedia environment, that can handle many different types of media.
- The Web is a relatively inexpensive medium for the distribution and updating of multimedia information.
- Many organizations and industry analysts believe that the Web will become a distribution system for synchronized multimedia content.
- Current Web technology can not effectively describe the spatial and temporal relationships between the different media elements, in a synchronized multimedia presentation. Complex scripting languages, e.g. Netscape's JavaScript and Microsoft's VBScript, or multimedia authoring tools, see **Section 3.3**, can be used to create these presentations. However, authoring tools use non-interoperable proprietary formats to store the data and script-based content is difficult to produce and maintain.
- Declarative languages and open standards are required for the creation and real-time delivery of continuous multimedia presentations. Declarative formats are easier to understand, edit and maintain. They are also more durable, so that a single company can not control or modify the format.

As a result of this meeting the W3C formed the SYMM Working Group, to develop the

Synchronized Multimedia Integration Language (SMIL) and in 1998 they released SMIL Version 1.0 [128].

Bulterman et al. [21], Hoschka [58], Hardman and Wilson [91] and the SYMM Working Group [128] describe how SMIL was designed to be a simple, easy-to-use declarative language. It effectively describes how different media objects can be combined, both spatially and temporally, into a multimedia presentation. SMIL consists of five main components or *elements* and they are:

1. The *Media Object Elements* – which are used to include media objects into a SMIL presentation. Each element has a name, which is a simple text description of the media and a *Uniform Resource Indicator* (URI), see **Section 2.3.1**. These names, e.g. "`audio`" for audio and "`img`" for images, are only used for readability, they are not used to define the type of the media. This information is either derived from the operating system itself or from other reliable sources. The URIs are used to locate the media on the Web and also describe how it is to be delivered, e.g. by using a streaming protocol or a text transfer protocol.

   Media elements also contain extra attributes to define their temporal characteristics. For example a user can explicitly define when an audio sample starts to play and its duration.

2. The *Spatial Layout Elements* – which are used to control the spatial layout of the media elements within a SMIL presentation. There are two main elements for this and they are the `root-layout` and the `region`. The `root-layout` element defines the size of the main window, or viewport, in which the entire presentation is visually rendered. The `region` element controls the position, size and scaling of the media elements, within this viewport. A presentation can have several regions, e.g. one for text, one for images and another for video.

3. The *Synchronization Elements* – which are used to synchronize the playback of the different media elements, in a presentation. The `seq` and the `par` elements are used for this and they both group together media elements, that can be played back in sequence or in parallel, respectively. These elements can also be nested together, to form more complex relationships.

4. The *Hyperlinking Elements* – which are used to create uni-directional links, to media elements within the same document and also separate documents. The

SMIL linking mechanism is very similar to the HTML linking mechanism, see **Section 2.3.1**. However SMIL also provides the `anchor` element, which is used to create links to and from spatial and temporal *subparts* of a visual media object. These subparts are usually rectangular regions or "hotspots" within an image or video. By clicking on these regions, users are taken to another part of the presentation.

5. The *Alternate Content Element* – which is used to select an "acceptable" element from a predefined set of alternate elements, based on the capabilities and settings of a user's system. Authors use the `switch` element to create this set and it can contain media, synchronization and hyperlinking elements. An element is "acceptable" if it conforms to the SMIL specification, its media type can be decoded and all of its *test attributes* evaluate to "true". These attributes are used to test the capabilities of a system and they can be added to any element within this set. They include tests for caption control, screen size, language settings, colour support and network bandwidth. If an element is "acceptable" it is selected. However an element is not selected, if any of its attributes evaluate to "false".

Presentations are created in SMIL by combining the elements, mentioned above, into SMIL *documents*. These documents thoroughly describe the layout, the temporal behaviour and the relationships between the media objects, within the presentation. Schmitz et al. [106] and Rein [112] explain however, that very few Web browsers have the functionality to process a SMIL document and then play the resulting presentation. To overcome this problem, several companies have developed new tools or modified existing programs, to provide this functionality. These programs, for example RealNetworks RealPlayer 7, Apple QuickTime 4.1 and Oratix Grins, are usually activated by the browser, whenever a user clicks on a link to a SMIL document. A screen-shot of a simple SMIL presentation, in RealPlayer 7, is shown in **Figure 3.4**.

This presentation contains five media elements; one for streaming audio, another for text and three more for images. Four region elements are also used to set the position and size of the visual objects, e.g. the text and the images. A brief example of the SMIL code, used for this presentation, is shown in **Table 3.1** and a complete listing of the code is in **Appendix A.1**.

The `root-layout` element has attributes to define the title, the background colour and the size, using the `width` and `height` attributes, of the main presentation window. It

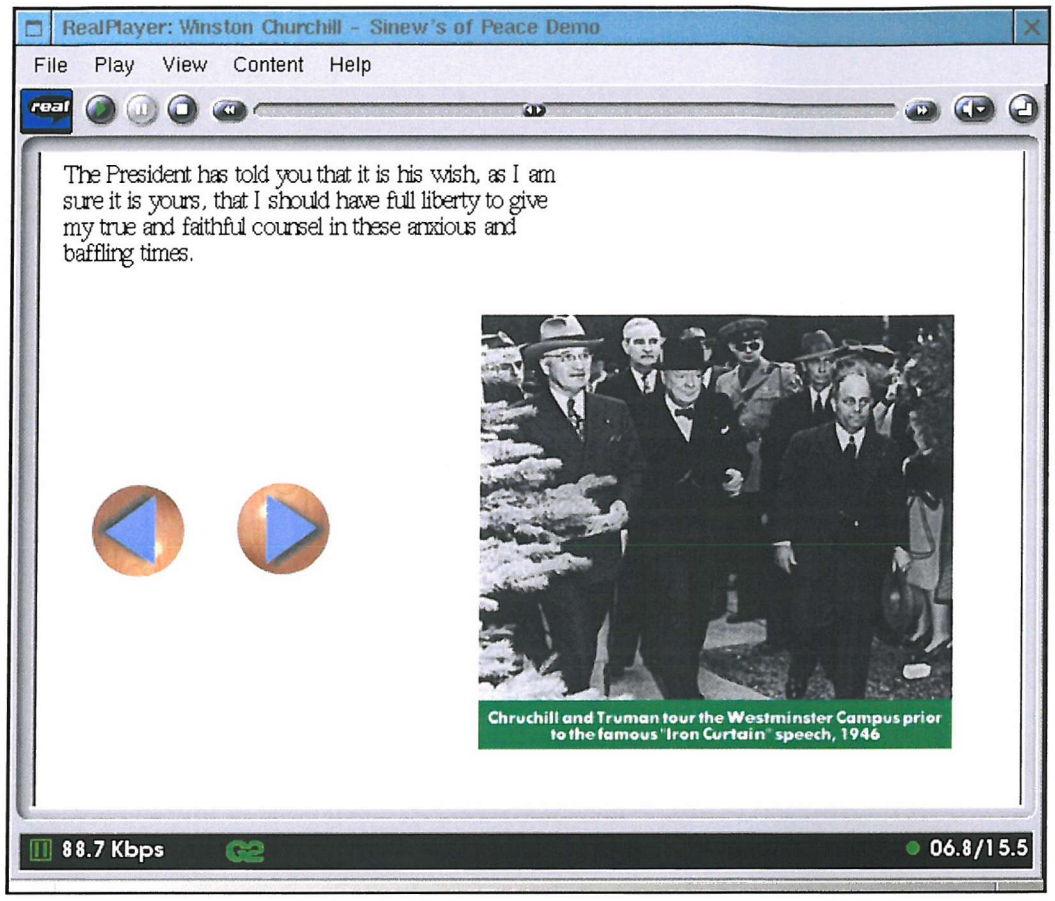

Figure 3.4: A screen-shot of a SMIL presentation in RealPlayer 7.

also has an unique identifier, using the `id` element. The `region` elements have attributes to define their position in the viewport, using the `top` and `left` attributes and their size. Again both regions have identifiers. The "Transcript" region also has a `z-index` attribute, which is used to determine if another region can be placed or "stacked" on top of another. A low value for this attribute ensures that other regions can not be stacked on top of this one.

The `audio`, `textstream` and `ref` elements are media elements and they are played in parallel, using the `par` synchronization element. This element also has an identifier, which allows other elements to link or "jump" to this part of the presentation. These media elements have attributes to define the location of the actual media, using the `src` attribute and they also define the explicit duration of this media, using the `dur` attribute. The `textstream` and the `ref` media objects will be displayed in the "Transcript" and the "LeftArrow" regions of the viewport, respectively. Finally, the `ref` media element

```
<root-layout id="Main-Window" background-color="white"
             title="Winston Churchill - Sinew's of Peace"
             width="600" height="400"/>
<region id="Transcript" left="2%" top="2%" width="52%"
        height="40%" z-index="1"/>
<region id="LeftArrow" left="5%" top="50%" width="62"
        height="63"/>
...
<par id="part2">
 <audio src="sinew02.rm" dur="15.6s"/>
 <textstream id="text2" region="Transcript"
             src="sinew02.rt" dur="15.6s"/>
 ...
 <ref region="LeftArrow" src="purpwood_left.rp"
      dur="15.6s">
  <anchor href="#part1" dur="15.6s"/>
 </ref>
 ...
</par>
...
```

Table 3.1: A sample of the Sinew's of Peace SMIL presentation

defines itself to be a link, using the anchor element, to another part of the presentation. It uses the href attribute to define the destination of the link; in this case "part1". If users click on this media element, they will be taken back to the first part of this presentation.

The Synchronized Multimedia Integration Language is steadily growing in popularity and an increasing number of tools are being developed to help create and play SMIL presentations. Schmitz et al. [106] and Rein [112] explain however, that the language itself has several limitations. As mentioned previously, most Web browsers can only process and render HTML documents; they do not have the functionality to process and then play SMIL documents. Separate programs provide this extra functionality and as a consequence, several organizations[11] describe SMIL as being a stand-alone data interchange format, for multimedia authoring tools and players.

Hoschka [59] and Rutledge et al. [95, 96] describe how authors have found some of the encoding mechanisms, in SMIL Version 1.0, to be inefficient and cumbersome. For example, no mechanisms exist for the relative positioning of region elements; authors have to explicitly define the position of each region element in the viewport window. If

[11] Microsoft, Compaq and Macromedia.

there are many regions in a presentation, this process can be very tedious and the resulting code very inefficient. Authors have also found the code to be too repetitive, e.g. when a user clicks on a link to another part of the presentation. In some situations, the authors only want a small part of the presentation to change; the rest stays the same. To achieve this in SMIL, authors have to repeat the code for the static part of the presentation, which can result in excessively large programs.

Since the release of SMIL Version 1.0 as a W3C Recommendation, the SYMM Working Group has received numerous proposals and contributions, from individuals and organizations, to improve and extend the language. Hoschka [59] describes these extensions and improvements in more detail and they include:

- A proposal to extend and enhance HTML documents and Web browsers with SMIL functionality. This proposal was called the *Timed Interactive Multimedia Extensions for HTML* (HTML+TIME) and it was defined by Schmitz et al. [106] in 1998. HTML+TIME consists of a set of extensions which are derived from the elements used in SMIL. These extensions have a similar syntax to the SMIL elements and they would be used to add timing, interaction, synchronization and delivery capabilities to HTML.

- A proposal to create a *Document Object Model* (DOM) for SMIL. This proposal was originally defined in the HTML+TIME specification. A DOM [31, 32] provides a standard set of objects for representing HTML and XML documents, a model to describe the relationships between these objects and an *Application Programming Interface* (API), which allows programs and scripts to dynamically access and modify these documents. This interface is language and platform neutral and it can be used to access and change the structure, the content and the style of a document. The DOM for SMIL would allow programs or scripts to easily manipulate synchronized multimedia documents. For example, a program could be used to play a media object after a certain amount of time or when a user clicks on a particular part of the document.

- A proposal to extend timing and synchronization functionality. This proposal would allow authors to select the type of synchronization to use, e.g. hard or soft, and it would provide more control over the synchronization behaviour. If a presentation has two streams, for example an audio and video stream and the video is delayed by the network, hard synchronization will either drop parts of the video

stream or pause the audio until video delivery resumes. Soft synchronization, in this situation, will just continue to play the audio. This proposal also provides several new synchronization elements and time attributes. These new elements could be used, in SMIL documents, to reduce the amount of repetitive code and the attributes would assist in the timing and synchronization of media elements.

- A proposal to extend the hyperlinking functionality. This proposal would allow authors to create both inline and out-of-line links and reference resources within SMIL documents, using the XLink and XPointer recommendations respectively. Originally SMIL Version 1.0 was going to use XLink. Before SMIL became a W3C Recommendation however, all references to XLink had to be removed because it was a working draft, subject to change. The SYMM Working Group modelled the hyperlinking mechanism after HTML instead. This proposal would also allow authors to define the shape of the "hot-spot" used for the anchor element, e.g. a circle or polygon instead of a rectangle, and which parts of a document are activated when a user clicks on a link. This activation functionality would also reduce repetitive code.

- A proposal to extend the layout and presentation functionality. This proposal would extend the original layout language, used in SMIL Version 1.0, to support more of the properties and values used in the *Cascading Style Sheets level 2* (CSS2) [145] specification. CSS2 is known as a *style sheet* language and it is used to attach different styles, e.g. fonts and colours, to structured documents, such as HTML and XML. The layout language, used in SMIL Version 1.0, is actually derived from a small subset of CSS2 and therefore, it can be easily replaced with a CSS2 style sheet. This proposal would allow authors to more effectively describe the layout and the visual and acoustic properties of the media objects used within a SMIL presentation. For example, authors would be able to change the colours of anchors, centre media objects in regions, position audio is space and control the audio volume.

The W3C has also received several other contributions, which would be used to control the delivery of media objects, to create transition effects, to improve accessibility and to extend internationalisation support.

In February 1999, the SYMM Working Group decided to combine all of these proposals and contributions into a new version of SMIL, code named "*Boston*" [129]. This

new version is currently under development and it will be fully backwards compatible with SMIL Version 1.0. The syntax of SMIL "Boston" documents will be defined, again, using a XML DTD and it will be specifically designed to be reused in other XML-based languages, that require timing and synchronization support. The language itself will consist of semantically related *modules*, e.g. a linking module, a media object module, that can be combined to form *profiles*. These profiles will be used to solve particular problems, e.g. to handle simple presentations and to integrate SMIL functionality into HTML. SMIL Version 2.0 is currently a W3C Public Working Draft.

3.6 Analysis

The visual domain has always been regarded as the primary sense for normally sighted people and this has been shown in the development of interactive computer systems. The majority of these systems are usually completely visual in nature and as the complexity of these systems increase, more and more visual information will be displayed on screens. This will eventually overwhelm the users, making it harder for them to understand the actual information being presented.

Humans interact with the outside world using a mixture of the five senses; sight, hearing, touch, taste and smell. Modern operating systems (OSs) currently support the visual and the audio domain, although audio is primarily used for warnings, alarms and status information. Recent developments in soundcard technology however, have improved the audio capabilities of these systems. It is now possible to "place" the audio in 3-D space; for example to the left or behind the user. This is achieved by applying specific effects to the audio information. The human ear is designed to receive this information but it can also identify the sound's location, by the length of time taken for the sound wave to reach both ears. This type of effect is mainly used in computer games and it has been shown that users will score more points in a game with the sound turned-on, rather than off.

The audio domain has been used in many different ways, by multimedia authoring companies, standards' organisations and "open" hypermedia research groups. Multimedia authoring companies have developed tools, that can be used to combine different types of media into a single application. These tools also allow the distribution of these applications over a variety of media, including the Internet and CD-ROMS.

There are three main types of multimedia authoring environment; the Timeline and the Flowchart authoring methods and the "Book" metaphor. The Timeline method uses

layers to create a multimedia presentation, with each layer representing different types of media. The Flowchart method uses icons to represent the data, particular events and routines such as loops. The "Book" metaphor presents the user with blank pages, in which multimedia objects can be placed. Each environment allows the user to place audio at a specific point within the presentation. However audio is just another part of the presentation to be played back. It is not possible to create a link to and from an audio layer or icon. The "book" metaphor allows users to place objects within a page, that appear or do something during the audio playback. This audio however, is not controlling these objects, the underlying timing mechanism in the "page" is used to do this. Therefore the audio objects, in the "book" metaphor, are just another entity to be played back.

The standards' organisations have developed several standards to handle audio information and they include MPEG and Digital Audio Broadcasting (DAB). MPEG is an extensive standard, developed by ISO and it currently consists of five main phases; MPEG-1, MPEG-2, MPEG-4, MPEG-7 and MPEG-21. MPEG-1 and MPEG-2 were developed to provide algorithms and tools for the transmission and storage of high quality digital audio and video data. They have been used in CD-ROMS, Digital Video Disks (DVDs) and digital television. MPEG-2 also has an Advanced Audio Coding (AAC) algorithm, with support for upto 48 channels. This format encodes data into a form which is indistinguishable from the original source. These two formats however, were mainly designed to handle the algorithms and tools to encode audio and video information. MPEG-2 can use DSM-CC to create multimedia services over broadband networks. However it was realised that these two standards alone would not be sufficent to handle multimedia content.

MPEG-4 was specifically designed to provide an advanced set of tools for the creation and delivery of multimedia content. MPEG-1 and MPEG-2 originally encoded the audio and video data into tightly packaged bitstreams. MPEG-4 however, encodes this data into Audio-Video Objects (AVOs), which can be independent entities or combined to form compound media objects. A considerable amount of work has been carried out on the audio domain, including the Text-To-Speech (TTS) interface and Structured Audio (SA), for the delivery and playback of musical scores. MPEG-4 can create multimedia applications by using scene descriptions, which use AVOs and the BInary Format for Scenes (BIFS), which translates the scenes into a binary format, for easier transmission and execution. The AudioBIFS tool is used to apply specific algorithms, to the audio information, so that it can be placed in 3-D space. MPEG-4 however, has only just become an international standard and at the moment, an interest has been shown in this

standard but very few implementations actually exist.

MPEG-7 and MPEG-21 are new phases of the MPEG process and they are currently under development. MPEG-7 has been designed to provide a set of standardised tools for the description of multimedia content. These tools will help overcome the problems of finding, identifying and retrieving the large amounts of available audio-visual material. MPEG-21 will be used to create multimedia frameworks and at the moment however, the MPEG group is obtaining more information on how this can be done.

DAB is a digital radio system that delivers high-quality audio programmes and data services to mobile, portable and fixed receivers. It was designed to eventually replace the analogue radio systems used throughout Britain and Europe. Other countries including Canada and Singapore, are also considering this standard. A DAB transmission consists of several services multiplexed together to form an ensemble. These services provide information for the control and operation of the receivers, on the programme itself and on conditional access, which is similar to "pay-per-view" systems. DAB also uses the Multimedia Object Transfer (MOT) protocol, for the streamed or packet-based delivery of multimedia objects and the Receiver Data Interface (RDI) for the interface. Programmes that have been extended with MOT will include traffic and travel information, picture and text transmission.

DAB is a relatively new medium for the delivery of audio and only a few digital radio stations, that support MOT and RDI, actually exist within Britain. Currently digital receivers are far more expensive than their analogue counterparts, which has resulted in a slow uptake of the technology. Programme Associated Data (PAD) can be considered a form of hypertext because the data is usually synchronized with the audio. To achieve this however, the data is usually embedded within the audio stream. On the receiver, this information will be displayed at the relevant time. Users will be able to scroll through this information, but they will not be able to request more information on a particular item.

Radio by default is a unidirectional system, in which radio signals are constantly transmitted and received on either a mobile or fixed device; there is no return channel. The consortium that developed DAB however, are also developing a return channel to be used with interactive receivers. This will allow users to select and retrieve relevant information.

Traditionally hypermedia systems have neglected the audio domain. This was due to

a lack of technology, the complexity of the audio itself and the problem of visualising audio information. Several "Open" Hypermedia Systems (OHSs) however, have developed audio tools and these are Microcosm's SoundViewer tool, the Sound FT module for Mavis and the Harmony Audio Player for HyperWave.

The SoundViewer tool and the Harmony Audio Player both use a scrollbar to represent the current position within an audio file and the length of this scrollbar represents the duration. Both tools can be used as standalone applications, however when they are combined with their respective OHSs, they allow users to create links to and from the audio domain. These links are displayed within the tools and users can follow them, by simply clicking on the link. With Harmony however when a user selects an audio document, the audio file is downloaded, from a HyperWave server, to the client's machine. The SoundViewer tool can use the audio files stored on the same computer that is running Microcosm and it can also share the audio files on another computer, that is connected by a local network. However if there is a problem with the network and / or the computer, then these shared files will not be accessible.

The Sound FT module uses a Fourier Transform on a selected portion of a digital sound, such as speech. This sample can then be used as a key for fuzzy matching. However this module and the Harmony Audio Player both used audio waveforms, to display the audio samples. This can be hard, for the "average" user to understand.

The World Wide Web (WWW) can also be used with audio. Originally when a user follows a link to a sound file, it would be downloaded to the client's machine and then played back, using a native audio application. Downloading this file could take a long time, especially if the network connection was slow and / or the file is large. With the development of streaming technology however, it has become possible to split these large files into smaller packets and then stream these packets to the clients. Each client would contain a streaming tool, that would buffer enough of these packets so that they could be played back to the user. The Web however, can not be used to create links to and from audio files. Audio is viewed as being just another media format that can either be downloaded or streamed to the clients.

This streaming technology and the development of the Synchronized Multimedia Integration Language (SMIL), allows users to create synchronized multimedia presentations over the Web. SMIL can be used to combine different types of media, such as audio, video, text and images, with synchronization elements into a presentation. Hyperlinking elements can also be used to create hotspots and links from a SMIL presentation

to other documents, including SMIL, on the Web. These links however are embedded within the SMIL documents and therefore, all of the problems associated with this type of link arise; for example link management and the dangling link problem. It is possible to create advanced presentations in SMIL, but there are very few SMIL editors and so these presentations have to be coded by hand. This can be a very tedious process. A solution would be for multimedia authoring companies, to have an "`Export to SMIL..`" function in their tools. This would reduce the complexity and the time required to produce SMIL documents.

Audio in SMIL is just another media element, that can be used in conjunction with other types of media. It is possible to create a link to and from an audio element however, the link information would have to be stored within the document. Also this type of link would have to be supported by the implementations of the standard. Several of these exist but they all implement a specific subset of the language, which can reduce the functionality of SMIL presentations.

3.7 Summary

This chapter has focused on the audio domain and how traditionally it has been neglected, in favour of the visual domain. This can be clearly seen in the development of interactive computer systems, which are completely visual in nature. Sound however, can convey an enormous amount of information and the human ear can use this to detect many different types of sound and where they are located.

In this chapter, the most popular techniques for creating multimedia applications and the way in which they handle audio information are discussed. The different parts of the MPEG standard are also described and they include MPEG-1 and MPEG-2 for the coded representation, storage and delivery of high-quality audio and video data. The MPEG group realised that audio was just as important as video and therefore with MPEG-4 more advanced audio tools were developed. These include tools to handle musical scores, audio scenes and special effects for 3-D audio placement. As a result MPEG-4 is the most up-to-date standard for handling audio information, however at this stage very few implementations exist.

Digital Audio Broadcasting has also been described and at the moment it provides high-quality digital radio and data broadcasting services. These services can provide

information about individual programmes and with the development of a return channel, it will become possible for users to request more information about these programmes.

Finally this chapter describes how the audio domain has been used in several "Open" Hypermedia Systems (OHSs) and the WWW. Several OHSs support the creation of links to and from audio files. However with the Harmony Audio Player these files are downloaded to the client for use and with the SoundViewer tool, the files are usually stored on the same machine that is using Microcosm. The World Wide Web traditionally downloaded temporal media files, such as audio and video, to the client's machine for playback. However as the quality and quantity of these files increases and as the Internet becomes more congested, it is becoming impossible to download these large files.

To help overcome this problem streaming protocols were developed. These are used to split large temporal media files into smaller packets, which are then streamed to the clients. The next chapter will describe these protocols and the underlying network protocols in more detail.

Chapter 4

Streaming Media Protocols

4.1 Introduction

This chapter describes the current version of the Internet Protocol (IPv4), that is being used over the Internet today. The previous chapter describes how the quality and quantity of temporal media files is steadily increasing. Traditionally these files would be downloaded, over the Internet, to the client and then played back. This can however, take a considerable amount of time, due to poor network connections and congestion.

To overcome this problem the IETF have developed two new protocols, which can be used to stream the media over the Internet. These streaming protocols use IPv4, for the control and delivery of the media data. They are called the *Real-time Transport Protocol* (RTP) and the *Real Time Streaming Protocol* (RTSP) and they are described in this chapter.

Several limitations of the current Internet Protocol, have caused the IETF to develop the next generation of this protocol, IPv6. These limitations and the five main changes to IPv4 are discussed, in more detail, within this chapter.

4.2 The Traditional Protocols

The *Transmission Control Protocol / Internet Protocol* (TCP/IP) is the main protocol suite used with the *Internet*. Halsall [42] 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* – which handles three different types of address, over the current Internet protocol IPv4 [78]. They are *unicast*, *broadcast* and *multicast* addresses. 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 this group. An IPv4 address is 32 bits in length and it is made up of four 8 bit numbers; for example `124.35.67.8`.

2. *Fragmentation* and *reassembly* – which handles the way in which the data is transmitted, across different parts of the Internet. If the datagrams, sent by a host, are larger than the packet sizes, used by a particular part of the Internet, then they 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* – which is used to determine which subnets, within the Internet, the datagrams must travel over, to get to the destination host. This could involve travelling over several different LANs or WANs.

4. *Error reporting* – which consists of several functions, that will detect and report errors back to the IP used in the source host. For example, the process of reassembly

could cause several packets to be discarded, which will result in an error being sent back to the source's IP. Halsall [42] 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 [42].

4.3 The Real time Transport Protocol

The *Real-time Transport Protocol* (RTP) was developed by the "Audio-Video Transport Working Group" of the *Internet Engineering Task Force* (IETF) and it has recently become an Internet standard. The IETF's RFC 1889 [8] specification describes RTP 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, which is discussed in the previous section. It can also be used with other transport or underlying network protocols.

RTP usually works in conjunction with another protocol called the *Real Time Control Protocol* (RTCP)[1], which provides minimal control over the delivery and quality of the data. It performs four main functions and these are:

1. *Feedback Information* – which 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 and the number of packets lost, 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* – which 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* – which 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* – which is an optional function, that 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.

[1] It is also known as the RTP Control Protocol.

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.

4.4 The Real Time Streaming Protocol

The *Real Time Streaming Protocol* (RTSP) is a proposed Internet standard which was developed by Netscape Communications Corporation, RealNetworks[2] and Columbia University. The current RFC [53] describes it 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 such as RTP, see **Section 4.3**. 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 URL; see RFC 2327 [53], Section 3.2. URLs are described in **Section 2.3.1**. RTSP supports three schemes and they are:

1. "`rtsp`" – which identifies a reliable protocol, such as TCP, for the delivery of the commands. This is the most commonly used scheme.

2. "`rtspu`" – which identifies an unreliable protocol, such as UDP, for the delivery of the commands.

3. "`rtsps`" – which requires a TCP connection, secured by the *Transport Layer Security* (TLS) [130] 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.

[2]Formerly known as Progressive Networks.

RTSP is intentionally similar, in syntax and operation, to the *HyperText Transfer Protocol* (HTTP/1.1) [108], see **Section 2.3.1**, which is an Internet draft 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 [119], however, describe one of the fundamental reasons why RTSP is based upon HTTP/1.1. The previous version of HTTP, 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 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 the real-time delivery of streaming data.

Although RTSP is similar to HTTP/1.1, there are several areas in which it differs. One of these is RTSP's need to maintain "session state" in almost all situations. In RTSP a session represents a complete transaction, between a client and a server. For example, the creation and setup of the streams, play back and then the closing down of the streams. To handle this type of transaction, RTSP may use the *Session Description Protocol* (SDP) [98], to create *session descriptions*. These contain all of the information required for a multimedia session.

A SDP message consists of three main sections:

1. The *Session Description* – which contains general information about the session such as the protocol version, the session name, the creator and session identifier.
2. The *Time Description* – which contains time information.
3. The *Media Description* – which contains information about the actual media such as bandwidth information, media title and media name and transport address. It can also contain extra media attributes.

A session description can have more than one time and media description, depending on

the number of media streams. When a SDP message is received, clients will use it to handle the streams in a session. RFC 2326 [53] Appendix C, describes how RTSP could use SDP, in more detail.

As mentioned previously however, the data in RTSP 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[3]. 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 multiple users, the server needs to be able to maintain the "session state" for each client. For example, one client might be setting up several streams and therefore the session, for this client, will be in the "`SETUP`" state. For another client, the session might be in the "`PLAY`" state. By maintaining the session state, the server will ensure that it can correlate the RTSP requests, from each client, to the relevant session. HTTP/1.1, however, is a stateless protocol; there is no need to save the state of each session for each client. The RTSP state diagram is shown in **Figure 4.1**.

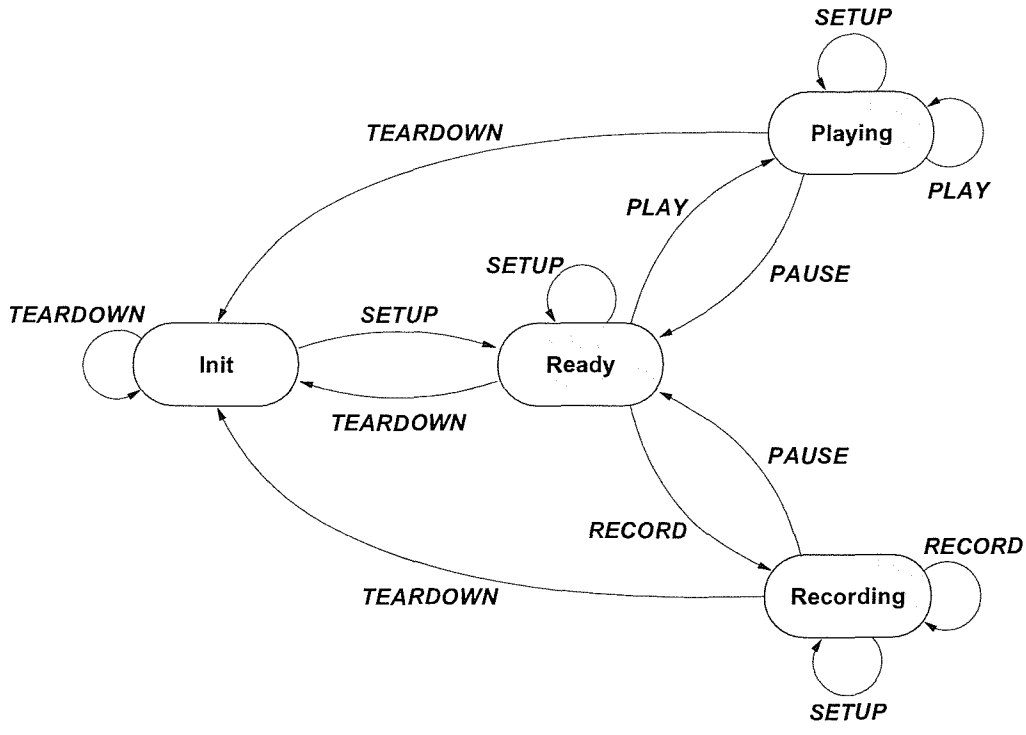

Figure 4.1: The RTSP client/server state diagram.

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

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. This is shown in **Figure 4.2**, which is a simple diagram showing how three RTSP clients, have connected to a RTSP server. Each of the clients (A, B and C) have a single control channel, which could be either reliable or unreliable, e.g. TCP or UDP, respectively. As mentioned previously, both the clients and the server can send and receive messages, over this channel. Clients A and C however, only have one data stream which is used to transmit the media data, e.g. packets of audio, video etc., from the server to the client. Client B has two data streams, possibly one for audio and another for video. There is no communication, on the data streams, between the client and the server. The server can only use these streams to send packets of data to the client.

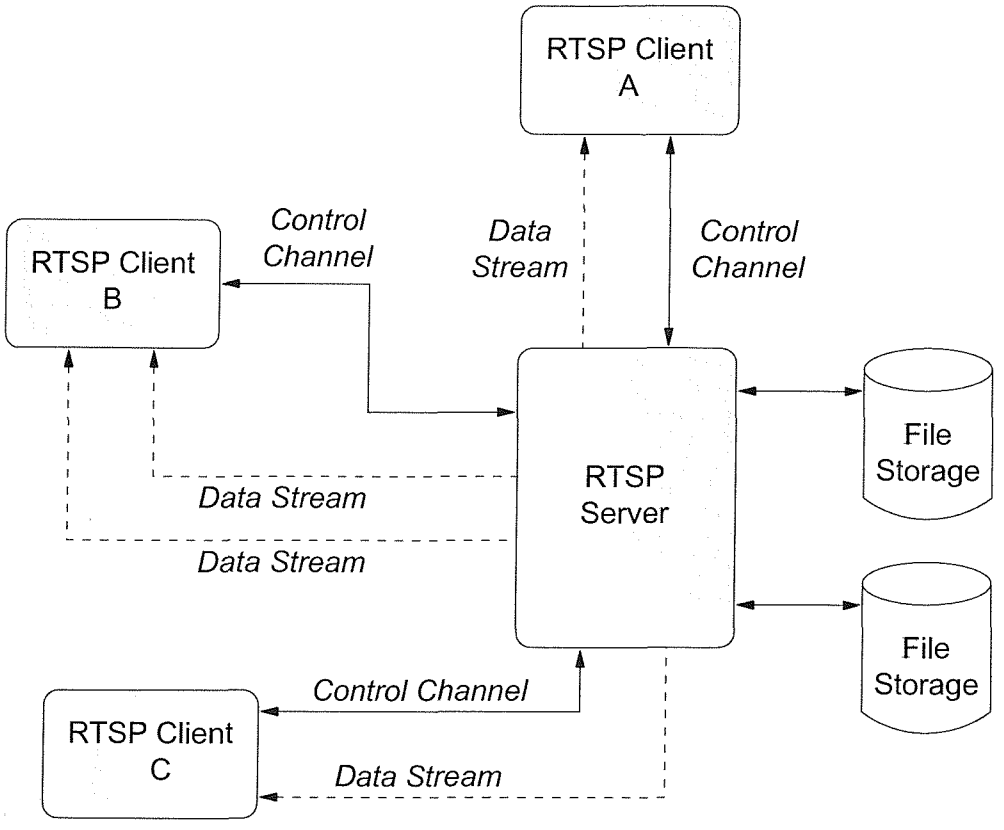

Figure 4.2: The connections between three RTSP clients and a server.

RTSP is really 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 itself. Early implementations of these servers support UDP, multicast UDP, TCP and 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.

The World Wide Web Consortium (W3C), see **Section 2.3.1**, and several other companies, e.g. RealNetworks, are currently using RTSP. The W3C has developed a declarative language called SMIL, see **Section 3.5.5**, which can be used to combine independent multimedia objects into a synchronised multimedia presentation. These media objects are usually streamed to the clients and therefore, SMIL uses RTSP to control the delivery of these streams. Once created, these presentations can be easily transmitted over the WWW. When a Web browser receives a SMIL presentation, it usually executes an application, e.g. RealNetworks RealPlayer etc., which buffers and then plays the streams.

4.5 IPv6: The next generation Internet Protocol

Sun Microsystems [20] describes how IPv4 became an Internet standard in 1981 and at that time, the Internet was a community of approximately one thousand systems. In 1999 this number had grown to approximately 100 million and it has been estimated that an increasing number of devices will become "Internet-aware". These include mobile phones, pagers and *Personal Digital Assistants* (PDAs). As the number of these devices grow, so will the demand for more addresses.

As mentioned previously in **Section 4.2**, IPv4 has an address length of 32 bits made up of four 8 bit numbers. This provides about four billion possible addresses. However

this is a theoretical limit and the actual number is constrained, by several factors, to a few hundred million. With the current rate of growth it has been estimated that these addresses will run out between 2002 and 2012.

Temporary solutions to this problem have already been developed and one of these is the *Network Address Translator* (NAT) [107]. This device sits between the Internet and private communities of users. It uses one IPv4 address to communicate with the Internet and several other private addresses for each of the users. All requests from this community appear to come from a single machine, hosting the NAT, on the Internet. On receiving information the NAT forwards it to the correct user. This device works well for small private networks. However as the size of the network increases, it can restrict the flow of the data. It also undermines Internet security.

To help solve this problem and several other limitations of IPv4, the *Internet Engineering Task Force* (IETF) decided to develop the next generation of the Internet Protocol; called *IPng* or *IPv6* [116]. The protocol itself became an Internet standard in 1998 and it consists of five main changes to IPv4:

1. *Expanded addressing capabilities* – which increases the address size from 32 bits to 128 bits (usually represented as eight 16 bit numbers.) This provides roughly $6*10^{23}$ addresses per square meter over the entire face of the earth. Unicast and multicast addresses are supported, with broadcast addresses being phased out in favour of a special form of multicast. A new *anycast* addressing scheme is defined which is similar to multicast, except that the packets are only sent to one node within the anycast group.

2. *Header format simplification* – which reduces the overhead of processing the "common case" headers, used within packets. This process also limits the bandwidth cost of the IPv6 header.

3. *Improved support for extensions and options* – which changes the way IP header options are encoded. This improves the forwarding of packets, reduces the stringent requirements for the length of options and provides greater flexibility for introducing new options.

4. *Flow label capability* – which introduces a new 20 bit *Flow Label* field within the header. This field is used to label specific sequences of unicast or multicast packets, that require special handling by IPv6 routers. The routers use this label to provide a

non-default quality-of-service (QoS) or a real-time service, possibly for streaming media. It ensures that the packets "flow", from the source to the destination, with a minimum amount of processing overhead at the routers. This part of the IPv6 protocol is experimental and is still under development.

5. *Authentication and privacy capabilities* – which provides greater support for authentication and data integrity. Optional support for data confidentiality is also provided.

There is no doubt within many organisations and research communities that IPv6 will eventually replace IPv4. Sun Microsystems [20] explain however that the transition phase, between these two protocols, will take a reasonable amount of time and as a result the Internet will be made up of both IPv4 *and* IPv6 hosts. To assist and accelerate this transition, the basic socket APIs for IPv6 [109], supports both protocols. These APIs provide the functionality for users to develop IPv6-aware applications and since they also support IPv4, these applications will be able to communicate, over the Internet, using either protocol.

Several implementations of this protocol already exist and an experimental backbone network, the *6bone*, has already been created. This is mainly being used to assist in the development and testing of different aspects of the protocol.

4.6 Summary

The Internet is one of the most popular mediums for communication and information exchange in use today. At its inception, the Internet consisted of about 1000 users, however this has grown to approximately 100 million. The reasons for its popularity was the development and eventual use of the current Internet Protocol (IPv4). This allowed many systems to communicate using a suite of core protocols.

Another factor in the Internet's popularity, was the development of the WWW. The Web uses its own protocol (HTTP) "on-top-of" the current Internet Protocol and as the popularity of the Web increases, so does the use of the Internet. The Web can handle different types of media, including streaming media. Media streams are used to overcome the problems of downloading large audio and video files for playback.

IPv4 however, was not designed for this type of media and therefore the IETF developed two streaming media protocols; the Real-time Transport Protocol (RTP) and the

Real Time Streaming Protocol (RTSP). These protocols use IPv4 for the control and delivery of media streams. RTP is an Internet Standard and has been used in a number of systems, including video conferencing. RTSP is a proposed Internet Standard and it is being used for the delivery of media in SMIL presentations.

An increasing number of devices are becoming Internet-aware. However, the current protocol is slowly running out of Internet addresses. To overcome this problem and several other limitations with IPv4, the IETF has developed the next generation of Internet Protocol (IPv6). This protocol extends the address space and introduces a new field, the flow label, which is used in IPv6 headers. This label can be used to provide a real-time service, possibly for streaming media.

This chapter has discussed two of the most commonly used protocols, for streaming media over the Internet and the WWW today. However "Open" Hypermedia Systems (OHSs) are not using these protocols, for handling audio information. By using a streaming protocol the open hypermedia tools, described in the previous chapter, would be able to store the audio files on separate streaming servers. These servers would then stream the audio to the tools. This would overcome the problems of downloading these files to the client's machine or storing them on the same machine that is running the OHS.

The next chapter describes how the original SoundViewer tool, for Microcosm, was extended using RTSP, thus overcoming the problems mentioned above.

Chapter 5

The Design and Implementation of the Streaming SoundViewer Tool

5.1 Introduction

In this chapter the original SoundViewer tool for Microcosm is discussed. There are several ways in which audio information can be visually displayed and they include the scrollbar and waveform method. Each of these graphical metaphors are reviewed and the interface, that was eventually chosen for the SoundViewer, is discussed. The implementation of this tool is described and finally a simple demonstration, showing how links can be created to and from the audio domain, is given.

Chapter 3 however, describes how this open hypermedia audio tool stores the audio files on the same machine that runs Microcosm. These files can be quite large, especially if they are high-quality samples. It is possible to store the files on a separate computer on the local network. A permanent connection to this computer, would allow Microcosm and hence the SoundViewer to access the audio. If there is a problem however, with the network and / or the computer, then the SoundViewer will not be able to access these files.

The previous chapter describes two of the most popular streaming media protocols that are being used today. The WWW is already using them to stream temporal media, such as audio and video files, from servers to Web browsers. By using a streaming protocol, to handle audio files, the SoundViewer tool would be able to overcome the problems mentioned above.

This chapter describes the design and implementation of the streaming SoundViewer tool for Microcosm. It explains how the original SoundViewer was extended using the final draft of the Real Time Streaming Protocol. Finally a demonstration of this new tool is given.

5.2 The original SoundViewer Tool

The SoundViewer 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 [118] describes the development of the SoundViewer tool and discusses some of the unique properties of the audio domain, which is described in more detail in **Chapter 3**.

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 3.3**, **3.5** and **3.5.4** 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 or the system shutting down.

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 SoundViewer 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, RealNetwork's RealPlayer and Apple's Quicktime player 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 SoundViewer tool was designed to represent WAV, MIDI and CD-Audio in a 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 [118] 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 SoundViewer, is called *Microcosm*. This "open" system is discussed in more detail in **Section 2.3.2**.

5.2.1 The Interface

The user interface of the SoundViewer 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 **Figure 5.1**.

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 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 SoundViewer 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

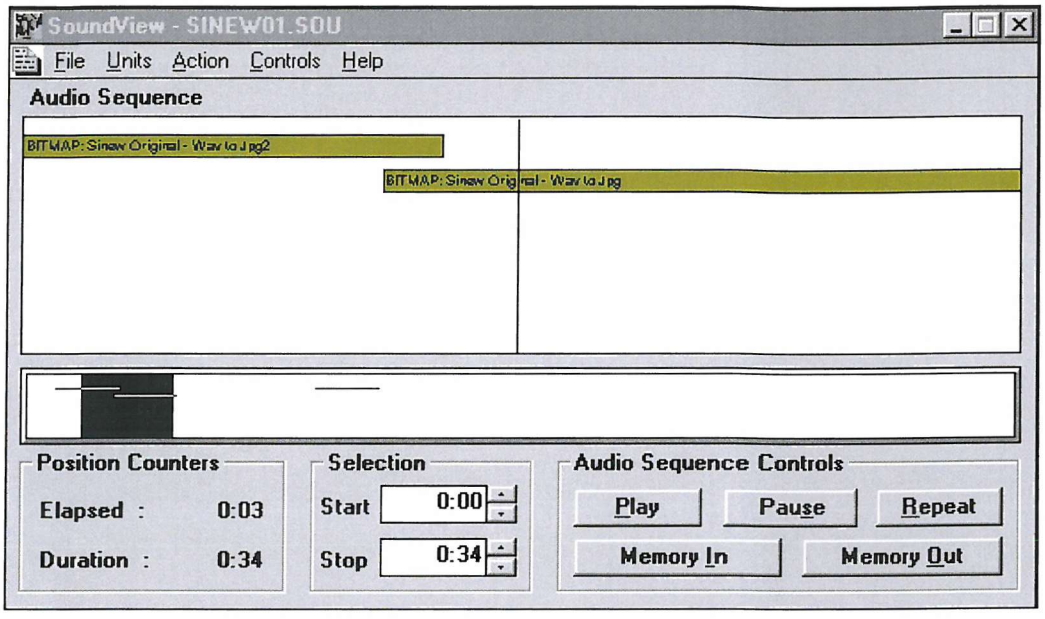

Figure 5.1: The SoundViewer Interface.

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 SoundViewer 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 SoundViewer 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 SoundViewer automatically loads the project file.

To create and traverse links to and from the SoundViewer 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:

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 SoundViewer 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 SoundViewer 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 SoundViewer, 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 **Figure 5.3**. 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.

5.2.2 The Implementation

The original SoundViewer 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 SoundViewer, 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 **Figure 5.1**.

Goose and Hall [118] describe in their paper how the SoundViewer 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 [118]). 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 SoundViewer. 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 **Figure 5.2**. 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 SoundViewer and the creation of an abstract layer, between the user interface and the audio controls (MCI), ensures that any new and emerging audio

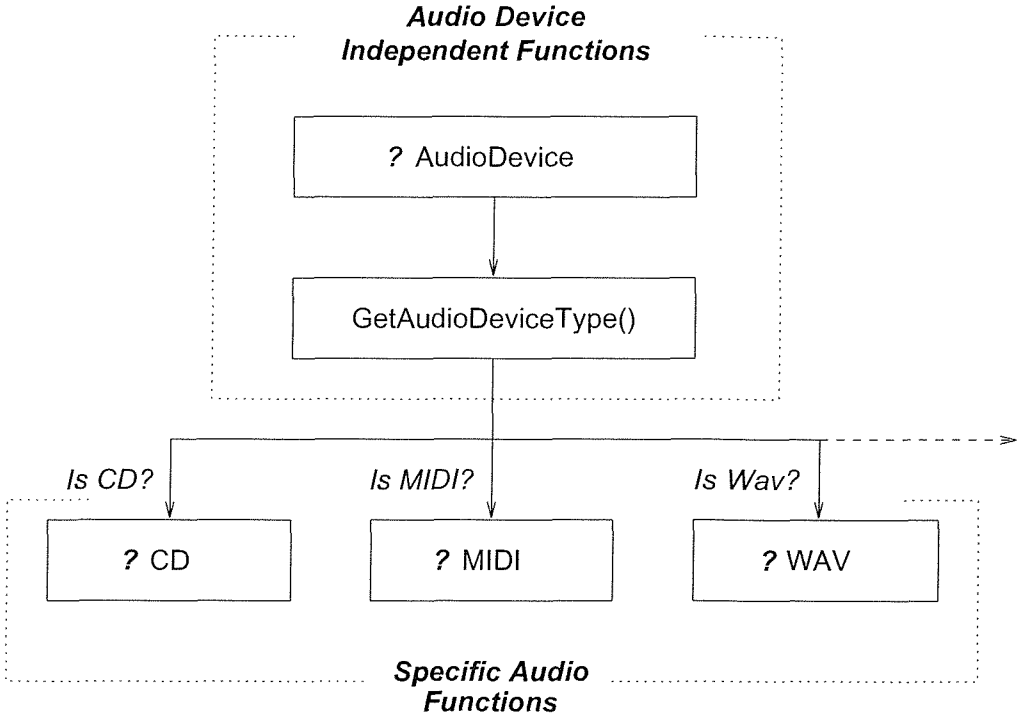

Figure 5.2: The Audio Device Layers.

formats can be easily supported.

5.2.3 The original SoundViewer case study

To test the original SoundViewer 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 2.3.2** and **5.2.1**.

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 SoundViewer tool to be displayed and the user could then play the audio sample. **Figure 5.3** shows a small sample of a demonstration.

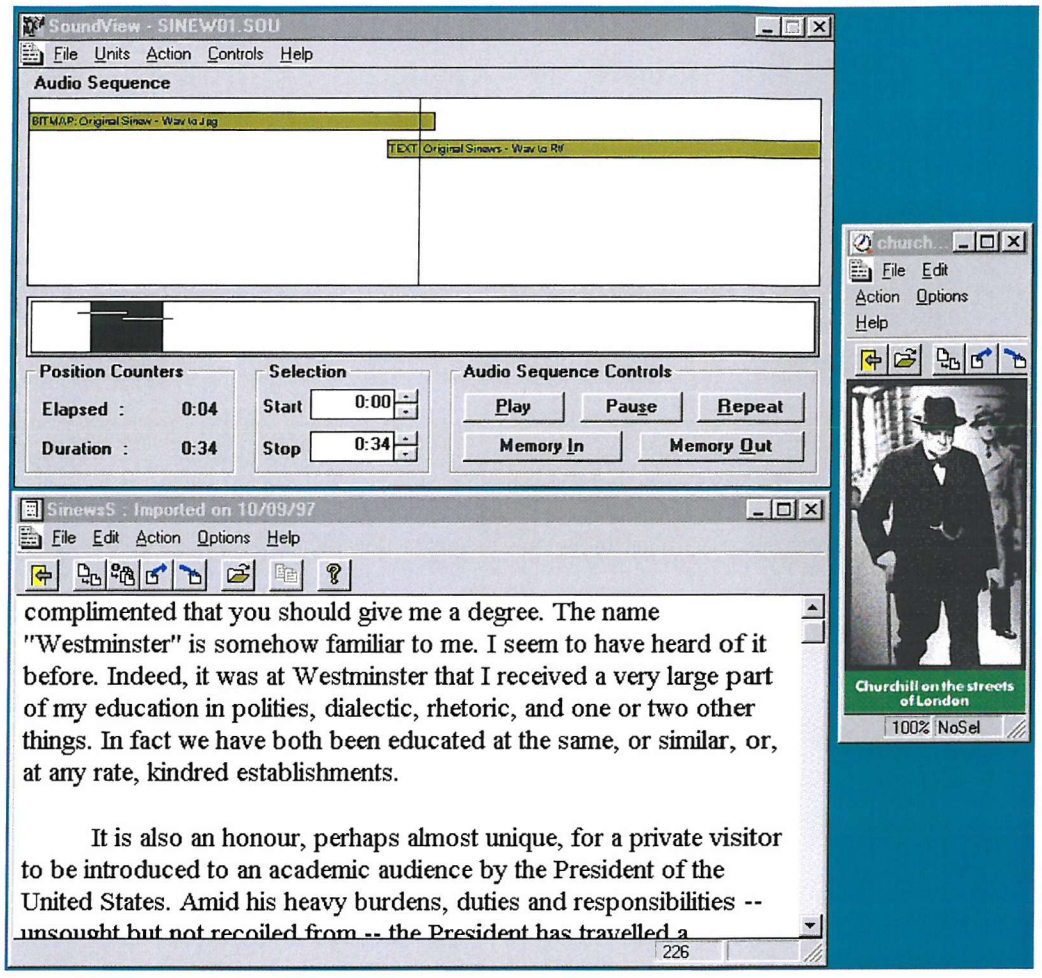

Figure 5.3: The original SoundViewer demonstration.

In this screen-shot, the SoundViewer 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, **Figure 5.3**, one of the links displays a picture of Winston Churchill and the other, the text transcript of his speech.

5.3 The development of the streaming SoundViewer

Over the last few years the World Wide Web, see **Sections 2.3.1** and **3.5.4**, 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 4**.

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. **Sections 4.3** and **4.4** 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 SoundViewer and an initial implementation of the protocol is already available for public use, it was decided that this protocol would be used with the SoundViewer. The following sections describe the design and implementation of this streaming audio tool.

5.3.1 The Design

As described in the previous section, **Section 5.3**, 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 4.4**, 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 SoundViewer, a streaming SoundViewer 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 SoundViewer, the viewer's audio module was extended. The audio module consists of two layers, see **Section 5.2.2**; 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 **Figure 5.4**.

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 SoundViewer, 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 5.2.1**, but apart from

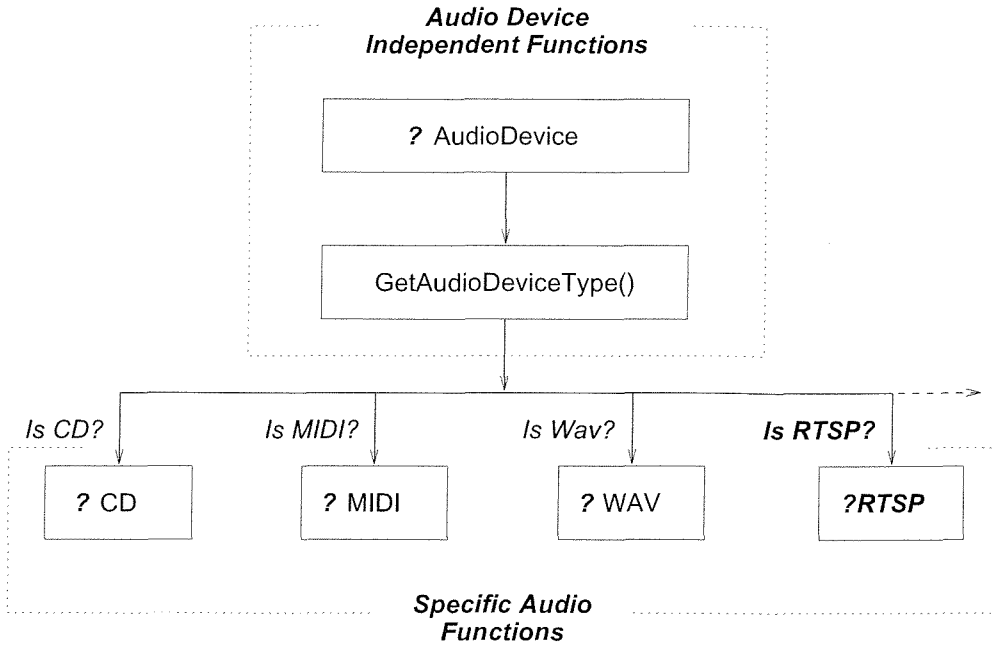

Figure 5.4: The Audio Device Layers with RTSP.

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 pause playback until enough data had been received.

With the original SoundViewer, 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 SoundViewer has effectively become an RTSP client. **Figure 5.5** shows the interaction between the streaming SoundViewer 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 4.4**. 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 5.6** and **5.7** 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 SoundViewer 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 RFC 2326 [53]. 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. **Figure 5.8** shows the interaction between the streaming SoundViewer and the media server, when the play command is sent to the server.

With the original SoundViewer two events occur when the audio file is played:

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.

[1] This is usually implementation specific.

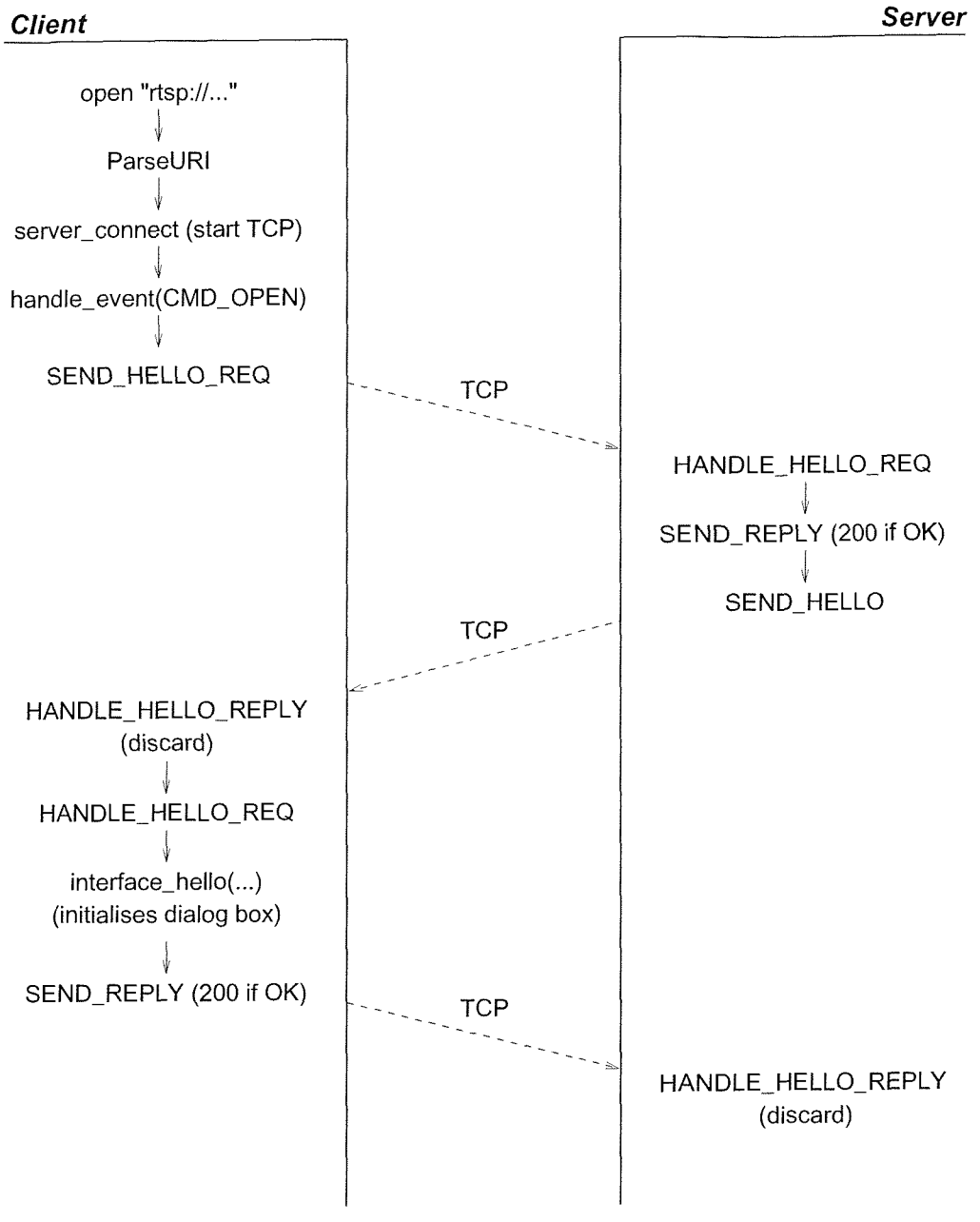

Figure 5.5: Client / Server interaction for the RTSP open function

Both of these components use a Media Control Interface (MCI) function to obtain this position. MCI, however, will only work with files stored on the local machine. With the streaming SoundViewer, 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.

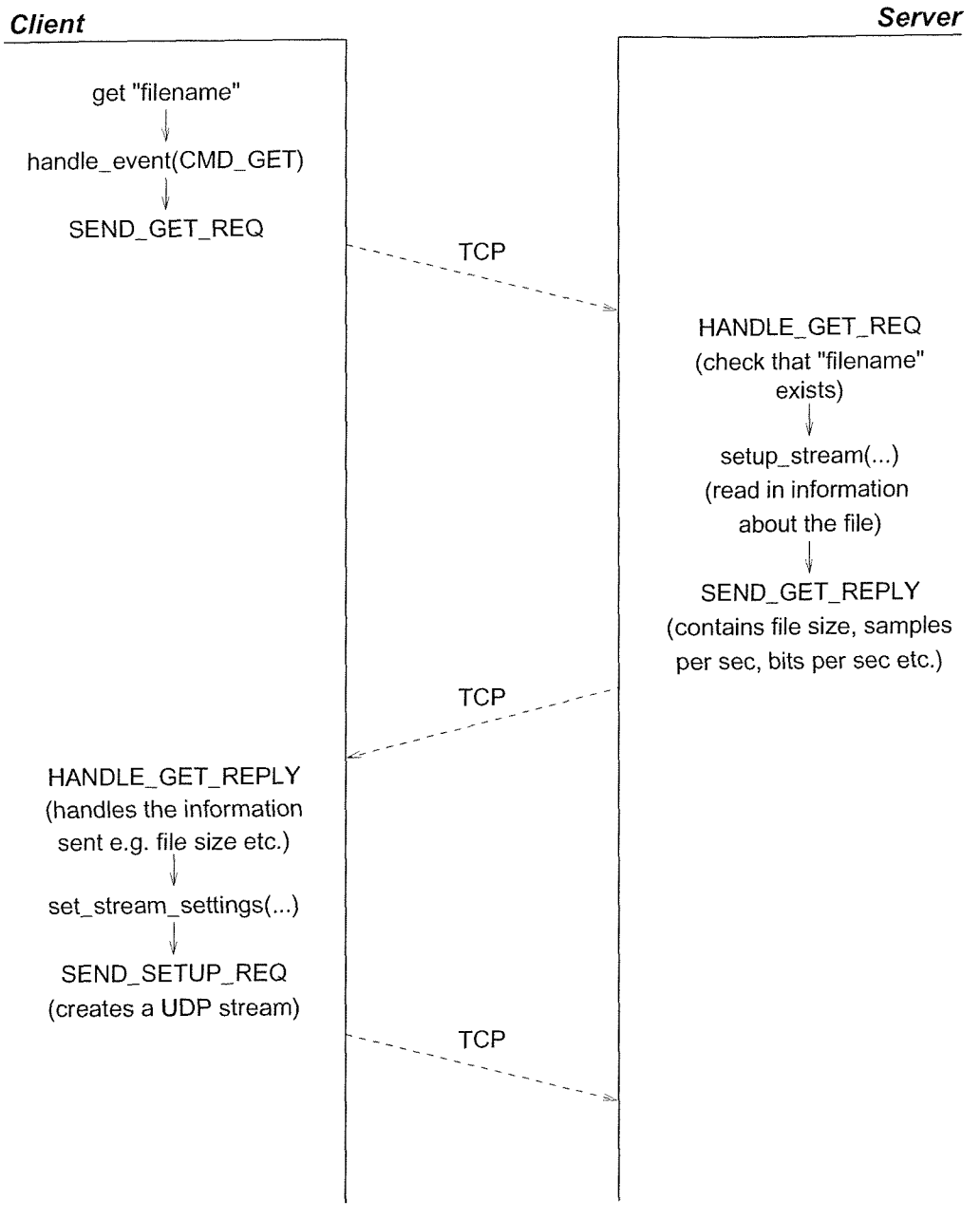

Figure 5.6: Client / Server interaction for the RTSP `get` function

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.

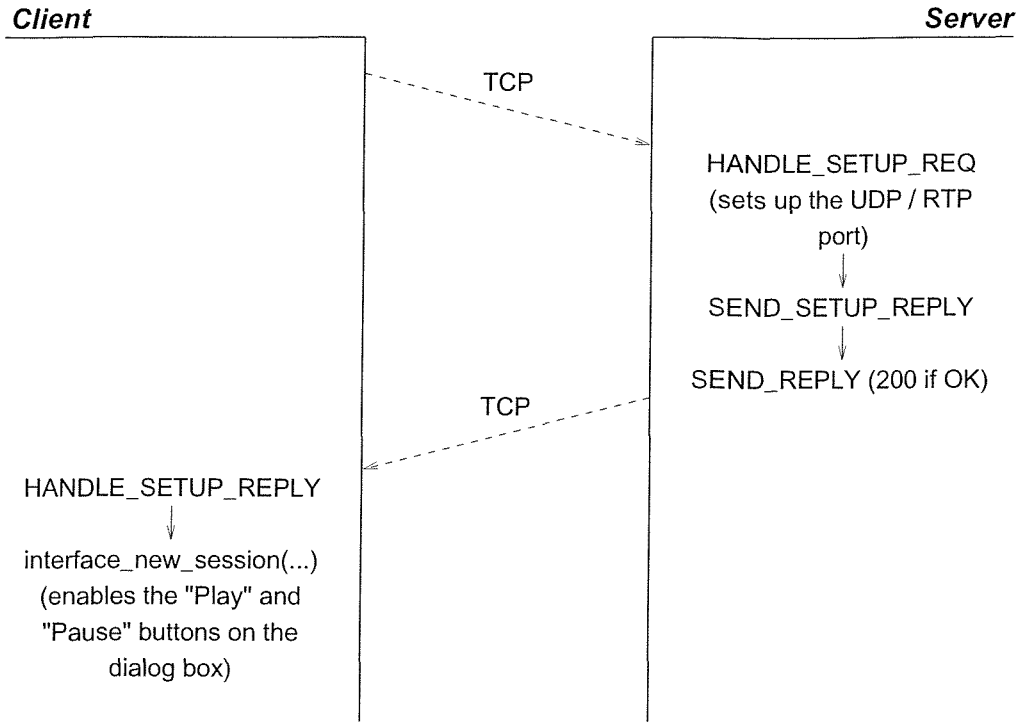

Figure 5.7: Client / Server interaction for the RTSP `get` function (cont'd)

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 SoundViewer's audio module. There was no need to modify the MCI controls for the other supported audio formats, since their functionality has not changed.

5.3.2 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 3.5** and **5.2**. Therefore the original SoundViewer 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

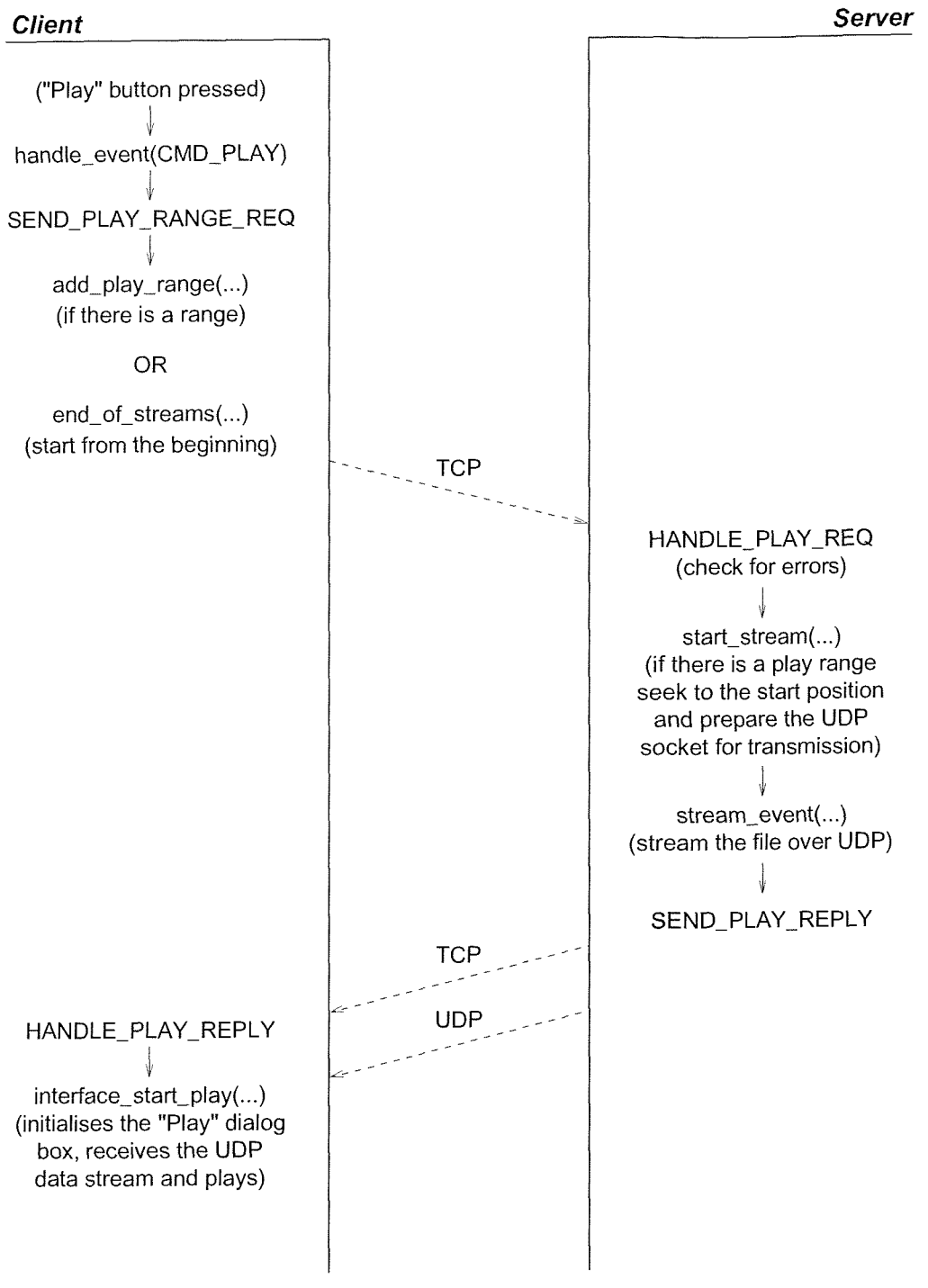

Figure 5.8: Client / Server interaction for the RTSP `play` function

NT. As a result, the streaming viewer could not be created by simply "inserting" the relevant RTSP client code into the original SoundViewer 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 SoundViewer 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 5.2.2, 5.3.1** and **Figure 5.2**.

Another possible solution would have been to convert the SoundViewer 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 SoundViewer tool was relatively straight forward. New code had to be created to handle the virtual time unit (VTU), see **Section 5.3.1** and the interface between the RTSP client and the SoundViewer'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 SoundViewer, a demonstration similar to the original case study, see **Section 5.2.3**, 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 **Figure 5.9**.

In this screenshot, when the `Play` button has been pressed, a command is sent to the media server to stream the audio to the SoundViewer. 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.

5.4 Summary

In this chapter, the design and implementation of the streaming SoundViewer tool for Microcosm has been described. The design rationale of the original SoundViewer tool is

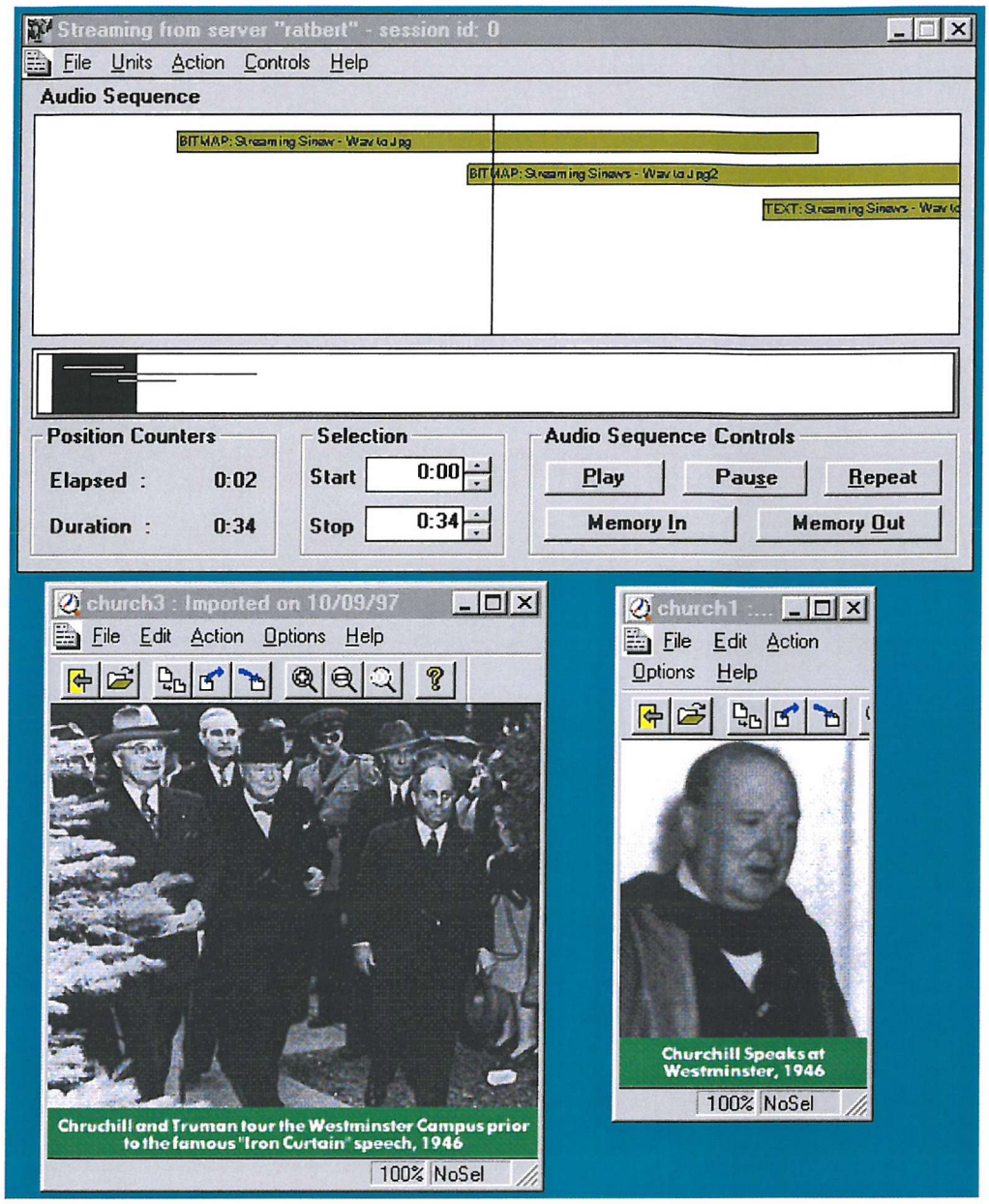

Figure 5.9: The streaming SoundViewer.

also discussed, especially the graphical metaphor used for the interface. The implementation of the original tool and the streaming version are reviewed and demonstrations of both tools are given.

This chapter has shown that it is possible to extend an existing "open" hypermedia audio tool, to support streaming media protocols; specifically the Real Time Streaming

Protocol (RTSP). The streaming SoundViewer has effectively solved the original problem, of storing the audio files on the same computer that is running Microcosm.

However, **Chapter 2** describes the reasons why the Open Hypermedia Protocol (O-HP) was developed. The open hypermedia research community realised in 1994, how the current generation of "Open" Hypermedia Systems (OHSs) could not interoperate. The majority of these systems stored their link information on separate link servers and used proprietary protocols for communication. As a result of this, the clients developed for one system could only be used on that system. Whenever a new data format needed to be supported, the developers would have to either extend an existing client or create a new application. If another OHS had already developed a tool for this format, the other OHSs would not be able to use it. This resulted in the developers wasting a considerable amount of time on the clients, instead of the main area of research, the link servers.

The Open Hypermedia Systems Working Group (OHSWG) decided therefore to develop the Open Hypermedia Protocol. It would be used for communication between the OHS clients and the link servers. An OHP-aware client would then be able to communicate with an OHP-aware server. Over time however, this protocol grew in size until the OHSWG realised, that a single protocol would not be able to handle all of the functionality required. As a result the protocol was divided into specific hypertext domains, including the traditional form of navigational hypertext. OHP-Navigational (OHP-Nav) was successfully demonstrated at two conferences.

The Southampton members of the OHSWG realised however, that a higher-level structure could be created, to work across three of the most common hypertext domains; navigational, spatial and taxonomic hypertext. As a result the Fundamental Open Hypermedia Model (FOHM) was created. This model concentrates on the way in which associations can be shared across these domains. It does not use a protocol for communication, between an OHS client and server. Therefore it can be described as an interoperable exchange format, for the three hypertext domains mentioned above.

The next chapter describes how the RTSP protocol itself, can be extended to support open hypermedia. To ensure that this new protocol uses an interoperable exchange format, FOHM is used. This combination will complement both the protocol and the model itself, because RTSP will be able to handle open hypermedia and FOHM will be able to use this protocol for communication.

Chapter 6

An Open Hypermedia tool for Temporal Linking with Audio Streams

6.1 Introduction

This chapter describes the development of an Open Hypermedia tool that allows users to create, display and follow temporal links with audio streams. This is achieved by implementing a new version of the RTSP framework and extending it with new methods, that support open hypermedia.

The previous chapter describes how an existing open hypermedia audio tool, for Microcosm, was extended to support audio streams. It was possible, using this tool, to create links to and from streaming audio. This functionality however, to create temporal links, could only be used with this "Open" Hypermedia System (OHS).

The majority of OHSs exhibit similar behaviour, because they use proprietary protocols for the communication between their clients and servers. Clients developed for one system will not be able to communicate with another and as a result, these systems can not interoperate. **Section 2.5** describes this in more detail and the development of the Fundamental Open Hypermedia Model (FOHM).

FOHM is a higher-level structure that is designed to work across hypermedia domains. Therefore it can be described as an interoperable exchange format for open hypermedia. However, this model only describes the structure of the interoperable associations between these domains; it does not describe a protocol for communication.

This chapter describes how the new RTSP framework uses FOHM, as an interoperable exchange format, for storing link information. The benefits of this are twofold. This new protocol can use FOHM to handle the open hypermedia information and FOHM can use RTSP, as a communication mechanism. The new methods, mentioned previously, will be used with FOHM to create, display and follow temporal links.

The RTSP framework has also been designed to work with IPv6 and therefore an IPv6-aware server will be able to communicate with both IPv4 and IPv6-aware clients.

6.2 The Design of the new RTSP Framework

As described in **Section 5.3** an implementation of the RTSP framework, see **Section 4.4**, was used to develop the streaming SoundViewer tool, for the open hypermedia system Microcosm. This section also explains that the IETF require at least two reference implementations of a protocol, before it can even be considered to be a proposed standard. The version of the protocol that was used for this tool however, was based on the final draft of the specification and its reference implementation. This final draft was modified before it became the actual proposed standard, RFC 2326 [53] and the implementation was very basic; it had enough functionality to handle simple streams. No reference implementation, of the proposed standard, was ever released. This was mainly due to two reasons; the protocol had already been tested on a number of systems and a substantial amount of feedback had already been received. This proved the viability of the protocol and therefore, no further implementations were required.

To ensure that the new framework was compliant with the current specification, it was decided that a new implementation would be required. When designing this new implementation, several factors had to be considered and these were:

- The *underlying transport protocols.* RTSP runs "on-top-of" the current Internet Protocol, IPv4 see **Section 4.2**. During the design of the framework however, the IETF were also developing the next version of the Internet Protocol, IPv6 see **Section 4.5**. IPv6 will eventually replace IPv4 and to ensure backwards compatibility, during the eventual transition, it also supports IPv4. As a result of this, it was decided that the new RTSP implementation would support IPv6. This ensures that the RTSP client and server will be able to communicate, over the Internet, using either protocol.

RTSP's underlying delivery mechanism is usually the Real-time Transport Protocol (RTP), see **Section 4.3**, although developers are free to use different protocols. For this implementation, it was decided that RTP would be used because it is the de-facto Internet standard for the real-time delivery of continuous media.

- The *Operating System.* The previous item described the reasons why the underlying protocols were chosen. At the time of development however, only a few operating systems had limited support for IPv6 and the majority of these were UNIX-based. These included Linux, Sun Microsystems Solaris, FreeBSD and other BSD derivatives. Microsoft Windows, the most prevalent operating system, at this time had very little support[1] for this new protocol. Therefore it was decided that one of the UNIX-based systems would be used for development.

- The *Programming Language.* The original reference implementations of RTSP used the C programming language for development. This was mainly due to the fact that C is a portable language and most operating systems have at least one C compiler at their disposal. These implementations could then be easily used and tested on a number of different systems, by simply re-compiling the source code.

  Stroustrup [10] describes how C was originally designed to replace the large amounts of assembler code, used in the most demanding of system programs. It achieves this by providing types, operators and structures that simply and efficiently handle the low-level objects used within the system; for example characters, numbers and addresses. It has been used to develop advanced applications, including the UNIX operating system and its derivatives.

  C however does have several limitations. It was designed to be "close to the machine" and as a result it is sometimes called a *lower-level* language. It is still possible, for instance, to include assembly language within C programs. For the development of more advanced programs, users require a reasonable amount of knowledge of the underlying hardware, especially for memory management and access. It is very easy for inexperienced *and* experienced developers to create programs that cause memory leaks, which can eventually result in the system crashing.

  To help reduce this type of problem, it was decided that a higher-level, object-oriented language would be used. Object-oriented languages provide a much more structured approach to program design and development.

[1] Microsoft has recently released advanced libraries and tools, for IPv6 development on Windows 2000.

During the design process, it was decided that a couple of libraries would be developed; to handle the underlying transport protocols and the *Session Description Protocol* (SDP), see **Section 4.4**. These two libraries would reduce the amount of code used within the RTSP implementation and once developed, they could also be used in other applications. The following sections describe the design of these two libraries and the design of the remaining RTSP framework.

6.2.1 The Socket++ Library

The Socket++ library is specifically designed to handle the creation, manipulation and deletion of IPv4 or IPv6 *sockets*. A socket is an endpoint for communication and in most operating systems, the `socket` function is used to create this endpoint. The function itself usually takes three parameters:

1. The *domain* parameter – which is used to specify a communication domain such as IPv4 or IPv6.

2. The *type* parameter – which is used to determine the type of socket to create such as TCP or UDP, see **Section 4.2**.

3. The *protocol* parameter – which specifies the type of protocol to be used with the socket. Normally only a single protocol exists for a particular type of socket and protocol family.

The function returns a *descriptor* which is an integer used for any subsequent calls to the socket; for example for reading and writing.

RTSP uses two protocols. The first is the communication channel and it is used for communication between the client and the server. The second is the data channel which is used for the actual delivery of the data, from the server to the client. The communication channel requires a reliable protocol to ensure that the messages are delivered. From the specification, see RFC 2326 [53], this is usually TCP. The data channel does not require a reliable protocol and therefore, UDP will be used.

The library will contain several classes for handling the socket addresses, the sockets themselves and *exceptions*, which will be used to handle errors. There are two types of address; one is 32 bits in length for IPv4 and another is 128 bits in length for IPv6, see **Section 4.5**. In future only the data structures, used to hold the IPv6 addresses, will be

required because as mentioned previously, IPv6 also supports IPv4. Exceptions transfer control, from where an error occurred, to a designated piece of code. Users can explicitly define what this code will do and as a result they have a lot more control over the handling of errors.

The design of this library uses a specific object oriented modelling technique known as *inheritance*. Rumbaugh et al. [79] describe inheritance as the process of building hierarchical relationships between classes, by sharing attributes and operations. Usually a single class is defined broadly and then refined into more specialised *subclasses*. Each subclass incorporates or *inherits* the properties of the superclass and it can also add its own unique properties.

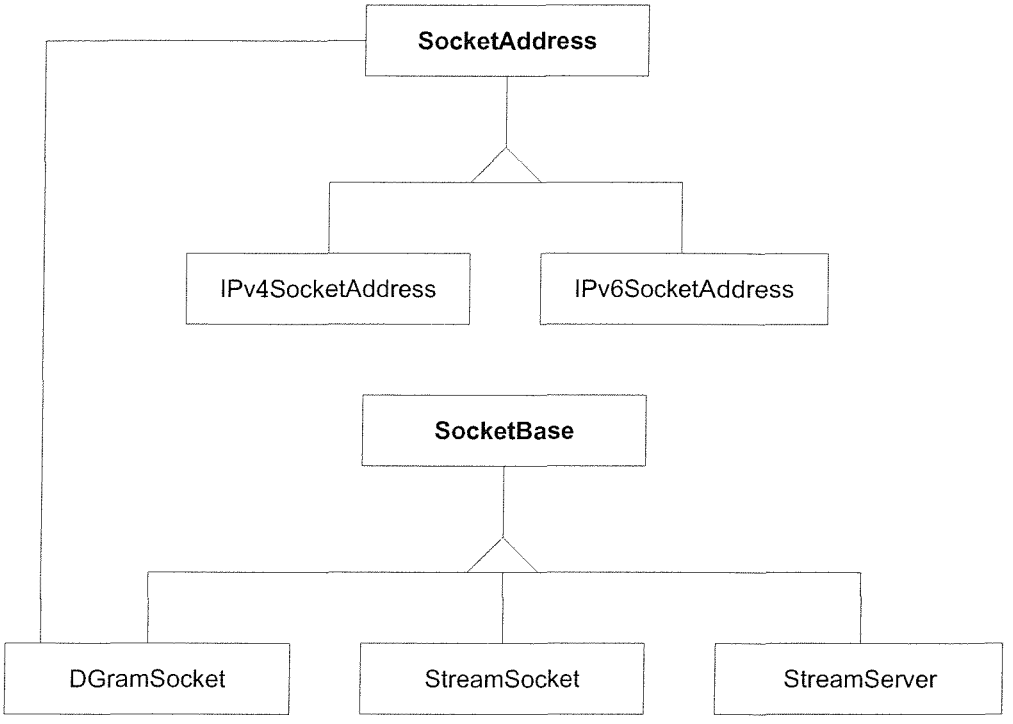

Figure 6.1: The main classes for the Socket++ library.

Figure 6.1 is the class diagram for the two superclasses that will be used in this library; the `SocketAddress` and the `SocketBase` classes. This diagram shows that the `IPv4SocketAddress` class and the `IPv6SocketAddress` class are derived (denoted by the upside-down 'v') from the superclass `SocketAddress` and they will be used to handle IPv4 and IPv6 addresses respectively. The `DGramSocket` class is derived from the `SocketBase` class and it will be used to define UDP sockets. The `StreamSocket` and the

`StreamServer` classes are also derived from the `SocketBase` class and they will be used to define TCP sockets for a client and a server respectively. A server usually has a socket for listening for new connections and when a client attempts to connect, the server will create a new socket to handle this connection. The original socket will continue to listen for new connections. In this way a server can handle multiple clients.

Inheritance is used in this design because the derived classes all share common attributes and operations; for example the `SocketAddress` class will have similar structures to handle addresses and the `SocketBase` class will contain the descriptor, returned from the `socket` function. There is no need to duplicate this functionality in each of the derived classes; it can be handled in the superclass.

The `DGramSocket` class also has a connection to the `SocketAddress` class because UDP sockets do not maintain a connection; they are connectionless. Therefore UDP sockets use specific functions to get the address of the connected peer. This address will be stored in the `SocketAddress` object, within the `DGramSocket` class.

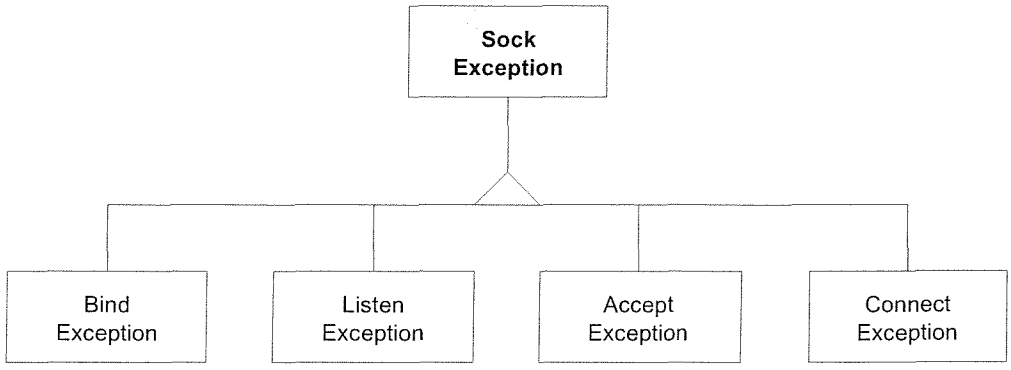

Figure 6.2: The classes for handling socket exceptions.

The exceptions that will be used within this library are shown in **Figure 6.2**. The superclass is called `SockException` and it will have four derived classes, that are called or *thrown* when a particular error occurs. They are:

- The *BindException* class – which will be thrown when an address, either IPv4 or IPv6, can not be bound to a socket.

- The *ListenException* class – which will be thrown when a TCP socket can not be used, by a server, for listening.

- The *AcceptException* class – which will be thrown when a TCP socket, on the server, can not accept a connection.

- The *ConnectException* class – which will be thrown when a TCP socket, on the client, can not be connected to the server.

These classes will be used to handle the communication between the RTSP client and server.

6.2.2 The SDP++ Library

The *Session Description Protocol* (SDP) [98] can be used with RTSP to handle the description of sessions. As described in **Section 4.4**, a session description contains all of the information required for a multimedia session or presentation. With SDP a message consists of three main sections; the *Session* description, the *Time* description and the *Media* description. A session description can also contain more than one time and media description, depending on the number of media streams.

Since SDP can contain multiple instances of Time and Media descriptions, the design of this library is slightly more complicated than the Socket++ library described in the previous section. Each of the descriptions can be modelled using an individual class, however they must all be used *together*, to form a complete session description. Therefore to assist in this design, an *aggregate* class is used. Aggregation is the process of forming a composite object from other (possibly smaller) objects.

Figure 6.3 shows the class diagram for the design of the SDP++ library. It will consist of two main superclasses, the `SDPCommonDesc` class and the `SDPCommon` class. The `SDPCommonDesc` will be inherited by two classes; the `SDPSessDesc` class which will handle the Session description and the `SDPMediaDesc` class which will handle the Media description. The Time description will be handled by the `SDPTimeDesc` class.

The `SDPCommon` class will be the aggregate class and it will be "made-up" (denoted by the diamond shape) of three classes; the `SDPSessDesc` class, the `SDPMediaDesc` class and the `SDPTimeDesc` class. The class diagram also shows, using the black dots with the "`1+`" next to them, that the `SDPCommon` class will have one or more `SDPMediaDesc` and `SDPTimeDesc` classes. This will again depend on the number of media streams.

The `SDPMessage` and the `SDPParser` classes will be derived from the aggregate

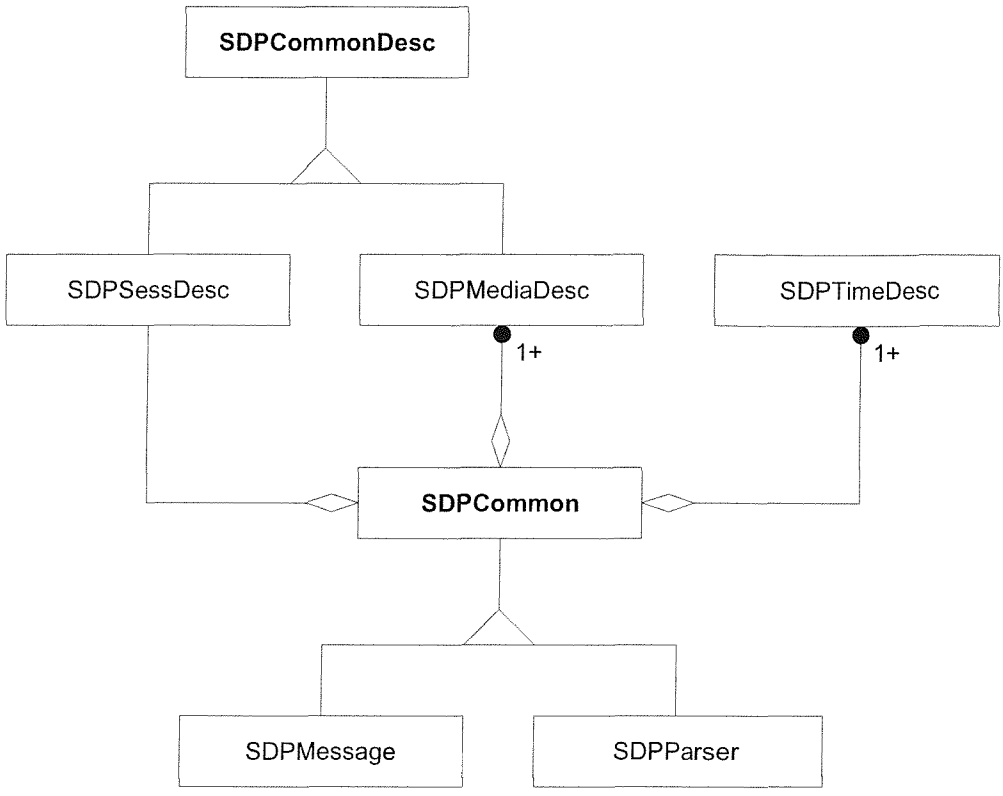

Figure 6.3: The classes for handling SDP.

class and they will be used to create and parse session descriptions respectively. The `SDPMessage` class will be used on the server and it will have functions to dynamically create the messages. The `SDPParser` will be used by the clients, to obtain information on the individual media streams.

6.2.3 The remaining RTSP Framework

The principles used, in the design of the two libraries, were also applied to the design of the remaining framework. Inheritance was used again in a number of different classes, including:

- The `IpMsgHand` superclass and its two derived classes, the `ClientIpMsgHand` and the `ServerIpMsgHand`. These subclasses will be used to handle the parsing of input messages, from the communication channel, on the client and the server respectively.

- The `Eventloop` superclass and its two derived classes, the `ClientEventloop` and the `ServerEventloop`. These subclasses will be used to handle the event loop in the client and the server respectively. These event loops will use specific functions to determine when data is to be read-in or written-out, on the communication and / or data channels.

- The `RTSPSession` superclass and its two derived classes, the `RTSPClientSession` and the `RTSPServerSession`. Clients can only have a single session, with possibly more than one stream. The `RTSPClientSession` class will be used to manage this session, on the client. A server can handle multiple clients and therefore, must be able to handle multiple sessions. As its name suggests, the `RTSPServerSession` will be used to handle each of these sessions, on the server.

Each of the classes mentioned above all have similar class diagrams and **Figure 6.4** is the class diagram for the session management classes.

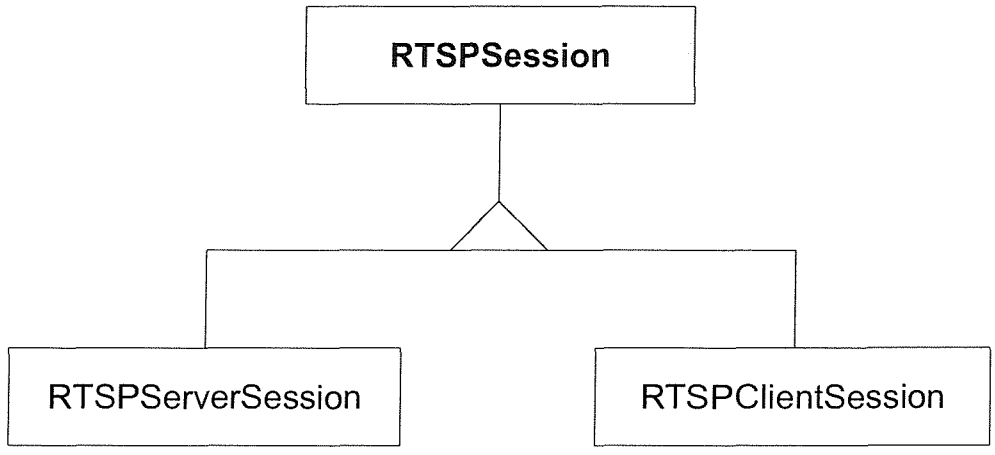

Figure 6.4: The classes for session management.

Another type of class that was used in the design of the RTSP framework was the *abstract* class. Abstraction is the process of focusing on what an entity is and ignoring the implementation details. Therefore an abstract class is used to describe the "look" or the *interface* of a particular object. It has no implementation and therefore an instance of this class can not be created. Abstract classes are used with inheritance to define the interface for the superclass, which can then be refined in the subclasses. It is possible to build an implementation in stages, by using this technique.

For RTSP an abstract class will be used to create a common interface to handle the media streams. There are many different file formats to store media information and each one requires specific data structures and functions to open, read and eventually play this information. A common interface will ensure that each function will have the same declaration and by using inheritance, the function definitions can be implemented in subclasses. If a new media format is created users can define new function definitions, in a subclass, to handle the information.

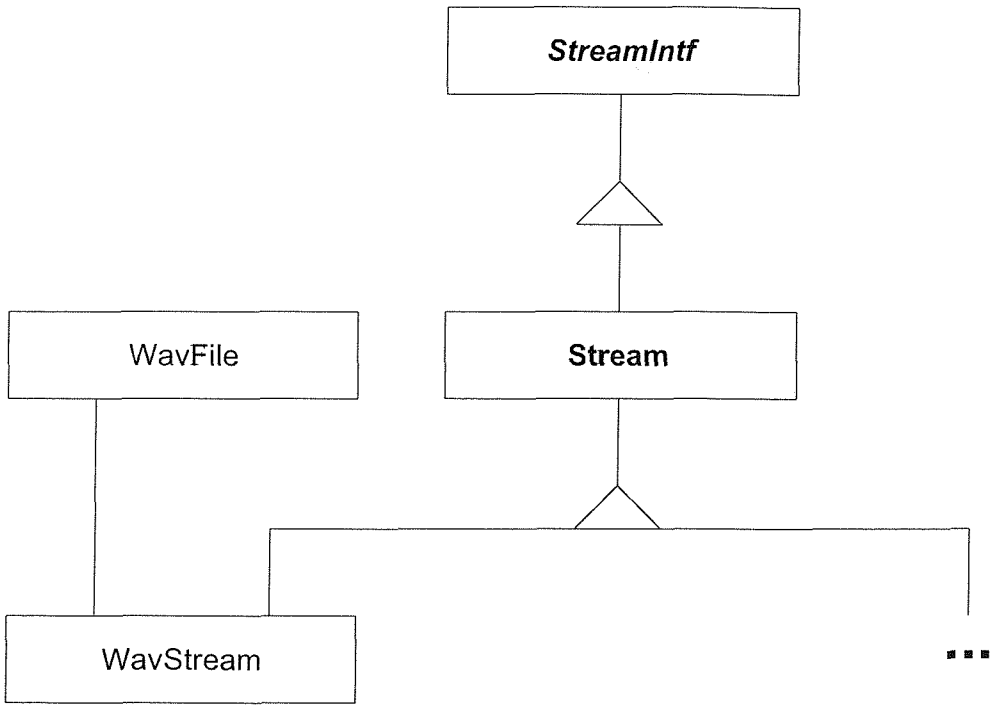

Figure 6.5: The classes for handling streams.

The class diagram for handling the streams is shown in **Figure 6.5**. The `StreamIntf` is the abstract class and it will be used to define the interface for the `Stream` superclass. The `WavStream` subclass will be derived from this superclass and it will be used to handle WAV audio streams. The functionality to handle this audio format, is defined in another class; the `WavFile` class. For handling streams however, the user will only use the functions defined in the `WavStream` subclass, which will in turn call the functions from the `WavFile` class. The ellipses in the diagram are used to show that other possible subclasses exist, although they have not been defined yet. It is possible to add new subclasses to the diagram and hence the implementation, to support different streaming formats.

Abstract classes have been used in the design of several other components for the framework, including:

- The *Streamer* component – which provides the `StreamerIntf` abstract class for the `Streamer` superclass. Two subclasses are derived from this superclass; the `ClientStreamer` and the `ServerStreamer`, which will be used on the client and server respectively. These classes will be used to handle the lower-level functionality of the streams; for example starting and stopping the streams, obtaining the socket and port information for each stream.

- The *ControlBuf* component – which provides the `ControlBufIntf` abstract class for the `ControlBuf` superclass. Two subclasses are derived from this superclass; the `ClientControlBuf` and the `ServerControlBuf`, which will be used on the client and server respectively. These classes will be used to handle the input and output buffers of both the client and the server.

The RTSP design also contains several other individual classes that could not be modelled using inheritance. These include the `Interface` class, which will provide a simple interface for the client and the `Client` and `Server` classes which will be used, in conjunction with the other classes, to define the RTSP client and the server.

6.3 The Temporal Linking mechanism

Several systems already exist, see **Chapter 2**, that can be used to author and manipulate links between different types of media. These include the World Wide Web (WWW) and other Open Hypermedia Systems (OHSs). These systems however, have several limitations. The WWW for instance, embeds the link information into the actual documents, which reduces the transmission time. Users however can not modify these documents; only the original author is allowed to do this. Another problem is link maintenance, especially when the destination point, of a link, is either moved to another location or removed entirely.

Open Hypermedia Systems have also been developed to create, edit and follow links. These systems separate the link information from the documents and usually store this information in link databases or *linkbases*. The authoring or resolving of a link involves creating a new entry in the database or finding and then retrieving the relevant information, respectively. These linkbases however, are usually on separate servers and the OHSs

use their own proprietary protocols, to communicate with these servers. As a result, the majority of these systems can not interoperate.

The Open Hypermedia Systems Working Group (OHSWG), see **Section 2.5**, have been developing the Open Hypermedia Protocol (OHP) to address this specific problem. A subset of this protocol, which was designed to handle navigational links, has been successfully demonstrated at two conferences. It was during the second conference however, that the Southampton members of the group, decided to concentrate on the development of a higher-level structure for interoperability. This structure eventually became the Fundamental Open Hypermedia Model (FOHM) and it is designed to work across three of the most common hypertext domains; spatial, navigational and taxonomic hypertext.

With the new RTSP framework, it was decided that the link information would be kept separate from the audio data. Embedding links within audio streams would require a detailed analysis of the different audio formats. Advanced techniques would have to be used to encode the links into the audio streams, so that they would not affect the acoustic properties of the media. This could be achieved by analysing the audio and replacing specific areas, that contain frequencies beyond the range of the human ear, with the link information. However certain audio formats such as MPEG-1 layer 3 or MP3s, see **Section 3.4.1**, filter out these areas to reduce the size of the files. Further research into this area is beyond the scope of this Ph.D.

If the link information is kept separate from the audio data, it would have to be stored in a linkbase and these linkbases could be on different servers. To ensure that the link information is stored in a non-proprietary interoperable format, it was decided that FOHM would be used to handle the creation, storage and delivery of temporal links. Both FOHM and RTSP also complement each other. FOHM can use RTSP for communication and the protocol can use FOHM for handling open hypermedia information.

FOHM has already been defined using an open standard, the eXtensible Markup Language (XML), see **Section 3.5.5**. XML uses Document Type Definitions (DTDs) to define the structure of documents and the FOHM DTD, with a small modification, will be used to create link databases. The DTD can also be used, with a XML parser, to validate these linkbases. By using FOHM, the RTSP framework will ensure that it will be able to communicate with other FOHM-compliant linkbases.

The original FOHM DTD was designed to describe the elements for a single *association*. An association can also be called a link. A linkbase however will contain several

associations and therefore the DTD was modified, so that multiple associations could be created within a single XML document; effectively creating a FOHM linkbase. The modification was the addition of the linkbase element, `<!ELEMENT linkbase (association)+>`, to the DTD. This element describes how a linkbase contains one or more associations (the '+' after the `association` tag). A complete listing of the new FOHM linkbase DTD is in **Appendix A.2**.

As explained previously, FOHM was designed to work across three hypertext domains; navigational, spatial and taxonomic hypertext. For temporal linking in RTSP however, only the traditional form of navigational hypertext is required. Navigational hypertext allows users to "associate" or "link" related pieces of information together and **Appendix A.3** contains an example of a navigational association in FOHM. This example consists of four main sections and they are:

1. The `id` section – which contains an identifier for the link. This is used to reference this particular association.

2. The `bindingvector` section – which contains the actual link information. It consists of two `binding` elements, one for each end of the link. The first `binding` element contains a `data`, an `AXISLOC` and a `featurevector` element. The `data` element holds the actual data, within a `content` element and in this case it is a RTSP URL to a WAV audio file; `rtsp://ratbert:3048/sinew01/sinew22-01.wav`. The `AXISLOC` element contains several elements, which are used to describe the temporal aspects of this part of the link. In this situation the RTSP URL will be active from 1000ms to 2000ms. The `featurevector` element is used to describe the direction of this `binding` and in this case it is the source.

   The second `binding` element contains the same elements. The `data` element contains another URL, `rtsp://ratbert:3048/rtsp.wav` and the `AXISLOC` element describes how it will be active from 500ms to 2500ms. The `featurevector` element describes how this `binding` is the destination.

3. The `relationtype` section – which describes the relationship between the two `binding` elements mentioned above. The `name` element is used to give the relationship a name and in this case it is called a "link". The `featurespace` element contains a value, which is used to determine the values of each `featurevector` in this association. For navigational links, the `featurespace` value will be "direction" and as explained in **Section 2.5**, this value can be used to define three types

of `featurevector`; source, destination or bi-directional.

4. The `structuretype` section – which describes the Abstract Data Type (ADT) for this type of association. For navigational hypertext the ADT is predefined to be a set.

This association effectively describes a temporal link between two different audio files.

6.4 Extensions to RTSP to support FOHM

RTSP can be used to control the delivery of a single stream or a presentation, containing several streams. Clients control a single stream by using a RTSP URL, see **Section 4.4**, which consists of the location and the filename, on the RTSP server, of the actual media file. When a client opens this URL, a communication channel is formed between the client and the server. When the client issues a request to setup a stream, using the `SETUP` method, the server will create a new stream and a session. The URL and the session identifier will then be used to correlate all subsequent requests, over this channel, with the appropriate session.

With a presentation, RTSP has to control the delivery of several streams, each with its own unique RTSP URL. These URLs are usually stored within a *container* or *presentation* file, which is located on the RTSP server. These URLs can point to files on the same server or to files situated on different machines on the Internet. Depending on the type of control users can either access individual streams, using the stream's URL, or the entire presentation, using the URL to the container file. The URL to the container file is also known as an *aggregate* URL.

To handle temporal linking with audio streams in RTSP, a presentation file will be used. It will contain a URL to an audio file and possibly several URLs to FOHM linkbases, depending on the number being used. An example of a presentation file is shown in **Table 6.1** and it contains a RTSP URL to a WAV audio file, "`rtsp://ratbert.ecs.soton.ac.uk:3048/sinew02/sinew01-22.wav`" and a URL to a FOHM linkbase, "`rtsp://ratbert.ecs.soton.ac.uk:3048/sinew02/sinew02-lb.xml`".

RTSP maintains "session state" in almost all situations, see **Section 4.4**. This ensures that a server will be able to correlate the RTSP requests, from individual clients, to the relevant session. RTSP has four states, see **Figure 4.1** and they are the `INIT` state, the

```
# sinew02:  A simple presentation file in RTSP
rtsp://ratbert.ecs.soton.ac.uk:3048/sinew02/sinew02-22.wav
rtsp://ratbert.ecs.soton.ac.uk:3048/sinew02/sinew02-lb.xml
```

Table 6.1: The container file for a simple RTSP presentation.

SETUP state, the PLAY state and the RECORD state. For the new RTSP framework, the RECORD state has not been defined and no new states have been added. This ensures that the new framework conforms to the specification.

As mentioned above, when a client opens a RTSP URL a communication channel is formed between the client and the server. Requests or methods, such as DESCRIBE and SETUP, are sent over this channel to control the delivery of multimedia streams. For temporal linking these methods will have to be modified, so that they can describe FOHM linkbase files and create linkbase streams. The framework will also be extended to support several new methods, for the creation, display and traversal of links. These are described in more detail, in the following sections.

6.4.1 The AVAILABLE_LINKS method

FOHM linkbases will be too large to stream down, from the server to the client, at the same time as the audio. Each association in the linkbase will be similar in size, to the FOHM navigational association shown in **Appendix A.3**. Therefore the AVAILABLE_LINKS method will be used to download the linkbase(s) before the actual audio stream. The number of linkbases streamed down will depend on the number of RTSP URLs, to these linkbases, in the presentation file.

When a client calls this method, each linkbase will be streamed down to the client and stored in a temporary file. This file will then be validated using the FOHM linkbase DTD and a XML parser. XML defines several ways in which a DTD can be used with a XML document. A DTD can be embedded within the document itself, so that it can be downloaded at the same time as the document. It can also be stored on a separate WWW server and in this situation a URI, see **Section 2.3.1**, to this DTD is embedded into the document instead. To help reduce the size of the FOHM linkbase, its DTD is stored on a separate server.

Once validated, the linkbase(s) will then be parsed by the FOHMLinkbaseReader class, which is designed to retrieve all of the required values from the FOHM linkbase(s).

These values will include the source and destination URIs and any of the time values for these URIs, if they exist. This class will also check the format of the linkbase(s) to ensure, for instance, that a source URI will be followed by a destination URI and not another source URI. These values will be stored within a `FOHMCache` data structure, which will have eight elements:

1. The `id` element – which will be used to identify a link, stored in this structure. This element is required.

2. The `sURI` element – which will contain the source URI of a link. For temporal linking with audio streams, this URI will reference the audio stream in the presentation file. This element is required.

3. The `sBeg` element – which will contain the begin time (in milliseconds) of the source URI. It will be used, by the client, to determine when a link is active; for example a source URI begin time of 3000ms means that this link is not available for use until after 3000ms. This element is optional and if it is not given, then the link will be available throughout the entire presentation.

4. The `sEnd` element – which will contain the end time (in milliseconds) of the source URI. If this value is given, it will be used to determine the length of time (`sEnd` – `sBeg`) a link is active. If the current position of the stream is not between these two values, then this link will not be available for use. This element is optional.

5. The `dURI` element – which will contain the destination URI of the link. See **Section 6.4.2** for more information on this type of URI. This element is required.

6. The `dBeg` element – which will contain the begin time (in milliseconds) of the destination URI. It will be used, by the client, to determine when this URI will be active; for example if this value equals 3000ms and the destination URI is another audio file, then the client will send a message to the server to start streaming this new audio file from 3000ms in. This element is optional and it is only useful for temporal media.

7. The `dEnd` element – which will contain the end time (in milliseconds) of the destination URI. It will be used, by the client, to determine the length of time (`dEnd` – `dBeg`) that this URI is active. If the destination URI begin and end times are 3000ms and 5000ms respectively, then this URI will only be active for 2000ms.

For audio streams, this means that the link will be played back for 2000ms. Again this element is optional and it is only useful for temporal media.

8. The `found` element – which will be used when the client plays the presentation. When a presentation is played, the current position of the stream is checked, to see if it lies in-between the source URI begin and end times. If it does the destination URI and its begin and end times, if they exist, are printed out. The `found` value is then set to "true" because the value of the current position will only change by a small amount (the size of the received packet). It is possible that the duration between the source URI begin and end times will be quite large and therefore, the current position will fall again between these two values. This will result in the link information, for the same source URI, getting printed out time and time again. By checking to see if `found` element is "true", the client will only print out the link once.

Normal Play Time (NPT), see RFC 2326 [53] section 3.6, will be used to set the time information for both the source and destination URIs. A `FOHMCache` structure can be created with just four of its elements; the identifier, the source and destination URI and the `found` element. When this type of link is followed, the "flow-of-control" will pass from the current audio stream (the source URI) to the destination URI.

After the `FOHMCache` structure has been created and initialised, with the values from the linkbase, users will then be able to play the presentation. Users will also be able to play this presentation before the link information has been downloaded. In this situation, the client will call the `AVAILABLE_LINKS` method and initialise the cache, before playing the presentation. Either way, the FOHM linkbase(s) will only be streamed down once. All subsequent requests, to playback the presentation, will use the `FOHMCache` data structure.

During playback, this cache will be searched for relevant links. This only occurs when:

- The URI, of the stream being played, matches one or more of the source URIs in the cache.
- The current position of the presentation is in-between the source URI begin and end times.
- The `found` element of each `FOHMCache` object is set to "false".

If all of these values return true, then the link information will be printed out. After this the *found* element for this object will be set to "true", which ensure that this link is not printed out again.

To get the available links for a presentation, a user would type in a specific command; for example "`a sinew02`". This command will generate the `AVAILABLE_LINKS` request shown in **Table 6.2**. The RTSP URL, in this request, is the aggregate URL to the presentation file "`sinew02`", which is stored on the server. On the server this file will be searched to see if it contains any linkbases and if found, they will then be streamed down to the client.

```
AVAILABLE_LINKS rtsp://ratbert:3048/sinew02 RTSP/1.0
CSeq: 3
Session: 215369823
```

Table 6.2: An `AVAILABLE_LINKS` request.

6.4.2 The `FOLLOW_LINK` method

This method can only be used after all of the link information, from the linkbase(s), have been streamed down to the client and the `FOHMCache` has been initialised, e.g. after an `AVAILABLE_LINKS` request, see **Section 6.4.1**. After the cache has been initialised, users will then be able to follow these links by using their link identifiers, which are stored in the cache. The `AVAILABLE_LINKS` request will have automatically displayed these identifiers upon completion.

When users issue a command to follow a link, using its identifier, the client will search for this link within the cache. Once found, all of the relevant information for this link, e.g. the destination URI and the begin and end times for this URI, will then be used by the client, to decide how to handle the link.

The client can handle three different types of link:

1. A link to non-continuous media, such as text, images and graphics. In this scenario, the link will contain a destination URI that uses the HyperText Transfer Protocol (HTTP), see [108], for the delivery of the media. Since the RTSP server only handles streaming media, the client will handle this type of URI by activating an external browser, e.g. Netscape's Communicator. This Web browser, see **Section 2.3.1**, will use the destination URI to download and then display the media.

There will be no communication between the client and server, when this type of link is followed. Also the begin and end time for the destination URI will not be relevant, since this URI will only be used to reference non-continuous media.

2. A link to a different part or portion of the same presentation. This is analogous to the forward and rewind buttons of a CD player. When a user issues a command to resolve this type of link, a `FOLLOW_LINK` request will be sent to the server. This request will contain the destination URI and if relevant a destination range, which represents both the begin and end time for this URI. On receiving the message the server will immediately pause the presentation and then change the current position, of each of the streams within the presentation, to the new range. When this has occurred, the server will then play the presentation from its new start position, which was obtained from the range value. If no range value is given, then the presentation will be played back from the beginning.

3. A link to a new presentation or stream. With this scenario the existing presentation will be closed down because a new presentation or a session, for a new stream, can not be created within an existing presentation. When a user issues a command to follow this type of link, the client will send a `TEARDOWN` request to the server. The server, on receiving this request, will then close down the streams and free all of the resources associated with the current presentation. After this the client will use the destination URI, from the link, to start a new RTSP session with the server. Several messages will be passed between the client and server, including the `DESCRIBE` and `SETUP` requests. This will ensure that the session is initialised properly and all of the required resources are created and setup. When this is complete, the client will send a `PLAY` request to the server, which causes it to stream the new presentation to the client.

The `FOLLOW_LINK` request can not be used to resolve a link to another linkbase within the same presentation or a new presentation.

Table 6.3 shows two possible examples of this type of request. In both examples, a user would have already typed in the command to follow a link, e.g. "`f link201`". The link identifier, "`link201`", will be used to find the link in the `FOHMCache` data structure. Once found the destination URI, for this link, will be used to form a `FOLLOW_LINK` request. For the first example, this will result in a request to move the current position

of the WAV audio stream, `rtsp://ratbert:3048/sinew02/sinew22_01.wav`, to 4 seconds in from the beginning. This will either "rewind" or "forward" the stream, depending on the current position.

The second example follows a link to a new audio stream, `rtsp://ratbert:3048/sinew22_02.wav`. This will cause the original presentation to be closed down and the new stream to be setup and initialised. This link also contains a destination range from 3 to 5 seconds, which will result in this stream being played back from 3 seconds in, for a duration of 2 seconds (5s - 3s).

```
// Following a link within the same presentation.
FOLLOW_LINK rtsp://ratbert:3048/sinew02/sinew22_01.wav RTSP/1.0
CSeq: 4
Session: 215369823
Range: npt=4-

// Following a link to a new stream.
FOLLOW_LINK rtsp://ratbert:3048/sinew22_02.wav RTSP/1.0
CSeq: 5
Session: 215369823
Range: npt=3-5
```

Table 6.3: Examples of the `FOLLOW_LINK` request.

6.4.3 The `CREATE_LINK` method

This method will be used to create links from an audio stream within a presentation. It can only be used when the client and the server are in the `READY` state. Links can not be created whilst the presentation is being played back; in this situation the client and server will be in the `PLAY` state.

When a user wants to create a link, the client will check to see if the current session contains either a single stream or a presentation. If the session contains a single stream, then the client will return an error because temporal linking in RTSP will require more than one stream; to handle the audio and the FOHM linkbases. Therefore a single stream can not be used to display, follow and create links. A presentation however, can be used to control multiple streams, see **Section 6.4**.

If the session contains a presentation then the client will check to see if:

- The presentation already contains a linkbase. In this scenario, the client will check to see if a link, with the user's new link identifier, already exists in the `FOHMCache`. If it does, the client will inform the user that this link already exists and no further action will be taken. Otherwise, a `CREATE_LINK` request will be sent to the server. This request will contain the link identifier, the source URI, the destination URI and any range, i.e. the begin and end time values, for these URIs.

  Once received, the server will find the first linkbase within the presentation and seek to the end of this file. The server will then create a new FOHM association, with the URIs and the values from the request, and append it to the end of this linkbase. When this is complete, a reply will be sent to the client.

  The client, on receiving the reply, creates a new `FOHMCache` object and adds the URIs and the values, from the original request, to the object. This object is then inserted in to the original cache of links. This ensures that the user can see and use the link immediately, without having to call the `AVAILABLE_LINKS` method, see **Section 6.4.1** first. However if this command is used then the current cache is removed, a new cache is created and then all of the links, including the new ones, are streamed down to the client. The cache is created in the normal way and it will now contain all of the new links.

  The display of the link information during the playback of the presentation, is unaffected by the `CREATE_LINK` method because it relies on the cache. As described in the previous paragraph, any new links are immediately inserted into the cache, when the client receives `CREATE_LINK` reply.

- The presentation does not contain a linkbase. In this scenario, a `CREATE_LINK` request, with the source and destination URIs and their range values, will be sent to the server. The server, when it has received this request, will create a new linkbase file. The URI to this linkbase will be appended to the presentation's container file, which consists of all of the URIs to the files and hence the streams, used within this presentation. After this, the new linkbase will be opened and a new FOHM association, containing the source and destination URIs and their ranges, will be inserted. The server will then send a reply to the client.

  At this stage however, the server will have only modified the container file, created the linkbase and inserted the new link information. A stream for downloading this information, to the client, will not have been created. In fact the server will still be dealing with the original presentation, which does not contain a linkbase.

Therefore this presentation will have to be closed down and re-opened, so that a new stream for the linkbase will be created and initialised.

On receiving the reply, from the server, the client will close the presentation by sending a TEARDOWN request. After the server has notified the client that this request has succeeded, the client will then send several other requests, e.g. DESCRIBE, SETUP etc., to re-open the presentation. These requests will return information, to the client, about all of the streams in the presentation, including the new linkbase stream. The client will use this information to create and initialise all of the required resources. When this is complete, the client will then send an AVAILABLE_LINKS request, see **Section 6.4.1**, to download all of the link information. The cache will then set up in the normal way.

If the user decides to create more links, then the link information will be inserted into the new linkbase, which has just been created. The previous item, in this list, describes this in more detail.

To create a link from an audio stream a user must type-in a specific command and the format of this command is shown in **Table 6.4**. It has four parameters and they are the link identifier (linkId), the source URI (source_URI), the source range (source_range), the destination URI (dest_URI) and the destination range (dest_range). The source range and the destination range are optional.

```
// The user command for creating a link.
c linkId source_URI [source_range] dest_URI [dest_range]
```

Table 6.4: The format of the user command to create a link.

There are four different types of CREATE_LINK request and an example of each of these is shown in **Tables 6.5 - 6.7**. Each table contains the user command to create the link and the resulting CREATE_LINK request, that will be sent from the client to the server. Each of these links has a specific source URI and they all point to a WAV audio stream (sinew22_01.wav) within the presentation. The presentation file is the first part of the file name for the source URI, e.g. sinew01. Therefore the URI to this presentation will be rtsp://ratbert:3048/sinew01.

In **Table 6.5**, link3 and link4 have a destination URI to a Web page (sinew.html) and an image (church1.jpg), respectively. If either of these two links were followed, see **Section 6.4.2**, then a Web browser would be activated and the Web page or the image would be displayed. Since link3 does not have any ranges, source or destination, then

the link is available throughout the entire presentation. link4 however, has a source range from 1000ms to the end of the audio file. Therefore this link will be available from 1000ms in to the presentation, to the end.

```
// The user command to create a link to a Web page.
c link3 rtsp://ratbert:3048/sinew01/sinew22_01.wav
   http://www.ecs.soton.ac.uk/~cnhr/sinew.html

// The resulting CREATE_LINK request.
CREATE_LINK rtsp://ratbert:3048/sinew01/sinew22_01.wav RTSP/1.0
CSeq: 3
Session: 215369823
LinkId: link3
Destination-URI: http://www.ecs.soton.ac.uk/~cnhr/sinew.html

// The user command to create a link to an image.
c link4 rtsp://ratbert:3048/sinew01/sinew22_01.wav npt=1-
   http://www.ecs.soton.ac.uk/~cnhr/church1.jpg

// The resulting CREATE_LINK request.
CREATE_LINK rtsp://ratbert:3048/sinew01/sinew22_01.wav RTSP/1.0
CSeq: 4
Session: 215369823
LinkId: link4
Source-Range: npt=1-
Destination-URI: http://www.ecs.soton.ac.uk/~cnhr/church1.jpg
```

Table 6.5: A CREATE_LINK request to a Web page and an image.

The third CREATE_LINK example, link5, is shown in **Table 6.6** and it contains a destination URI to a single stream (sinew22-1.wav). This stream is not part of the same presentation because it does not have the container file name in its URI. Therefore if this link is followed, the original presentation will be closed down and the single stream will be setup and initialised. This link also has a destination range, from 2500ms to 4000ms. When the user issues a PLAY command, the stream will be played back from 2500ms in, for a duration of 1500ms (4000ms – 2500ms).

The fourth link example, link6, is shown in **Table 6.7** and it contains a destination URI which is the same as the source URI. When this link is followed, it will move the current position for playback, within the presentation. This is similar to the forward and rewind buttons on a CD player. This link can be followed from a 1000ms in to the presentation; the begin value of the source range. If the link is followed, the playback

```
// The user command to create a link to a new stream.
c link5 rtsp://ratbert:3048/sinew01/sinew22_01.wav
   rtsp://ratbert:3048/sinew22-1.wav npt=2.5-4

// The resulting CREATE_LINK request.
CREATE_LINK rtsp://ratbert:3048/sinew01/sinew22_01.wav RTSP/1.0
CSeq: 5
Session: 215369823
LinkId: link5
Destination-URI: rtsp://ratbert:3048/sinew22-1.wav
Destination-Range: npt=2.5-4
```

Table 6.6: A CREATE_LINK request to a new stream.

of the presentation is paused, the current position is moved to the begin time of the destination range (2500ms) and the play back is resumed, from this new position. In this example, the destination range also has an end time of 4000ms and therefore the presentation is only played back for 1500ms (4000ms – 2500ms).

```
// The user command to create a link to a point
// within the same presentation.
c link6 rtsp://ratbert:3048/sinew01/sinew22_01.wav npt=1-
   rtsp://ratbert:3048/sinew01/sinew22-01.wav npt=2.5-4

// The resulting CREATE_LINK request.
CREATE_LINK rtsp://ratbert:3048/sinew01/sinew22_01.wav RTSP/1.0
CSeq: 6
Session: 215369823
LinkId: link6
Source-Range: npt=1-
Destination-URI: rtsp://ratbert:3048/sinew01/sinew22-01.wav
Destination-Range: npt=2.5-4
```

Table 6.7: A CREATE_LINK request to a point within the same presentation.

6.5 The Implementation of the Tool

The original reference implementations of the RTSP framework were created using the C programming language. As described in **Section 6.2**, this language was chosen because of its portability and its popularity; most operating systems have at least one C compiler. For the new RTSP implementation however, it was decided that an object-oriented language would be used instead.

Several object-oriented languages exist including C++, Eiffel and Smalltalk. C++ however, is one of the most popular object-oriented programming languages available and Stroustrup [10] explains how it was originally designed to extend C by supporting data abstraction, object-oriented programming and generic programming. It has recently become an ISO standard and code written in C++ is more flexible, modular and it can be easily re-used, using a mixture of techniques including inheritance, see **Section 6.2.1**.

During the ISO standardisation process more advanced features were also developed for C++, including the *Standard Template Library* (STL). The STL provides generic container classes such as strings, linked lists (single and double), vectors (similar to arrays) and queues. This library is specifically designed so that the implementations of these classes will be fast and efficient. It also assists in program development because it reduces the time required to create and debug code; for example all of the memory management is contained within these classes, so that users do not have to look for memory leaks or related problems.

C++ is also portable and it has been used on a number of operating systems. As a result of this and the details mentioned above, it was decided that this language would be used, to implement the new RTSP framework.

Section 6.2 also describes how during the design of the RTSP framework, the IETF were also developing the next version of the Internet Protocol; IPv6, see **Section 4.5**. As more operating systems started to implement this protocol, it was decided that the framework would also support IPv6. At this time, the Linux operating system was readily available and it already had basic support for this new protocol. Therefore Linux was used for the development of this new tool. This system not only provides a decent working environment, it also provides the source for the IPv6 libraries. These were used, when needed, for debugging.

Linux supports several programming languages including C, C++ and Java. In most Linux distributions, compilers for these languages are provided by the *GNU Compiler Collection* (GCC) developed by the *Free Software Foundation* (FSF). The GCC also contains an implementation of the STL which was used, in conjunction with the GCC C++ compiler (g++), for the development of the temporal linking tool.

Section 6.3 describes the temporal linking mechanism for the new RTSP framework. It explains how the original FOHM DTD was modified, so that it could be used to create

link databases. To ensure that these link databases conform to the FOHM DTD, a validating XML parser is used. Several XML parsers and their Application Programming Interfaces (APIs) have already been developed, for different languages including C, C++ and Java. For the new RTSP framework however, it was decided that a C++ XML parser and its API would be used. This ensures that the same language would be used throughout the RTSP implementation.

IBM's *XML for C++ Parser* (XML4C++) [18] provides an API and a shared library to parse, generate, validate and manipulate XML documents. XML4C++ is a high performance parser and it is used specifically, with this framework, to validate the linkbases used within presentations. On the RTSP server, these linkbases will be validated each time the streams, in a presentation, are created and initialised. This ensures that they conform to the FOHM DTD. On the client however, the XML4C++ parser is only used to validate the linkbases, after an `AVAILABLE_LINKS` request, see **Section 6.4.1**. This request downloads the linkbases, from the server to the client, to temporary files and the parser will be used to ensure that these files were downloaded correctly. An error will occur if these temporary linkbase files are invalid.

The XML4C++ parser can be used to retrieve specific elements and their data from XML documents. This parser therefore, could be used to retrieve the information from the FOHM linkbases, such as the source and destination URIs. It was discovered however, that this can be a complicated process, especially if the XML document is large and if it has a complex structure. Therefore it was decided that a new class would be developed, to handle the retrieval of this link information. This class is called the `FOHMLinkbaseReader` class and it is described in more detail in **Section 6.4.1**.

6.6 Summary

This chapter has described the development of an Open Hypermedia tool for temporal linking with audio streams. A new implementation of the RTSP framework has been used to handle these audio streams, whilst FOHM has been used to handle the storage of the link information. This new framework has also been extended with several new methods, which use FOHM, for the display, following and creation of temporal links.

The Fundamental Open Hypermedia Model (FOHM), as mentioned previously, is an abstract structure that can be used across three main hypertext domains; navigational, spatial and taxonomic hypertext. The model itself however, is not restricted to these

domains. The FOHM structure was the result of research into interoperability between "Open" Hypermedia Systems (OHSs). Originally this research produced the Open Hypermedia Protocol (OHP) that could be used, with these systems, to improve interoperability between the clients and the servers. FOHM however, concentrates on the higher-level structure that could be used across the domains mentioned above. As a result it does not provide a communication protocol, it just provides an abstract structure for interoperability.

For the new RTSP framework, the traditional form of navigational hypertext (a link containing two endpoints), was used. By modifying the FOHM structure, it was possible to develop interoperable linkbases that could be used, to store navigational links with temporal information.

The underlying network protocols, that RTSP uses, were also changed. Traditionally IPv4, see **Section 4.2**, was used. However, the IETF have also developed the next generation of Internet Protocol; IPv6, see **Section 4.5**. This protocol will eventually replace IPv4 and it also backwards compatible with IPv4. Therefore, the new RTSP framework supports both protocols, so that the clients will be able to communicate with the servers, using either one.

The previous chapter has shown how an existing OHS, Microcosm, can be modified so that it supports audio streams. This was achieved by modifying its audio tool, the SoundViewer, to support an existing implementation of RTSP. This chapter however, has shown how RTSP can be extended to support open hypermedia, using an interoperable linkbase format called FOHM. In fact, as mentioned previously, FOHM and RTSP complement each other. RTSP provides a communication protocol for FOHM and FOHM provides an interoperable linkbase format for RTSP. Therefore RTSP and FOHM have both been extended. A demonstration of this new framework is shown in **Section 7.2**.

Chapter 7

Case Study and Evaluation

7.1 Introduction

The previous chapter discussed the development of a new Open Hypermedia Tool that allows users to display, traverse and create temporal links with audio streams. This was achieved by developing a new version of RTSP and then extending this protocol to support FOHM.

To demonstrate this new system a simple case study was developed, using audio, text and images from archives stored at the University of Southampton. This chapter describes this case study in more detail and it also provides an analysis of the research carried out in this thesis; specifically how the two applications, the streaming SoundViewer tool and the Open Hypermedia tool, have achieved the research objectives.

7.2 The Open Hypermedia Tool case study

To test the new Open Hypermedia tool for temporal linking with audio streams, a simple demonstration was developed. As explained in **Section 5.2.3**, the University of Southampton stores the Winston Churchill archives and it was decided that several audio files, from this archive, would be used. These files are stored on the new RTSP server.

The server itself is activated by calling the "`rtsp-server`" executable, with a specific *configuration* file. As its name suggests, this file contains information to configure the server and it includes:

- The *control stream* type – which defines the delivery mechanism for the control channel, such as TCP or UDP. By default it is set to TCP.

- The *protocol family* – which defines the protocol family to use, such as IPv4 or IPv6. By default it is set to IPv6.

- The *port number* – which defines the port number for connection. For RTSP there is a predefined number 554. The Linux operating system however, requires user-defined applications to allocate port numbers above 1024. This maintains security and ensures that these user-defined applications will not interfere with other system-dependent applications. By default it is set to 3048.

- The *delivery stream* type – which defines the delivery mechanism for the data channel(s), for example unicast, multicast or both. By default this is set to unicast.

- The *basepath* value – which defines the location of the actual data, such as the audio files and the presentation files.

During execution the server will only output requests that it receives, the responses to these requests and any errors. This information is usually redirected to a log file.

To activate the RTSP client, the "`rtsp-client`" executable is called. This file has two command line options and they are the protocol family which is described above and a RTSP URL. By default, if no protocol family is defined, the client will revert back to IPv4. The URL is used to immediately open a connection to the server and describe a particular stream or a presentation file. If this URL is not given, then the client will just execute. To connect to the server, after the client is running, the user will have to type in the command to open a RTSP URL.

For this demonstration a simple presentation file, containing the streams shown in **Table 6.1**, was created. As described in **Section 6.4**, one of these streams is an RTSP URL to an audio file and the other is a URL to a FOHM linkbase. When the user either types in the RTSP URL to the presentation file (`rtsp://ratbert.ecs.soton.ac.uk:3048/-sinew02`) on the command line or uses the built-in open command (the 'o' command), a connection will be formed between the client and the server (`ratbert.ecs.soton.ac-.uk:3048`). The presentation file `sinew02` will then be described, using the Session Description Protocol (SDP), see **Section 4.4**. This is shown in **Figure 7.1**.

This RTSP client is using the IPv6 protocol family. This is shown by the "`IP6`"

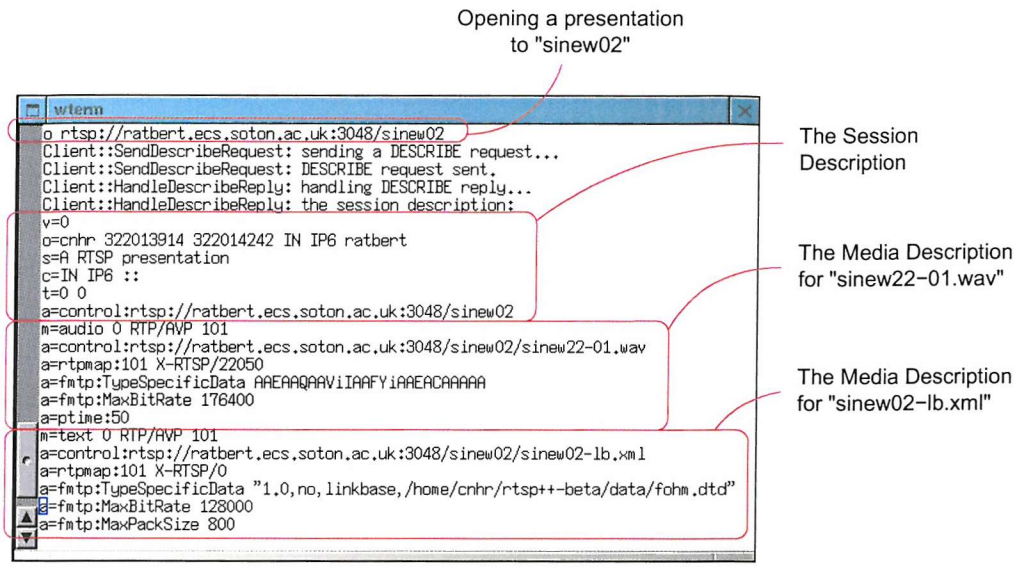

Figure 7.1: The RTSP client after opening a new presentation.

symbol used in the session description. Two media descriptions are used to describe both the audio stream (`sinew22-01.wav`) and the FOHM linkbase stream (`sinew02-lb.xml`). The extra information in these media descriptions, such as the "`TypeSpecificData`" and the "`MaxBitRate`", are used to help create and initialise the streams. This occurs when the user types in the command to setup the presentation (the '`s`' command), which results in the `SETUP` request being sent to the server.

When the presentation has been setup, the user can ask the server to either display the available links for this presentation, create new links or just play the presentation. If the user types in the command to display the links (the '`a`' command), the `AVAILABLE_LINKS` request is sent to the server, see **Section 6.4.1**. This will download the linkbase (`sinew02-lb.xml`) to the client, retrieve and store the relevant information in the `FOHMCache` and then display the links. The links stored in this linkbase, are shown in **Figure 7.2**.

The linkbase contains three links from the audio stream (`sinew22-01.wav`) to an image via "`link1`", to a point within the same presentation via "`link2`" and to a new audio stream via "`link3`". The first link is available from 500ms to 1000ms, the second link is available from 1000ms to 3000ms and the third link is available from 1500ms to 2000ms. On playback of this presentation each of these links will be displayed, when the current position within the audio stream (the source URI) falls in between these two

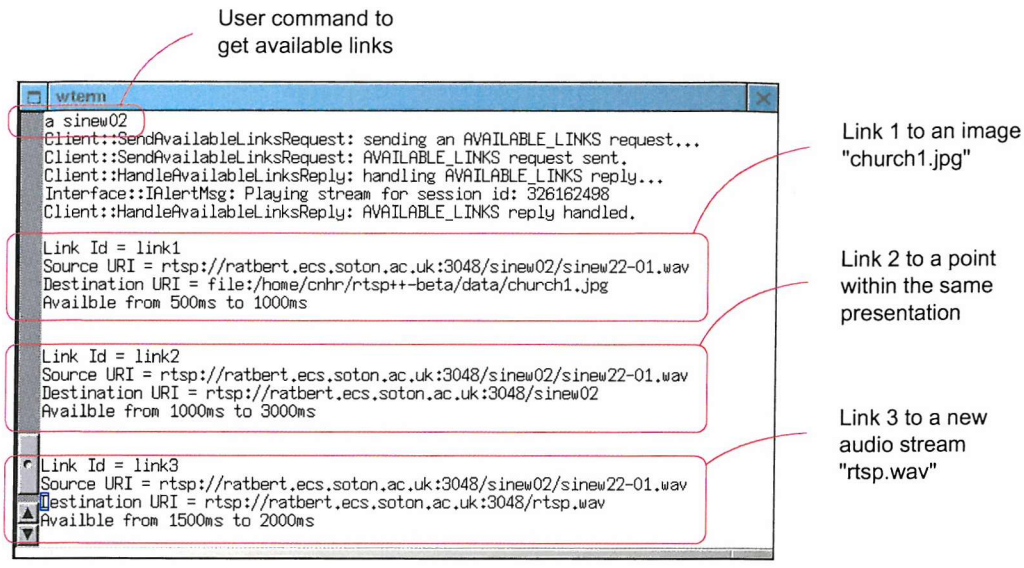

Figure 7.2: The RTSP client after the AVAILABLE_LINKS request.

values (the source URI begin and end times); e.g. "link1" will be displayed when the current position is in between 500ms and 1000ms.

After displaying the available links, the user can either create a new link, follow an existing link or playback the presentation. To create a new link the user will type in the create link command (the 'c' command) with the relevant values such as the link identifier, the source and destination URIs and optionally a begin and end time for each URI. This will generate a CREATE_LINK request, see **Section 6.4.3**, which is sent to the server. **Figure 7.3** shows the user creating a new link from the audio stream (sinew22-01.wav) to a HTML page (http://ernie.ecs.soton.ac.uk/data/incoming/cnhr/Churchill/sinew.html). This link has a source URI begin and end time of 2s to 4.5s (2000ms to 4500ms) and therefore it will be available from 2000ms in to the presentation for a duration of 2500ms (4500ms – 2000ms).

Once this new link ("link4") has been created, the user can then type in the command to get the available links again. This will only be used, by the user, to ensure that the link has been created properly. By calling this command, the FOHM linkbase will not be downloaded again because this new link information will be automatically stored in the FOHMCache, see **Section 6.4.1**. The FOHM linkbase will be updated on the server but there is no need to download it again. **Figure 7.4** displays the results of using this command again.

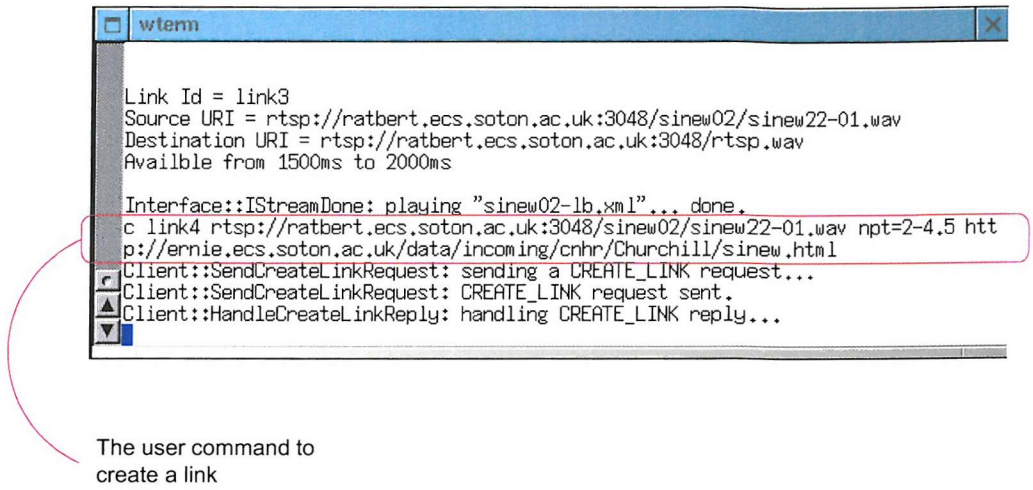

Figure 7.3: The command used to create a new link.

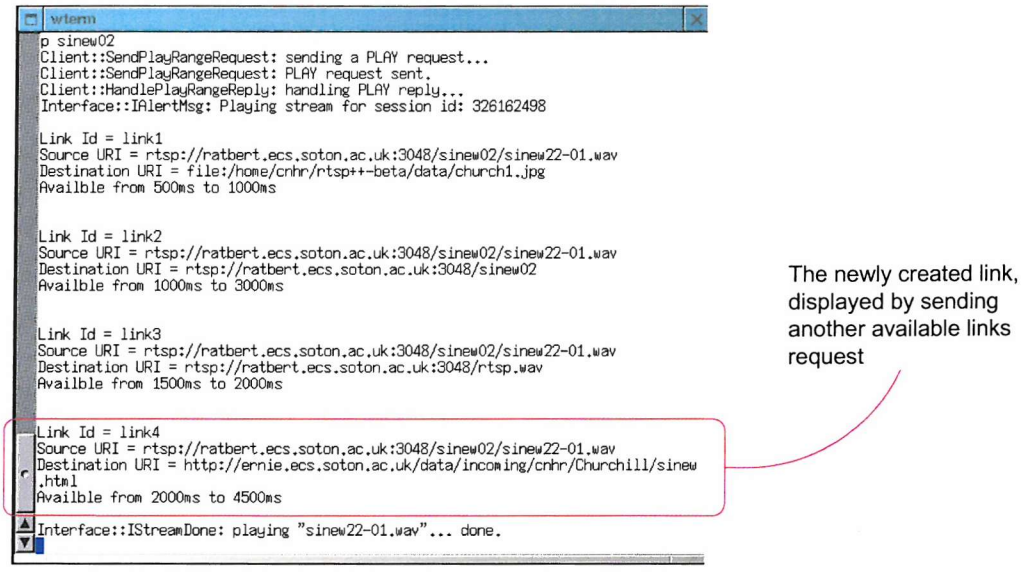

Figure 7.4: The AVAILABLE_LINKS request, after creating a new link.

To follow a link the user types in the follow link command (the 'f' command) with the link identifier. This will be handled either by the client or by the server, depending on the type of link. In this presentation, "link1" and "link4" are links to an image and a HTML page respectively. Therefore the client will be used to handle these links, by executing an external program to display them. "link2" and "link3" are links to a position within the presentation and to a new RTSP stream respectively. In this situation a FOLLOW_LINK request, see **Section 6.4.2**, will be sent to the server, to handle the streams

in the required way.

Figure 7.5 displays the results of following "`link4`". In this situation the client executes an external program (the Netscape Web browser) and displays the Web page.

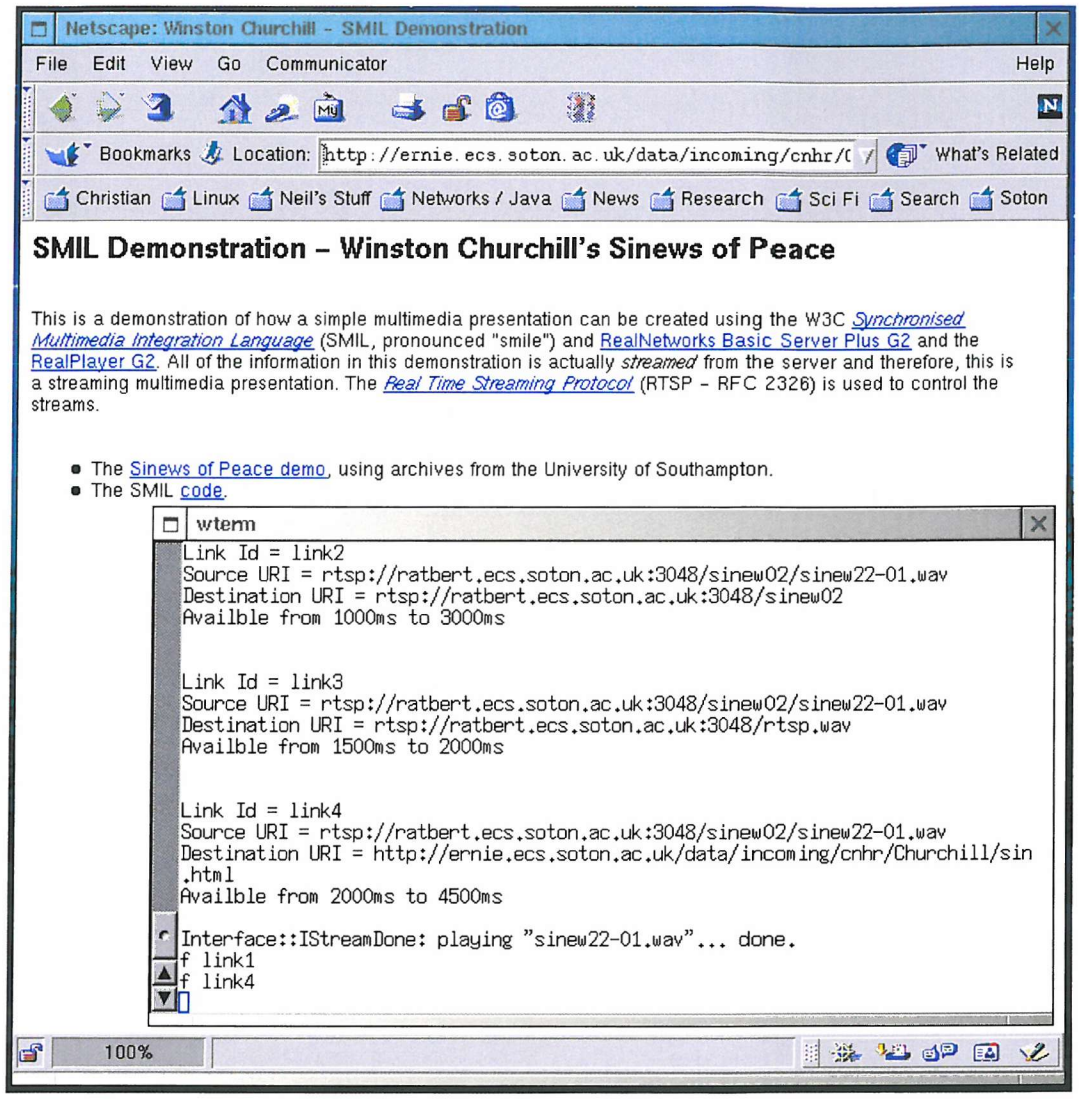

Figure 7.5: The results of following "`link4`".

7.3 Evaluation of the Research

There were three main objectives of this research, see **Chapter 1**, and they were:

1. To extend an existing Open Hypermedia tool to support audio streams,

2. To extend a streaming media protocol to support Open Hypermedia and

3. To enhance the Open Hypermedia mechanism, FOHM, with a communication protocol.

These objectives have all been achieved and an analysis of each objective is discussed in the following sections.

7.3.1 Objective 1: The streaming SoundViewer Tool

During the initial stages of the research, it was realised that the majority of Open Hypermedia Systems (OHSs) had very little support for the audio domain. Some systems such as Microcosm and HyperWave did have support for audio, see **Sections 3.5.1** and **3.5.3** respectively. However these tools could only create links to and from audio files stored on the same machine as the OHS. The World Wide Web (WWW) on the other hand was already using streaming protocols, see **Chapter 4**, to stream different types of media to users on the Internet.

Therefore it was decided that the first objective of the research was to develop an initial "proof-of-concept" tool, that could combine an existing Open Hypermedia application with a streaming media protocol. The code for the SoundViewer tool was readily available and the IETF had already released a reference implementation of the Real Time Streaming Protocol (RTSP), see **Section 4.4**. By combining this protocol with the SoundViewer, the first objective was achieved.

When developing the original implementation of the SoundViewer, see **Section 5.2**, Goose and Hall [118] describe some of the complications that arose; specifically how to create a tool that allows users to graphically author links to and from audio. The audio domain by its very nature can not be seen and therefore it does not have a unique visual representation. As a result developers have created different "visual metaphors" for audio and they include the scrollbar method, waveforms and musical scores.

For Microcosm's SoundViewer tool an enhanced scrollbar, see **Section 5.2.1**, was developed. Although the interface to this tool is dated, it uses the Windows 3.1 "look-and-feel", this graphical metaphor is ideal. It allows users to visually create, display and traverse links to and from an audio file. Links are displayed as shaded rectangles in the *detail* window, which is a zoomed-in view of a particular section of the audio. Users can follow links by clicking on these rectangles.

To create the streaming SoundViewer, the tool had to be extended to support RTSP, so that it could receive audio streams from a RTSP server. The SoundViewer effectively becomes a RTSP *client*, see **Section 5.3**. When a user opens a SoundViewer project, that contains a RTSP URL to an audio file stored on a server, the tool creates a network connection with that server. The server then sends information about the file to the client. The SoundViewer uses this information to initialise its enhanced scrollbar, so that the length of the scrollbar represents the length of the audio file. Playback of the file and hence the delivery of the audio stream, can then begin.

The original SoundViewer has a modular design and an abstract layer, that separates the user from the lower layer audio device commands, see **Section 5.2.2**. The user selects the type of audio to playback and the tool decides which lower layer audio functions are used. This approach was used to reduce the complexity of developing the original program and it also helped when the tool was extended to support RTSP. A new module was developed to handle all of the RTSP functionality and this was then "inserted" into the relevant section of the code. When users select an audio stream to playback, the tool will use the new RTSP module to handle the stream, see **Section 5.3.1**.

To demonstrate the original tool and the new streaming version, two case studies were developed, see **Sections 5.2.3** and **5.3.2** respectively. It was decided, during the development of the streaming SoundViewer, that the interface would not be changed; only the underlying functionality was modified to support audio streams. This also ensured that the streaming SoundViewer had a consistent "look-and-feel" and that the same tool could be used to playback local audio files, as well as audio streams. As a result both versions look exactly the same. However the streaming version displays, in its title bar, the name of the RTSP server it is using, see **Figure 5.9**.

The streaming SoundViewer tool has proved that it is possible to combine streaming media protocols, made popular by the World Wide Web (WWW), with Open Hypermedia Systems. A brief demonstration of this tool was given at HyperText 98, see DeRoure et al. [23] and the concepts, behind the development of this tool, have recently been mentioned in a paper by Page et al. [84].

This first objective however, has focused on extending an existing Open Hypermedia tool to support a streaming media protocol. The next stage of the research was to see if the streaming media protocol could be extended to support Open Hypermedia; in effect turning the first objective around. This second objective is discussed and evaluated in the next section.

7.3.2 Objective 2: Extending RTSP to support Open Hypermedia

The next stage of the research focused on the streaming media protocol itself; to see if could support the concepts of Open Hypermedia. Again the Real Time Streaming Protocol (RTSP) was used because a reference implementation already existed and it provided all of the functionality required to handle audio streams. However the reference implementation, developed by the Internet Engineering Task Force (IETF), was based on an earlier version of the standard and it only had enough functionality to handle simple streams. Therefore to support the actual standard, it was decided that a new implementation of the protocol was required, see **Section 6.2**.

During the development of this new implementation or *framework*, the IETF were also creating the next generation of Internet Protocol (IPv6). This protocol will replace IPv4, the current Internet Protocol and it provides several new enhancements including expanded addressing capabilities and more importantly a *flow label*. This label can be used to improve the real-time delivery of streaming media, see **Section 4.5**. Since RTSP runs "on-top-of" the current Internet Protocol and IPv6 is also backwards compatible with IPv4, it was decided that the new framework would use IPv6. However the flow label is still under development and therefore it could not be used in the new implementation.

After the framework had been developed, the next stage of the research was to extend the protocol to support Open Hypermedia and temporal linking. To complete this portion of the work and hence achieve the second objective, two important decisions had to be made:

1. How to modify the protocol – so that the resultant implementation would still be compliant with the RTSP specification.

2. The type of Open Hypermedia System to use – so that users could create, follow and display the available links.

To conform to the RTSP specification, new implementations of the protocol must contain a state machine, to maintain the session state and predefined methods, to handle the communication between the client and the server, **Section 4.4**. These methods are also used to control the delivery of the streams. Developers however are not allowed to change the state machine; to add new functionality they can include new methods and / or modify the existing ones. Therefore to ensure that the new framework was compliant,

it was decided that the existing methods would be modified and several new methods would be created, to support temporal linking, see **Section 6.4**.

As described in **Section 6.3**, a number of systems already exist that can be used to author and manipulate links between different media types. However several systems, such as the WWW, either embed the link information within the documents or use proprietary protocols to communicate with separate links servers. Embedding the link information within the audio was not a viable solution; separating the links from the audio ensures that the framework maintains "openness". Therefore to avoid embedded links and proprietary protocol formats, it was decided that a different mechanism would be used.

Initially two systems were considered, the Open Hypermedia Protocol (OHP) and the Fundamental Open Hypermedia Model (FOHM), see **Section 2.5**. OHP however already provides a mechanism for communication and it is quite large. FOHM on the other hand is just a data model for representing different types of hypermedia. Currently it supports navigational, spatial and taxononmic hypertext and it has the functionality to support more. For the new RTSP framework, it was decided that FOHM would be used because it can effectively describe and store the temporal link information.

FOHM was also used for another reason; it is defined using an XML DTD. XML [152] is a W3C standard and it is steadily growing in popularity. It is a versatile markup language and it will eventually replace HTML as the markup language for the Web. Several XML parsers already exist and one of these is IBM's XML4C++ parser, which was used by the new framework to validate FOHM linkbases.

To demonstrate the new Open Hypermedia tool, a case study was developed, see **Section 7.2**. This study shows how users can display, follow and create temporal links, using the new methods created for the framework. These methods were called the `AVAILABLE_LINKS` method, the `FOLLOW_LINK` method and the `CREATE_LINK` method, respectively and they are described in more detail in **Sections 6.4.1**, **6.4.2** and **6.4.3**.

By extending RTSP to support FOHM the second objective has been achieved. The new methods, mentioned above, allow users to manipulate temporal links and FOHM provides the functionality to effectively describe and store the link information. The most important aspects of this objective however, are that the links can be stored separately from the media and that they are streamed to the client. Although they are not streamed at the same time as the audio, see **Section 6.4.1**, they are still streamed to the client and stored ready for use. Several systems already exist that allow users to create and stream

multimedia presentations to users. These include MPEG and SMIL, see **3.4.1** and **3.5.5**, respectively. However any link information is usually embedded or encoded into the documents or file formats.

This research has been presented at OHS7[1] in Århus, Denmark and PGNET 2001[2] in Liverpool. The research has also been published, see Ridgway and DeRoure [100]. Page et al. [84] have taken the concepts of this research a stage further by examining how metadata, which contains link information, can be streamed with different types of media. This technique is called *continuous metadata.*

7.3.3 Objective 3: A communication protocol for FOHM

The final stage of the research focused on extending the Fundamental Open Hypermedia Model (FOHM) with a communication protocol. By itself FOHM can be described as an interoperable linkbase format for different hypertext domains. It does not provide a protocol for communication.

FOHM is a derivative work of the Open Hypermedia Protocol (OHP), which was designed to overcome the problems of proprietary protocols used between OHSs and their linkbases, see **Section 2.5**. When OHP is used, instead of these proprietary protocols, different OHSs can communicate with different linkbases. However this protocol has steadily grown in size, to accommodate different hypertext domains. As a result it was realised that one protocol can not handle all of the functionality required. Therefore the protocol was separated into smaller components, each designed to handle a specific domain, e.g. OHP-Nav for navigational hypertext and OHP-Space for spatial hypertext.

A number of researchers realised however that it was possible to separate the hypertext domains from the communication protocol. A model was developed, FOHM, that encompasses these domains and the way in which they interact. It can also be extended to support new domains when they are developed.

The second objective of the research was to develop an Open Hypermedia tool for temporal linking with audio streams. This objective was achieved, see **Chapter 6** and **Section 7.3.2**, by extending RTSP to support FOHM. As mentioned previously FOHM by itself, does not provide any form of communication mechanism. The new RTSP

[1] In conjunction with Hypertext 2001.

[2] The 2^{nd} Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting.

framework however does provide this mechanism; it uses IPv4 or IPv6 as the underlying communication protocol and the Real-time Transport Protocol (RTP), see **Section 4.3**, for delivery of the streams. Therefore by combining RTSP with FOHM the third objective has been achieved. In fact the combination is mutually beneficial; FOHM provides the Open Hypermedia functionality for RTSP and RTSP provides the communication mechanism for FOHM.

FOHM is defined using a XML DTD, which is used to describe the link information. With the new RTSP framework this information and hence the XML is streamed to the client, parsed and then stored ready for use, see **Section 6.4.1**. This technique for delivering XML is similar to another method, which has recently been developed by the World Wide Web Consortium (W3C): the *Simple Object Access Protocol* (SOAP) [146, 147, 148].

SOAP is described as being a lightweight XML protocol for the exchange of information in distributed environments. It consists of two frameworks:

1. The *envelope* framework – which is used for describing what is in a message and how to process it. Envelopes consist of an optional SOAP *header* and a *body*. Headers can contain zero or more blocks of information which can be processed by intermediate receivers or *nodes*. The body must contain information which will be processed by the final recipient of the message.

2. The *transport binding* framework – which uses an underlying protocol for the exchange of messages. Fundamentally SOAP messages are one-way, from a sender to a receiver, although they can be combined to form two-way communication.

The SOAP header is designed to extend the message without affecting the processing of the body. The client can specify which of the intermediate nodes can process specific blocks of information in the header. For example a SOAP message might contain a digital signature, in the header, for information stored in the body. An intermediate node could use this signature to validate the information, which would then be passed to the recipient for processing.

The *ebXML* [139] initiative is currently extending SOAP, to provide better interoperability in the electronic business (e-business) environment, see Ibbotson [82]. These extensions can be used by companies, to define electronic collaboration profiles and agreements, which improve the way in which they integrate their businesses.

Overall SOAP is a message passing framework. It provides a mechanism for a sender and a receiver to communicate. This communication however is only one-way, unless several SOAP messages are combined. Also the real-time delivery of data is beyond its scope. The new Open Hypermedia tool for temporal linking, on the other hand, requires two-way communication from the beginning. The client and the server must communicate with each other, to set-up, deliver and control the streaming media (audio or links).

7.4 Summary

In this chapter a case study of the Open Hypermedia tool for temporal linking with audio streams has been described. The initialisation of the new RTSP framework, both the client and the server, is discussed and a demonstration of the three new methods for the display, traversal and creation of temporal links is given. Each of these methods are described in more detail in the previous chapter, see **Sections 6.4.1**, **6.4.2** and **6.4.3** respectively.

This chapter then reiterates the original objectives, see **Section 1.2**, for this research and evaluates how each objective was achieved. The first objective was reached with the development of the streaming SoundViewer, see **Section 5.3**. The second objective was achieved by creating a new implementation of RTSP and then extending this new framework to support FOHM, see **Chapter 6**. The final objective was reached as a result of combining RTSP with FOHM; FOHM provides the Open Hypermedia functionality for RTSP and RTSP provides the communiction protocol for FOHM.

The next chapter brings the thesis to a close by drawing together the various threads of research. It also describes how FOHM and the new RTSP implementation can be improved and it then discusses several possible areas for future research.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis 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 or a link. This is not synonymous to the way sound is used in everyday situations. Humans generally use a mixture of the five senses, such as sight, hearing, touch, taste and smell, 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 research consisted of three main objectives:

1. The extension of an existing Open Hypermedia tool, the SoundViewer tool for Microcosm, that would allow users to create links to *and from* streaming audio.

2. The development of a new Open Hypermedia tool, that extends a streaming protocol to support an interoperable linkbase format, so that users can display, follow and create temporal links to audio streams.

3. The extension of the Open Hypermedia mechanism, used in the previous objective, to support a communication protocol.

There are several existing standards, MHEG and HyTime, that could have been used

to do this. MHEG however, was primarily designed to help create visual easy-to-use distributed systems and audio is just a small part of this. The audio domain has been supported by earlier versions of HyTime (SMDL), however it is a complex standard with few implementations. Both of these standards have focused specifically on the visual domain.

The audio domain is also being used by two other international standards; MPEG and DAB. MPEG consists of five phases with the first two (MPEG-1 and MPEG-2) concentrating on algorithms and tools for the efficient encoding, storage and delivery of high-quality digital audio and video media. MPEG-1 and MPEG-2 however, can not handle multimedia content by themselves. MPEG-4 was developed to overcome this problem and it is the most recent phase to become a standard. During its development the group realised that the audio domain was just as important as video. Therefore they developed more advanced audio tools, for 3-D spatialization effects, audio "scene" creation and the handling of musical scores. MPEG-4 could be used for the development of these Open Hypermedia tools, mentioned above, however it has only recently become a standard and very few implementations actually exist. MPEG-7 and MPEG-21 are still under development and they will be used to describe multimedia content and to develop multimedia frameworks, respectively.

Digital Audio Broadcasting (DAB) is a digital radio system that transmits digital audio and data services to mobile, portable and fixed receivers. These services are multiplexed together, with the audio, to form ensembles. Some of the data services are actually embedded into the audio itself, to assist in synchronisation. These embedded services provide extra information to do with the programme, including lyrics for songs. DAB is a relatively new medium for the delivery of digital audio and it could be used for a portable or mobile version of these Open Hypermedia tools. However digital radio transmits an infinite stream of audio and this is a potential area for future research, see **Section 8.2.4**.

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 (OHSs) 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 was modified, so that it could communicate with a streaming audio server.

To create this new streaming SoundViewer client, the functionality of the tool was 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.

This new streaming SoundViewer tool could be used to create links to and from audio streams, using the Microcosm OHS. However in 1994 the open hypermedia research community realised that the current generation of these OHSs could not interoperate. The majority of these systems use proprietary protocols for communication between their

clients and link servers. In some cases the link servers also used proprietary formats to actually store the links. As a result of this a client developed for one system could not be used, with the servers, on another.

To overcome this problem the Open Hypermedia Systems Working Group (OHSWG) decided to develop the Open Hypermedia Protocol (OHP). This protocol would be used as the communication mechanism between the clients and the servers. An OHP-aware client therefore, would be able to communicate with an OHP-aware server. However the size of the protocol slowly grew and the group realised that a single protocol would not be able to handle all of the functionality. As a result, OHP was divided into specific hypertext domains, including the traditional form of navigational hypertext. OHP-Navigational (OHP-Nav) was demonstrated at two conferences.

At this stage the focus of these protocols was on the communication between the clients and the servers. The Southampton members of the OHSWG realised however that a higher-level structure could be created, to work across the three most common hypertext domains; navigational, spatial and taxonomic hypertext. Therefore they created the Fundamental Open Hypermedia Model (FOHM), which is an abstract model for describing interoperable associational structures. FOHM is defined using a XML DTD, which uses specific elements to describe these structures. The model itself however, provides no communication mechanism.

For the second area of research, it was decided that the RTSP protocol would be extended, to support Open Hypermedia. This new tool would allow users to display, follow and create temporal links with audio streams. As mentioned previously however, the traditional OHSs can not interoperate and so it was decided that FOHM would be used to handle the link information. In fact, the combination of RTSP and FOHM is a mutual relationship. RTSP provides a communication protocol for FOHM and FOHM provides an interoperable open hypermedia format for RTSP.

The original FOHM DTD can be used to describe a single association. With a small modification this DTD can be used to describe multiple associations in a single XML document; in effect creating an interoperable linkbase format. For the new tool, the traditional form of navigational hypertext was used, since it provides the functionality to create links to and from audio streams.

A new version of RTSP was also created because the original implementation, used

with the SoundViewer tool, was based on a draft standard and it only had enough functionality to handle simple streams. This new implementation, of the RTSP framework, was then extended to support several new methods and user commands. These could then be used to display, follow and create temporal links.

This new framework was also extended to handle the next generation of Internet Protocol (IPv6). The current Internet Protocol (IPv4) will eventually be phased out, in favour of IPv6. This new protocol also supports the original protocol, so that during the transition phase between IPv4 and IPv6, clients will be able communicate using either protocol. By default the RTSP framework uses IPv6.

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. The research, carried out in this thesis, has shown that it is possible to extend an existing open hypermedia audio tool to support streams. It has also shown that an existing streaming protocol can be extended to support open hypermedia, by using an interoperable exchange format and several new methods for link creation, display and traversal.

8.2 Future Work

The Open Hypermedia tool for Temporal Linking with Audio Streams was developed to show how an existing streaming protocol could be extended, to support open hypermedia. This new tool can be used to display, follow and create links to and from audio streams. This section describes several improvements that can be made to this tool and a number of new areas for future research.

8.2.1 Improvements to the RTSP Implementation

Improvements can be made to the RTSP Framework and they are a threading Socket++ library, improved support for audio formats and the implementation of the "RECORD" method and its state, for the RTSP protocol.

The new Open Hypermedia tool uses the Real Time Streaming Protocol (RTSP), see **Section 4.4**, for the delivery of the audio streams. This protocol runs "on-top-of" the underlying network protocols, which are used over the Internet. The current Internet

Protocol (IPv4), see **Section 4.2**, will be slowly phased out, in favour of the next generation of protocol (IPv6), see **Section 4.5**. IPv6 extends and enhances the current protocol and it is also backwards compatible with IPv4.

To ensure that the RTSP framework could be used with either protocol, a new library (Socket++) was developed, see **Section 6.2.1**. This library was written in C++ and it can be used to create TCP or UDP sockets, for connection-oriented or connectionless communication respectively. These sockets can use either IPv4 or IPv6 addresses.

This library however does not use *threads*. A thread is similar to a *process* which are used to execute pieces of code together, at the same time. On a single processor machine, a technique known as *time-slicing* is used for this. This is where the kernel, of the operating system, allocates a small amount of time to each process. For the new framework, a process is used to execute a Web browser, whenever a link is followed to a non-RTSP URL, see **Section 6.4.2**. However processes take up more system resources than threads and they can also increase the complexity of the code. Threads are usually called lightweight processes.

To create a multiuser server, processes are usually created for each user. This occurs at the socket level, whenever the server accepts a connection from a client. As mentioned previously however, processes can consume a lot of the system's resources. Therefore threads can be used instead, to reduce this system overload. In effect the server will be a single process, containing multiple threads; one for each user. This would increase the efficiency of the new RTSP server.

Currently the RTSP framework only supports the WAV audio format, for streaming audio. Another improvement would be the support of more audio formats, such as MP3s (see **Section 3.4.1**), MIDI, AIFF, raw audio and possibly speech formats. By creating new classes to handle each of these formats and by using the `StreamIntf` and `Stream` classes, see **Section 6.2.3**, it would be possible to integrate these audio formats into the framework.

Another improvement would be the implementation of the "`RECORD`" method and hence its appropriate state, see RFC 2326 [53]. This would allow clients to record audio to a URI, which could be the server to which it is connected or another. As mentioned in **Section 4.4** RTSP needs to maintain "session state", so that the server can correlate RTSP requests, from the clients, to the correct session. A session can be in one of four states, see **Figure 4.1** and these are, the `INIT`, `SETUP`, `PLAY` and `RECORD` states. For the

new RTSP framework, only the first three have been implemented. The `RECORD` state was not needed for proof of concept.

The `WavFile` and `WavStream` classes however, see **Section 6.2.3**, already have the functionality to write the stream and hence the file to disk. Therefore to enable users to record audio streams to WAV files, the framework would only have to be extended to support the "`RECORD`" method and its equivalent state.

However, there are issues with copyright. Most streaming tools that are available for the Web, such as RealNetworks RealPlayer and Apple's Quicktime Viewer, do not provide the functionality to record audio. If they provided a "Record" button for their tools, then most users would be able to record music of the Internet. These companies would then be liable for breaching the copyright agreements.

8.2.2 Potential research for the RTSP Framework

There are a number of potential areas of research for the framework and they include the Socket++ library again and the development of a Graphical User Interface (GUI).

The Socket++ library, see **Section 6.2.1**, is designed to work with both IPv4 and IPv6. This library however, can be used as a testbed for further research, especially for the IPv6 protocol. Two possible areas of interest are:

- IP Multicast – which is a particular type of addressing, see **Section 4.2**, that allows datagram packets to be sent to a specific set of hosts. This set is known as a multicast group and users can join or leave it at any time. IP multicast has been used at conferences, to stream live sessions over the Internet. To view these sessions clients simply join the group, which has been setup by the conference organisers. IP multicast can also be used to stream live radio broadcasts, over the Internet.

  A possible area of research here, is how a client can create links to and from the audio streams. The server, that is hosting the group, might only allow the clients to view and then follow the available links. Obviously a client will not be able to interrupt the flow of the multicast stream, if a link for instance points back to a previous section of the presentation.

- The IPv6 Flow Label – which is used to provide a non-default Quality-of-Service (QoS) or a real-time service, see **Section 4.5**. This flow label could be used to handle streaming media, such as audio and video streams. It ensures that the packets

"flow", from the source to the client, with a minimum amount of processing overhead, at the IPv6 routers. At the moment however, this part of the protocol is still under development and once this is complete, it will require further research.

Another area of research is the investigation of a GUI, that could help the user in displaying, following and creating links. The RTSP framework currently uses a command line interface which can be difficult to use, especially when displaying multiple links and creating new links; see the `CREATE_LINK` user command in **Section 6.4.3**.

The majority of the streaming audio tools, that are used with the WWW, use a scrollbar to represent the position within the audio stream. The length of the scrollbar is also used to represent the duration of the stream. This is a common graphical metaphor for visualising audio, see **Section 5.2**. However this metaphor can not be used, by itself, to visually represent both the audio and the links. It can be used, in a limited way, to create links but these again will not be seen. Therefore in this situation a new window would have to be used, to display the links. By double-clicking on these links, users will be to follow them.

The SoundViewer Tool for Microcosm however, uses a different visualisation technique. Its interface, see **Section 5.2.1**, consists of two windows; an overview window, which represents the duration of the audio file or stream and a detail window. The overview window is similar to a scrollbar and it contains a highlighted rectangle, which represents a zoomed-in view in the detail window. This rectangle can be resized and moved. During playback this rectangle will also move to represent the current position within the audio. All of the links are visually displayed, as rectangles, in the detail window and users can click on these links to follow them. Links can be created to and from the entire file or stream. The detail window can also be used to highlight a portion of the audio, which can then be used to create more links. The highlighted portion will then became a link, visually represented as a smaller rectangle of the same length.

Although this interface is slightly dated, it does represent a good metaphor for visually handling both the links and the audio. Possibly a similar representation could be used with the new Open Hypermedia tool for temporal linking.

8.2.3 FOHM Improvements

The FOHM linkbase format is defined using a XML DTD, see **Section 6.3**. This defines the structure of FOHM associations by using specific elements. As a result, this

format does not know anything about the associations it describes; it only knows about the structure of the XML document. Therefore when describing an association, all of the information has to be provided; for example, the Abstract Data Type (ADT) that this association uses, the feature space and feature vectors it uses, see **Section 2.5** and the binding values.

For the RTSP framework this DTD was modified, so that the FOHM XML documents can contain multiple instances of associations; in effect creating interoperable FOHM linkbases. Associational structures in FOHM are designed to work across three main hypertext domains and for the FOHM linkbases only one of these was used; the traditional form of navigational hypertext. Therefore a FOHM linkbase, in this framework, will contain multiple instances of navigational associations.

However all of the information, for each association, has to be provided and as a result, there is a lot of redundant data; for example the values for the `relationtype` and `structuretype` elements will be the same for each navigational association. This also increases the size of the FOHM linkbases, see **Appendix A.3** for an example of a single association.

It is possible to overcome this problem by extending FOHM to support either:

1. A default associational structure. The majority of "Open" Hypermedia Systems (OHS) and the WWW use the traditional form of navigational hypertext. Therefore by default each association could support this type of hypertext, unless the client redefines it.

2. New elements in the XML DTD. These elements would be used to define the default associational structure for the entire linkbase. It would be defined only once, at the beginning of the FOHM linkbase and it would apply to each of the associations. There would be no need to use the `relationtype` and the `structuretype` elements.

Both of these methods could actually be used to overcome this problem, although defining navigational hypertext as the default structure to use, seems the obvious solution.

If in the future however, the new RTSP framework supports the other two hypertext domains (spatial and taxonomic), linkbases might contain a mixture of all three associational structures. As a result the information for each structure would have to be provided

again. A possible solution would be to store each type of associational structure in its own linkbase; e.g. a linkbase for spatial links and another for navigational.

8.2.4 Possible research areas for FOHM

A possible area of research is to reduce the size of the actual linkbases, so that they could be streamed down at the same time as the audio. Currently the framework uses the `AVAILABLE_LINKS` method, see **Section 6.4.1**, to stream down the link information before streaming down the audio. This ensures that the links are available for display, when the audio is played back.

The link information is also streamed down before the audio because of the size of the linkbases and the need to process this information, to obtain the relevant data; e.g. the source and destination URIs from the `content` elements, see **Section 6.3**. By reducing the size of the linkbases and by sending the link information, just before the relevant points in the audio, it might be possible to stream down the linkbases at the same time as the audio. However the link information will still need to be processed and if there are multiple linkbases, there could be long delays before the links would actually be displayed.

Another possible area of research is the use of temporally infinite streams, such as a live TV broadcast and context. Currently the Open Hypermedia tool uses finite audio streams for temporal linking; for example audio files of a specific length stored on a RTSP server. The framework can be extended to support streams of infinite length but this could impose a few problems.

The RTSP framework would be able to play this continuous media, however clients would not be able to display, create or follow links. The framework for instance allows users to create links from say 1000ms into the presentation, for a duration of 2000ms. Users would not be able to do this with temporally infinite media because by its very definition, there is no start point and therefore no fixed duration.

Digital Audio Broadcasting (DAB), see **Section 3.4.2**, is used to transmit, in real-time, an infinite stream of digital audio. In DAB, Programme Associated Data (PAD) is used to provide extra information about the current programme; for example lyrics for songs and phone-in telephone numbers. This information is synchronised with the audio and therefore it is embedded into the audio. A return channel, from a user's receiver to

the nearest digital radio station, is also under development and it might allow users to request extra information on a particular programme or possibly follow a link.

For the framework it might be possible to use a similar technique to DAB, in which the link information is embedded within the audio. However this is synonymous to embedding links into documents and this approach has many drawbacks; specifically the dangling link problem. For embedding PAD information into the continuous audio stream however, DAB transmitters must also use dedicated hardware, especially if it is done in real-time. The RTSP framework is just a software tool to demonstrate Open Hypermedia and temporal linking with audio streams. At this stage dedicated hardware for temporal linking has not even been considered.

Another issue is how FOHM would handle infinite audio streams and this is where the concept of context might be used. FOHM can already handle finite streams by using the `AXISLOC`, see **Section 6.3**, and specific time values. This element however, can not be used with temporally infinite streams; there are no durations or start and end times.

Context has been used in hypermedia to reduce the complexity of navigation. It can be used, in one sense, to assist users in finding specific information, by filtering out irrelevant data. Some systems actually adapt, to the users' requirements, so that only the relevant material will be shown. In another sense context is the view that the source and destination of a link should not be a node but rather a point (or context) within a node; for example linking to the fourth minute in a sound file.

For temporally infinite audio streams, context would have to be used to determine what information the users would want to see; for example a user could request for all of the information to do with a particular artist on a radio station. However the amount of information required to handle all cases for all users could also be infinite, which complicates this area even more. This area requires further research.

Appendix A

Program and DTD Listings

A.1 SMIL Program Listing

```
<smil>
 <head>
  <meta name="title"
        content="Winston Churchill - Sinew's of Peace
                 Demo"/>
  <meta name="author" content="Neil Ridgway"/>
  <meta name="copyright"
        content="Copyright (c) University of
                 Southampton"/>
  <meta name="base"
        content="rtsp://ernie.ecs.soton.ac.uk:554/data/-
                 incoming/cnhr/Churchill/"/>
  <layout>
   <root-layout id="Main-Window" background-color="white"
                title="Winston Churchill - Sinew's of
                       Peace"
                width="600" height="400"/>
   <region id="Transcript" left="2%" top="2%" width="52%"
           height="40%" z-index="1"/>
   <region id="LeftArrow" left="5%" top="50%" width="62"
           height="63"/>
```

```
   <region id="RightArrow" left="20%" top="50%" width="62"
           height="63"/>
   <region id="Image1" left="38%" top="22%" width="360"
           height="304" z-index="2"/>
   <region id="Image2" left="45%" top="25%" width="288"
           height="266" z-index="2"/>
   <region id="Image3" left="65%" top="25%" width="144"
           height="204"/>
  </layout>
</head>
<body>
 <seq>
  <par id="part1">
   <audio src="sinew01.rm" dur="34.9s"/>
   <textstream id="text1" region="Transcript"
               src="sinew01.rt" dur="34.9s"/>
   <ref region="Image1" src="church2r.rp" begin="2s"
        end="34.9s"/>
   <ref region="RightArrow" src="purpwood_right.rp"
        dur="34.9s">
    <anchor href="#part2" dur="34.9s"/>
   </ref>
  </par>

  <par id="part2">
   <audio src="sinew02.rm" dur="15.6s"/>
   <textstream id="text2" region="Transcript"
               src="sinew02.rt" dur="15.6s"/>
   <ref region="Image2" src="church3r.rp" begin="2s"
        end="15.6s"/>
   <ref region="LeftArrow" src="purpwood_left.rp"
        dur="15.6s">
    <anchor href="#part1" dur="15.6s"/>
   </ref>
   <ref region="RightArrow" src="purpwood_right.rp"
        dur="15.6s">
```

```
      <anchor href="#part3" dur="15.6s"/>
     </ref>
    </par>

    <par id="part3">
     <audio src="sinew03.rm" dur="37.9s"/>
     <textstream id="text3" region="Transcript"
                 src="sinew03.rt" dur="37.9s"/>
     <ref region="Image3" src="church1r.rp" begin="2s"
          end="37.9s"/>
     <ref region="LeftArrow" src="purpwood_left.rp"
          dur="37.9s">
      <anchor href="#part2" dur="37.9s"/>
     </ref>
    </par>
   </seq>
  </body>
</smil>
```

A.2 The FOHM Linkbase DTD

```
<?xml encoding="ISO8859-1"?>

<!-- The linkbase root element / document type -->
<!ELEMENT linkbase (association)+>

<!-- Association -->
<!ELEMENT association    (id, bindingvector, relationtype,
                          structuretype)>
<!ELEMENT id             (#PCDATA)>
<!ELEMENT relationtype   (name, featurespace)>
<!ELEMENT name           (#PCDATA)>
<!ELEMENT featurespace   (feature)*>
<!ELEMENT feature        (#PCDATA)>
<!ELEMENT structuretype  (#PCDATA)>
```

```
<!-- Binding -->
<!ELEMENT bindingvector (binding)*>
<!ELEMENT binding       ((refid|dataref), featurevector)>
<!ELEMENT refid         (#PCDATA)>
<!ELEMENT featurevector (featurevalue)*>
<!ELEMENT featurevalue  (#PCDATA)>

<!-- DataRef -->
<!ELEMENT dataref (refid?, (dataid|data), (NALOC|AXISLOC)?)>
<!ELEMENT dataid  (#PCDATA)>

<!-- Data -->
<!ELEMENT data        (id?, contentspec)>
<!ELEMENT contentspec (version, content)>
<!ELEMENT version     (#PCDATA)>
<!ELEMENT content     (#PCDATA)>

<!-- LocSpecs -->
<!ENTITY % locSpecInfo     "SELECTION?, SELECTIONCONTEXT?,
                           SELECTIONTYPE?">
<!ELEMENT SELECTION        (#PCDATA)>
<!ELEMENT SELECTIONCONTEXT (#PCDATA)>
<!ELEMENT SELECTIONTYPE    (#PCDATA)>

<!-- NALOC -->
<!ELEMENT NALOC ((%locSpecInfo;)?, spec)>
<!ELEMENT spec  (#PCDATA)>

<!-- AXISLOC -->
<!ELEMENT AXISLOC    ((%locSpecInfo;)?, fwdaxisset?, revaxisset?)>
<!ELEMENT fwdaxisset (axis)*>
<!ELEMENT revaxisset (axis)*>
<!ELEMENT axis       (name?, type, valueset)>
<!ELEMENT type       (#PCDATA)>
<!ELEMENT valueset   (value)*>
<!ELEMENT value      (#PCDATA)>
```

A.3 A Navigational Association in FOHM

```
<association>
 <id>link3</id>
  <bindingvector>
   <binding>
    <dataref>
     <data>
      <contentspec>
       <version>1</version>
       <content>
       rtsp://ratbert:3048/sinew01/sinew22-01.wav
       </content>
      </contentspec>
     </data>
     <AXISLOC>
      <fwdaxisset>
       <axis>
        <name>time</name>
        <type>ms</type>
        <valueset>
         <value>1000,2000</value>
        </valueset>
       </axis>
      </fwdaxisset>
    </AXISLOC>
   </dataref>
   <featurevector>
    <featurevalue>source</featurevalue>
   </featurevector>
  </binding>
  <binding>
   <dataref>
    <data>
     <contentspec>
      <version>1</version>
```

```
        <content>rtsp://ratbert:3048/rtsp.wav</content>
       </contentspec>
      </data>
      <AXISLOC>
       <fwdaxisset>
        <axis>
         <name>time</name>
         <type>ms</type>
         <valueset>
          <value>500,2500</value>
         </valueset>
        </axis>
       </fwdaxisset>
      </AXISLOC>
     </dataref>
     <featurevector>
      <featurevalue>destination</featurevalue>
     </featurevector>
    </binding>
   </bindingvector>
   <relationtype>
    <name>link</name>
    <featurespace>
     <feature>direction</feature>
    </featurespace>
   </relationtype>
  <structuretype>set</structuretype>
</association>
```

Bibliography

[1] A. A. Rodriguez, M. Fisher and B. Markey. Scripting Languages Emerge in Standards Bodies. *IEEE Multimedia*, pages 88–92, Winter 1995.

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

[3] A. Eliens, M. V. Welie, J. V. Ossenbruggen and B. Schonhage. Jamming (on) the Web. In *The 6th International WWW Conference Proceedings*, Santa Clara, California, USA, April 1997. International World Wide Web Conference Committee (IW3C2).

[4] A. M. Fountain, W. Hall, I. Heath and H. C. Davis. MICROCOSM: An Open Model for Hypermedia with Dynamic Linking. In *ACM European Conference on Hypertext*, pages 298–311, Paris, France, November 1990. Association for Computing Machinery (ACM), Cambridge University Press.

[5] 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.

[6] An ArborText Inc. SGML White Paper. SGML: Getting Started. Technical report, ArborText, Inc., http://www.arbortext.com/wp.html, ©1992, 1995.

[7] ANSI X3V1.8M Committee. *ISO/IEC CD 10743, Information Technology – Standard Music Description Language*. International Organization for Standardization (ISO), Geneva, Switzerland, 1991.

[8] Audio-Video Transport Working Group. *RFC 1889, RTP: A Transport Protocol for Real-Time Applications*. Internet Engineering Task Force (IETF),

http://www.ietf.org/rfc/rfc1889.txt?number=1889, January 1996.

[9] B. C. J. Moore. *An Introduction to the Psychology of Hearing*. Academic Press, 4th edition, 1997.

[10] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, 3rd edition, October 1997.

[11] V. Bush. As we may think. In *Atlantic Monthly*, pages 101–108, July 1945.

[12] L. Chiariglione. *Short MPEG-1 description*. ISO, http://drogo.cselt.stet.it/mpeg/-standards/mpeg-1/mpeg-1.htm, June 1996.

[13] L. Chiariglione. *Short MPEG-2 description*. ISO, http://drogo.cselt.stet.it/mpeg/-standards/mpeg-2/mpeg-2.htm, July 1996.

[14] L. Chiariglione. Impact of MPEG Standards on Multimedia Industry. *Proceedings of the IEEE*, 86(6):1222–1227, June 1998.

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

[16] Copyright © 1997-2000 Object Management Group (OMG) Inc. *CORBA Basics*. Object Management Group (OMG), http://www.omg.org/gettingstarted/corb-afaq.htm, December 2000.

[17] Copyright © Digital Television Group (DTG). *Systems and MHEG-5: Extensions for EuroMHEG 1*. Digital Television Group (DTG), http://www.dtg.org.uk/-mheg/euromheg1.pdf, March 1999.

[18] Copyright © International Business Machines Corp. *IBM's Alphaworks Home Page*. International Business Machines Corp., http://www.alphaworks.ibm.com, 2000.

[19] Copyright © Object Management Group, Inc. *The Common Object Request Broker: Architecture and Specification (Revision 2.4)*. Object Management Group, http://www.omg.org/technology/documents/formal/corbaiiop.htm, October 2000.

[20] Copyright © Sun Microsystems Inc. *IPv6 and the Future of the Internet*. Sun Microsystems Inc., http://www.sun.com/software/white-papers/wp-ipv6/index.html, 1999. A Technical White Paper.

[21] D. C. A. Bulterman, L. Hardman, J. Jansen, K. S. Mullender and L. Rutledge. GRiNS: A GRaphical INterface for creating and playing SMIL documents. In *Proceedings of the 7th International WWW Conference*, volume 30, pages 519–529, Brisbane, Australia, April 1998. International World Wide Web Conference Committee (IW3C2).

[22] D. C. Engelbart and W. K. English. A Research Center for Augmenting Human Intellect. *AFIPS Conference Proceedings*, 33(1), 1968.

[23] D. DeRoure, S. Blackburn, L. Oades, J. Read, N. Ridgway. Applying Open Hypermedia to Audio. In *Proceedings of the 9th ACM Conference on Hypertext and Hypermedia*, pages 285–286, Pittsburgh PA, USA, June 1998. Association for Computing Machinery (ACM).

[24] D. E. Millard and H. C. Davis. Navigating Space: The Semantics of Cross Domain Interoperability. In *Proceedings of the 2nd International Workshop on Structural Computing, Hypertext 2000*, pages 129–139, San Antonio, Texas, USA, May-June 2000. Association for Computing Machinery (ACM).

[25] D. E. Millard and S. Reich. OHP – Communicating between Hypermedia Aware Applications. In *A Workshop on Global Hypermedia Infrastructure*, Brisbane, Australia, April 1998. Association for Computing Machinery (ACM), International World Wide Web Conference Committee (IW3C2). Held in conjunction with the 7th International WWW Conference.

[26] D. E. Millard, H. C. Davis and L. Moreau. Standardizing Hypertext: Where next for OHP? In *Proceedings of the 6th International Workshop on Open Hypermedia Systems, Hypertext 2000*, pages 3–12, San Antonio, Texas, USA, May-June 2000. Association for Computing Machinery (ACM).

[27] D. E. Millard, L. Moreau, H. C. Davis and S. Reich. FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability between Hypertext Domains. In *Proceedings of the 6th Workshop on Open Hypermedia Systems, Hypertext 2000*, pages 93–102, San Antonio, Texas, USA, May-June 2000. Association

for Computing Machinery (ACM).

[28] D. E. Millard, S. Reich and H. C. Davis. Reworking OHP: the Road to OHP-Nav. In *Proceedings of the 4th Workshop on Open Hypermedia Systems*, pages 48–53, Pittsburgh, Pennsylvania, USA, June 1998. Association for Computing Machinery (ACM).

[29] D. Thorn, H. Purnhagen and the MPEG Audio Subgroup. *MPEG Audio FAQ*. International Organization for Standardization (ISO), http://www.tnt.uni-hannover.de/project/mpeg/audio/faq/, December 1999.

[30] Digital Audio Visual Council (DAVIC). *Davic 1.4.1 Specifications, Parts 1-14*. Digital Audio Visual Council (DAVIC), ftp://ftp.davic.org/davic/pub/Spec1_4_1/, December 1999.

[31] DOM Working Group. *Document Object Model (DOM) Level 1 Specification, Version 1.0*. World Wide Web Consortium (W3C), http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, October 1998.

[32] DOM Working Group. *Document Object Model (DOM) Level 2 Specification*. World Wide Web Consortium (W3C), http://www.w3.org/TR/WD-DOM-Level-2, October 1999.

[33] E. D. Scheirer, R. Väänänen and J. Huopaniemi. AudioBIFS: Describing Audio Scenes with the MPEG-4 Multimedia Standard. *IEEE Transactions on Multimedia*, 1(3):237–250, September 1999.

[34] E. D. Scheirer, Y. Lee and JW. Yang. Synthetic and SNHC Audio in MPEG-4. Technical report, Massachusetts Institute of Technology, Machine Listening Group, MIT Media Laboratory, 1999.

[35] EBU/CENELEC/ETSI JTC Broadcast. *ETS 300 401, Radio Broadcasting Systems: Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers*. European Telecommunications Standards Institute (ETSI), http://www.etsi.org/-broadcast/dab.htm, 2nd edition, May 1997.

[36] EBU/CENELEC/ETSI JTC Broadcast. *EN 50067, Specification of the radio data system (RDS) for VHF/FM sound broadcasting in the frequency range from*

87.5 to 108.0 MHz. European Telecommunications Standards Institute (ETSI), http://www.rds.org.uk, October 1998.

[37] EBU/CENELEC/ETSI JTC Broadcast. *EN 50255, Digital Audio Broadcasting System – Specification of the Receiver Data Interface (RDI)*. European Telecommunications Standards Institute (ETSI), http://www.etsi.org/broadcast/dab.htm, June 1998.

[38] EBU/CENELEC/ETSI JTC Broadcast. *EN 301 234 (Version 1.2.1), Digital Audio Broadcasting (DAB); Multimedia Object Transfer (MOT) protocol*. European Telecommunications Standards Institute (ETSI), http://www.etsi.org/broadcast/dab.htm, February 1999.

[39] EBU/CENELEC/ETSI Technical Committee. *DVB Specifications and Standards*. Digital Video Broadcasting (DVB) Project, http://www.etsi.org/broadcast/dvb.htm, 1995-1999.

[40] © EUREKA-147 Project. *EUREKA-147 – Digital Audio Broadcasting*. World DAB Forum (WorldDAB), http://www.worlddab.org/public_documents/eureka_brochure.pdf, August 1997.

[41] F. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, February 1994.

[42] F. Halsall. *Data Communications, Computer Networks and Open Systems*. Addison Wesley, 3rd edition, 1992.

[43] F. Kappe. Hyper-G: A Distributed Hypermedia System. In Barry Leiner, editor, *Proceedings of INET*, San Francisco, California, August 1993. Internet Society.

[44] F. Kappe and H. Maurer. Hyper-G: A large Universal Hypermedia System and some spin-offs. In *Computer Graphics - online*, ftp://ftp.siggraph.org/publications/misc/May_93_online/Kappe.Maurer/Kappe.Maurer.PS, May 1993.

[45] F. Nack and A. T. Lindsay. Everything you wanted to know about MPEG-7: Part 1. *IEEE Multimedia*, 6(3):65–77, Jul-Sep 1999.

[46] F. Nack and A. T. Lindsay. Everything you wanted to know about MPEG-7: Part

2. *IEEE Multimedia*, 6(4):64–73, Oct-Dec 1999.

[47] G. Geiger. A Digital Audio Player for the HyperWave Internet Server. Master's thesis, Graz University of Technology, January 1997.

[48] G. Hill, L. Carr, D. DeRoure and W. Hall. The Distributed Link Service: Multiple Views on the WWW. In *The 7th ACM Conference on Hypertext*, Washington DC, USA, 16th-20th March 1996. Technical briefing for Hypertext '96.

[49] H. C. Davis, A. Rizk and A. Lewis. OHP: A draft proposal for a standard open hypermedia protocol. In *Proceedings on the 2nd Workshop on Open Hypermedia Systems*, pages 27–53, Washington, D.C., March 1996. Association for Computing Machinery (ACM), Technical Report No. ICS-TR-96-10, University of California. Eds, U. K. Will and S. Demeyer.

[50] H. C. Davis, D. E. Millard, S. Reich, N. Bouvin, K. Grønbæk, P. J. Nürnberg, L. Sloth, U. K. Wiil and K. Anderson. Interoperability between Hypermedia Systems: The Standardisation Work of the OHSWG. In *Proceedings of the 10th ACM Conference on Hypertext and Hypermedia*, pages 201–202, Darmstadt, Germany, February 1999. Association for Computing Machinery (ACM).

[51] H. C. Davis, S. Reich and D. E. Millard. A proposal for a common Navigational Hypertext Protocol. Technical report, Presented at 3.5 Open Hypermedia System Working Group Meeting, Aarhus University, Denmark, September 1997.

[52] 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*, pages 181–190, Milan, Italy, December 1992. Association for Computing Machinery (ACM).

[53] H. Schulzrinne, A. Rao and R. Lanphier. *RFC 2326, Real Time Streaming Protocol (RTSP)*. Internet Engineering Task Force (IETF), http://www.ietf.org/rfc/-rfc2326.txt?number=2326, April 1998.

[54] H. W. Bitzer and K. Hofrichter. *MHEG-5 – The Standard API for Digital Television*. The MHEG Centre, http://www.mhegcenter.com/mheg/mheg5-ibc.pdf, September 1998.

[55] H. Yasuda and H. J. F. Ryan. DAVIC and Interactive Multimedia Services. *IEEE Communications Magazine*, 36(9):137–143, September 1998.

[56] F. G. Halasz. Reflections on NoteCards: Seven issues for the next generation of Hypermedia Systems. *Communications of the ACM*, 31(7):836–852, July 1988.

[57] P. Hoschka. Towards Synchronized Multimedia on the Web. *World Wide Web Journal*, 2(2), Spring 1997.

[58] P. Hoschka. An Introduction to the Synchronized Multimedia Integration Language. *IEEE Multimedia*, 5(4):84–88, Oct-Dec 1998.

[59] P. Hoschka. *"Synchronized Multimedia" Working Group Charter.* World Wide Web Consortium (W3C), http://www.w3.org/AudioVideo/Group/symm-wg-charter, January 2000.

[60] HTML Working Group. *HTML 4.01 Specification.* World Wide Web Consortium (W3C), http://www.w3.org/TR/1999/REC-html401-19991224, December 1999.

[61] ISO JTC1/SC34. *ISO 8879, Information Processing – Text and Office Systems – Standard Generic Markup Language.* International Organization for Standardization (ISO), Geneva, Switzerland, August 1986.

[62] ISO/IEC JTC1/SC29. *ISO/IEC 11172 (Parts 1-5), Information Technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbps.* International Organization for Standardization (ISO), Geneva, Switzerland, 1993-1999.

[63] ISO/IEC JTC1/SC29. *ISO/IEC 13522-4, Information technology – Coding of multimedia and hypermedia information - Part 4: MHEG registration procedure.* International Organization for Standardization (ISO), Geneva, Switzerland, November 1996.

[64] ISO/IEC JTC1/SC29. *ISO/IEC 13818 (Parts 1-9), Information Technology – Generic coding of moving pictures and associated audio.* International Organization for Standardization (ISO), Geneva, Switzerland, 1996-2000.

[65] ISO/IEC JTC1/SC29. *ISO/IEC 13522-1, Information technology – Coding of multimedia and hypermedia information - Part 1: MHEG object representation - Base notation (ASN.1)*. International Organization for Standardization (ISO), Geneva, Switzerland, May 1997.

[66] ISO/IEC JTC1/SC29. *ISO/IEC 13522-3, Information technology – Coding of multimedia and hypermedia information - Part 3: MHEG script interchange representation*. International Organization for Standardization (ISO), Geneva, Switzerland, May 1997.

[67] ISO/IEC JTC1/SC29. *ISO/IEC 13522-5, Information Technology – Coding of multimedia and hypermedia information - Part 5: Support for Base-Level Interactive Applications*. International Organization for Standardization (ISO), Geneva, Switzerland, April 1997.

[68] ISO/IEC JTC1/SC29. *ISO/IEC 13522-6, Information Technology – Coding of multimedia and hypermedia information - Part 6: Support for Enhanced Interactive Applications*. International Organization for Standardization (ISO), Geneva, Switzerland, October 1998.

[69] ISO/IEC JTC1/SC29. *ISO/IEC 16500 (Parts 1-9), Information technology – Generic digital audio-visual systems - DAVIC Specifications*. International Organization for Standardization (ISO), Geneva, Switzerland, December 1999.

[70] ISO/IEC JTC1/SC34. *ISO/IEC 10744, Information technology – Hypermedia-/Time-based Structuring Language (HyTime)*. International Organization for Standardization (ISO), Geneva, Switzerland, May 1997.

[71] ISO/IEC JTC1/SC34. *ISO/IEC 14772-1, Information technology – Computer graphics and image processing - The Virtual Reality Modeling Language - Part 1: Functional specification and UTF-8 encoding*. International Organization for Standardization (ISO), Geneva, Switzerland, June 1998.

[72] ISO/IEC JTC1/SC6. *ISO/IEC 8824 (Parts 1-4), Information technology – Abstract Syntax Notation One (ASN.1) Specifications*. International Organization for Standardization (ISO), Geneva, Switzerland, December 1998.

[73] ISO/IEC JTC1/SC6. *ISO/IEC 8825 (Parts 1 & 2), Information technology – ASN.1*

Encoding Rules. International Organization for Standardization (ISO, Geneva, Switzerland, December 1998.

[74] ITU-R Recommendation. *BO 1130-2: System description and selection for digital satellite broadcasting to vehicular, portable and fixed receivers in the bands allocated to BSS (sound) in the frequency range 1400–2700 MHz*. International Telecommunication Union (ITU), Geneva, Switzerland, December 1994.

[75] ITU-R Recommendation. *BS 1114-1: Systems for terrestrial digital sound broadcasting to vehicular, portable and fixed receivers in the frequency range 30–3000 MHz*. International Telecommunication Union (ITU), Geneva, Switzerland, December 1994.

[76] J. Dale. *A Mobile Agent Architecture for Distributed Information Management*. PhD thesis, University of Southampton, Faculty of Engineering and Applied Science, Department of Electronics and Computer Science, September 1997.

[77] J. P. Evain. The Multimedia Home Platform - An Overview. *EBU Technical Review*, Spring 1998. http://www.dvb.org/dvb_articles/dvb_mhp98.pdf.

[78] J. Postel. *RFC 791, Internet Protocol: DARPA Internet Program Protocol Specification*. Information Sciences Institute, http://www.ietf.org/rfc/rfc0791.txt?number=0791, September 1981.

[79] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall International Inc., Simon & Schuster, Englewood Cliffs, New Jersey 07632, 1991. ISBN 0-13-630054-5.

[80] J. V. Ossenbruggen, A. Eliëns and L. Rutledge. The Role of XML in Open Hypermedia Systems. In *Proceedings of the 4th Workshop on Open Hypermedia Systems*, Pittsburgh, USA, June 1998. Association for Computing Machinery (ACM).

[81] JF. Huard, A. A. Lazar, KS. Lim and G. S. Tselikis. Realizing the MPEG-4 Multimedia Delivery Framework. *IEEE Network*, pages 35–45, November/December 1998.

[82] John Ibbotson. *ebXML Trading-Partners Specification*. Internationales Congress

Centrum (ICC) / Graphic Communications Association (GPA), http://www.-gcs.org/papers/xmleurope2001/papers/html/S09-2.html, May 2001.

[83] 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*, pages 13–25, San Antonio, Texas, USA, December 1991.

[84] K. R. Page, D. Cruickshank and D. DeRoure. It's About Time: Link Streams as Continuous Metadata. In H. Davis, Y. Douglas and D. G. Durand, editor, *Proceedings of the 12th ACM Conference on Hypertext and Hypermedia*, pages 93–102, University of Aarhus, Århus, Denmark, August 2001. Association for Computing Machinery (ACM).

[85] R. Koenen. MPEG-4: Multimedia for our time. *IEEE Spectrum*, 36(2):26–33, February 1999.

[86] R. Koenen. *MPEG-4 Overview - (Melbourne Version)*. ISO, http://drogo.-cselt.stet.it/mpeg/standards/mpeg-4/mpeg-4.htm, October 1999.

[87] L. A. Carr, D. W. Barron, H. C. Davis and W. Hall. Why use HyTime? *Electronic Publishing*, 7(3):163–178, September 1994.

[88] L. Carr, D. DeRoure, W. Hall and G. Hill. The Distributed Link Service: A tool for Publishers, Authors and Readers. In *Proceedings of the 4th International World Wide Web Conference*, volume 1, pages 647–656, Boston, USA, December 1995. International World Wide Web Conference Committee (IW3C2), O'Reilly and Associates.

[89] L. Carr, G. Hill, D. DeRoure, W. Hall and H. Davis. Open Information Services. *Computer Networks and ISDN Systems*, 28:1027–1036, November 1996.

[90] L. Chiariglione. *MPEG-4 FAQs*. ISO, http://drogo.cselt.stet.it/mpeg/standards/-mpeg-4/faq.htm, July 1997.

[91] L. Hardman and M. Wilson. *SMIL Hands on Tutorial*. World Wide Web Consortium (W3C), http://www.cwi.nl/mmpapers/SMIL/Tutorial.ps.gz, September 1998.

[92] L. M. Garshol. *Introduction to XML.* http://www.stud.ifi.uio.no/~lmariusg/-download/xml/xml_eng.html, August 1999.

[93] L. Moreau, N. Gibbins, D. DeRoure, S. El-Beltagy, W. Hall, G. Hughes, D. Joyce, S. Kim, D. Michaelides, D. Millard, S. Reich, R. Tansley and M. Weal. SoFAR with DIM Agents. In *Proceedings of the 5th International Conference on the practical application of Intelligent Agents and Multiagent Technology*, pages 369–389, Manchester, UK, April 2000.

[94] L. Rutledge, J. Buford and R. Price. Mobile Objects and the Hyoctane Distributed Hyperdocument Server. *Computers & Graphics*, 20(5):633–639, 1996.

[95] L. Rutledge, J. V. Ossenbruggen, L. Hardman and D. C. A. Bulterman. Anticipating SMIL 2.0: The Developing Cooperative Infrastructure for Multimedia on the Web. In *Proceedings of the 8th International WWW Conference*, pages 343–353, Toronto, Canada, May 1999. International World Wide Web Conference Committee (IW3C2).

[96] L. Rutledge, L. Hardman and J. V. Ossenbruggen. Evaluating SMIL: Three User Case Studies. In *Proceedings of the 7th ACM International Multimedia Conference (Part 2)*, pages 171–174, Orlando, Florida, USA, October 1999. Association for Computing Machinery (ACM).

[97] M. Echiffre, C. Marchisio, P. Marchisio, P. Panicciari and S. D. Rossi. MHEG-5 – Aims, Concepts and Implementation Issues. *IEEE Multimedia*, 1(5):84–91, January/March 1998.

[98] M. Handley and V. Jacobson. *RFC 2327, SDP: Session Description Protocol.* Internet Engineering Task Force (IETF), http://www.ietf.org/rfc/rfc2327.txt?number=2327, April 1998.

[99] N. D. 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.

[100] N. Ridgway and D. DeRoure. FOHM+RTSP: Applying Open Hypermedia and Temporal Linking to Audio Streams. In S. Reich and K. M. Anderson, editor,

Lecture Notes in Computer Science, University of Aarhus, Århus, Denmark, August 2001. Springer Verlag, Heidelberg (ISSN 0302-9743).

[101] 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*, 1(1):81–96, January 1988.

[102] Object Management Group (OMG) Home Page. http://www.omg.org, December 2000.

[103] P. Flynn. *Frequently Asked Questions about the Extensible Markup Language*. World Wide Web Consortium (W3C), http://www.ucc.ie/xml, June 1999.

[104] P. Georghiades and G. Jacobs. Mixed Media. *Personal Computer World*, 20(11):225–242, November 1997.

[105] P. H. Lewis, H. C. Davis, S. R. Griffiths, W. Hall and R. J. Wilkins. Media-based Navigation with Generic Links. In *Hypertext '96*, pages 215–223, Washington DC, USA, 16^{th}-20^{th} March 1996.

[106] P. Schmitz, J. Yu and P. Santangeli. *Timed Interactive Multimedia Extensions for HTML (HTML+TIME)*. World Wide Web Consortium (W3C), http://www.-w3.org/TR/1998/NOTE-HTMLplusTIME-19980918, September 1998. Note for discussion only.

[107] P. Srisuresh and M. Holdrege. *RFC 2663, IP Network Address Translator (NAT): Terminology and Considerations*. Internet Engineering Task Force (IETF), http://www.ietf.org/rfc/rfc2663.txt?number=2663, August 1999.

[108] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. *The Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616)*. Copyright © The Internet Society, http://www.w3.org/Protocols/rfc2616/rfc2616.html, June 1999.

[109] R. Gilligan, S. Thomson, J. Bound and W. Stevens. *RFC 2553, Basic Socket Interface Extensions for IPv6*. Internet Engineering Task Force (IETF), http://-www.ietf.org/rfc/rfc2553.txt?number=2553, March 1999.

[110] R. Joseph and J. Rosengren. MHEG-5: An Overview. *http://www.fokus.gmd.-de/ovma/mug/archives/doc/mheg-reader/rd1206.html*, December 1995.

[111] R. M. Akscyn, D. L. McCracken and E. A. Yoder. KMS: A distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM*, 31(7):820–835, July 1988.

[112] L. Rein. *Is HTML+Time Out-of-Sync With SMIL?* Copyright © Seybold Publications and O'Reilly & Associates, Inc., http://xml.com/xml/pub/98/10/-htmltime.html, October 1998.

[113] Requirements Group. MPEG-7: Context, Objectives and Technical Roadmap, v.12. Technical report, ISO, http://www.darmstadt.gmd.de/mobile/MPEG7/Documents/W2861.htm, July 1999.

[114] Requirements Group. *MPEG-7 Overview (version 2.0)*. International Organization for Standardization (ISO), http://drogo.cselt.stet.it/mpeg/standards/mpeg-7/mpeg-7.htm, March 2000.

[115] S. Battista, F. Casalino and C. Lande. MPEG-4: A Multimedia Standard for the Third Millennium, Part 1. *IEEE Multimedia*, 6(4):74–83, Oct-Dec 1999.

[116] S. Deering and R. Hinden. *RFC 2460, Internet Protocol, Version 6 (IPv6) Specification*. Internet Engineering Task Force (IETF), http://www.ietf.org/rfc/-rfc2460.txt?number=2460, December 1998.

[117] S. 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.

[118] S. Goose and W. Hall. The Development of a Sound Viewer for an Open Hypermedia System. In *The New Review of Hypermedia and Multimedia*, volume 1, pages 213–231, 1995.

[119] S. Pizzi and S. Church. Audio Webcasting Demystified. *Web Techniques*, pages 55–60, August 1997.

[120] S. R. Mounce. A Brief Discussion of the Standard Music Description Language.

http://www.techno.com/SMDL.html, 1990.

[121] S. R. Newcomb. Multimedia Interchange Using SGML / HyTime (Part 1: Structures). *IEEE Multimedia*, pages 86–89, Summer 1995.

[122] S. R. Newcomb. Multimedia Interchange Using SGML / HyTime (Part 2: Principles and Applications). *IEEE Multimedia*, pages 60–64, Fall 1995.

[123] S. R. Newcomb, N. A. Kipp and V. T. Newcomb. The "HyTime" Hypermedia / Time-based Document Structuring Language. *Communications of the ACM*, 34(11):67–83, November 1991.

[124] S. Reich, J. Griffiths, D. E. Millard and H. C. Davis. Solent – a Platform for Distributed Open Hypermedia Applications. In *Proceeding of the 10th International Conference on Database and Expert Systems (DEXA)*, Florence, Italy, August 1999. (Berlin/Heidelberg/New York), LNCS, Springer.

[125] S. Reich, U. K. Wiil, P. J. Nürnberg, H. C. Davis, K. Grønbæk, K. M. Anderson, D. E. Millard, J. M. Haake. Addressing interoperability in open hypermedia: the design of the open hypermedia protocol. *The New Review of Hypermedia and Multimedia*, 5:207–248, 1999.

[126] E. D. Scheirer. Structured Audio and effects processing in the MPEG-4 multimedia standard. *Multimedia Systems*, 7:11–22, July 1999.

[127] T. Sikora and L. Chiariglione. MPEG-4 Video and its potential for future Multimedia Services. In *Proc. of IEEE - ISCAS Conference*, http://wwwam.HHI.DE/mpeg-video/papers/sikora/iscas.htm, June 1997.

[128] Synchronised Multimedia Working Group (SYMM-WG). *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*. World Wide Web Consortium (W3C), http://www.w3.org/TR/1998/REC-smil-19980615, June 1998.

[129] Synchronised Multimedia Working Group (SYMM-WG). *Synchronized Multimedia Integration Language (SMIL 2.0) Specification*. World Wide Web Consortium (W3C), http://www.w3.org/TR/smil20, September 2000. Last-Call public Working Draft.

[130] T. Dierks and C. Allen. The TLS Protocol Version 1.0. Technical report, Internet Engineering Task Force (IETF), ftp://ftp.nordu.net/internet-drafts/draft-ietf-tls-protocol-05.txt, November 1997.

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

[132] 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, January 1992. CERN, World Scientific.

[133] T. J. Berners-Lee, R. Fielding and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*. Internet Engineering Task Force (IETF), http://www.-ietf.org/rfc/rfc2396.txt?number=2396, August 1998.

[134] T. Meyer-Boudnik and W. Effelsberg. MHEG Explained. *IEEE Multimedia*, pages 26–38, Spring 1995.

[135] The Digital Audio-Visual Council (DAVIC) Home Page. http://www.davic.org, May 2000.

[136] The Digital Television Group (DTG) Home Page. http://www.dtg.org.uk, May 2000.

[137] The Digital Terrestrial Television Action Group (DigiTAG) Home Page. http://-www.digitag.org, May 2000.

[138] The Digital Video Broadcasting (DVB) Home Page. http://www.dvb.org/dvb_framer.htm, May 2000.

[139] The exXML Initiative. http://www.ebxml.org, March 2002.

[140] U. K. Wiil and K. Østerbye, Eds. Proceedings of the ECHT '94 Workshop on Open Hypermedia Systems. Technical Report R-94-2038, Association for Computing Machinery (ACM), Dept. of Computer Science, Aalborg University, September 1994.

[141] U. K. Wiil and S. Demeyer, Eds. Proceedings of the 2^{nd} Workshop on Open Hypermedia Systems, Hypertext '96. Technical Report ICS-TR-96-10, Association for Computing Machinery (ACM), Department of Information and Computer Science, University of California, Irvine, March 1996.

[142] V. Balabanian, L. Casey, N. Greene and C. Adams. An Introduction to Digital Storage Media — Command and Control. *IEEE Communications Magazine*, pages 122–127, November 1996.

[143] W. Hall. Ending the Tyranny of the Button. *IEEE Multimedia*, 1(1):60–68, Spring 1994.

[144] 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.

[145] W3C Working Group on Cascading Style Sheets and Formatting Properties. *Cascading Style Sheets, level 2 (CSS2) Specification*. World Wide Web Consortium (W3C), http://www.w3.org/TR/1998/REC-CSS2-19980512, May 1998.

[146] W3C XML Protocol Working Group. *SOAP Version 1.2 Part 0: Primer*. The World Wide Web Consortium (W3C), http://www.w3.org/TR/2001/WD-soap12-part0-20011217, December 2001.

[147] W3C XML Protocol Working Group. *SOAP Version 1.2 Part 1: Messaging Framework*. The World Wide Web Consortium (W3C), http://www.w3.org/TR/-2001/WD-soap12-part1-20011217, December 2001.

[148] W3C XML Protocol Working Group. *SOAP Version 1.2 Part 2: Adjuncts*. The World Wide Web Consortium (W3C), http://www.w3.org/TR/2001/WD-soap12-part2-20011217, December 2001.

[149] World DAB Forum (WorldDAB). *DAB - Frequently Asked Questions*. World DAB Forum (WorldDAB), http://www.worlddab.org/dab/whatis.htm, 1998.

[150] XML Linking Working Group. *XML Pointer Language (XPointer)*. World Wide

Web Consortium (W3C), http://www.w3.org/TR/1999/WD-xptr-19991206, December 1999.

[151] XML Linking Working Group. *XML Linking Language (XLink)*. World Wide Web Consortium (W3C), http://www.w3.org/TR/2000/WD-xlink-20000221, February 2000.

[152] XML Working Group. *Extensible Markup Language (XML)*. World Wide Web Consortium (W3C), http://www.w3.org/TR/1998/REC-xml-19980210, February 1998.