Overview of Multimedia Application Development

by

KIRK JONATHAN MARPLE

B.Sc., Kutztown University, 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

Department of Computer Science

We accept this thesis as conforming
to the required standard

_____

_____

_____

_____

THE UNIVERSITY OF BRITISH COLUMBIA

August 1996

TABLE OF CONTENTS

ABSTRACT

This thesis presents a survey of multimedia application development and a case study of an implementation of a multimedia application. The steps of multimedia application development are described including the planning of the application development, content acquisition and storage, and content delivery. The survey would be useful for multimedia developers and interactive media designers interested in gaining a broad knowledge of multimedia application design.

The case study presented is of a networked multimedia playback application and it illustrates the application development steps in the context of the application. The goals of the project were to provide synchronized video playback over a network while leveraging existing technology in order to support extensibility, portability, and interoperability.

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

To my wife, Laura, for putting up with the long nights at the computer and for persistently nudging me to finally complete this thesis.

To my thesis supervisor, Dr. Kellogg S. Booth, and faculty reader, Dr. Gerald Neufeld, for their time spent reviewing this thesis and their insightful suggestions.

To my employer, Microsoft Corporation, for providing an excellent work environment and for creating the development tools and operating system which were used to implement the thesis project.

INTRODUCTION

The ability to bring together various types of data - static or dynamic, visual or auditory - and render them for a user in a useful manner is one definition of the term *multimedia*. Multimedia has also been described as "Variety + Integration" [Williams91]. Multimedia integrates separate data types into a cohesive framework. The individual data types as well as the relationships between the data types are managed within this framework.

On a daily basis the computer user is involved in performing many different tasks. These tasks may be as basic as writing *electronic mail* (e-mail), or as complex as editing and assembling a segment of digitized video. It is common to refer to the objects manipulated in these tasks as *documents*. In a word processing task, the set of words and formatting commands the user enters is called a document. In a video editor, segments of video and audio, their spatial and temporal layout, and the commands which merge the data can also be thought of as a document - a *multimedia document*.

A word processing document with an embedded image is a multimedia document in its simplest form. But in common usage, multimedia documents must include at least one additional media type (other than text and static images). This will normally be audio or video.

A computer program which assists in the completion of a task can be termed an *application*. A *multimedia application*, such as a digital video editor, assists the video editing process by merging video and audio segments. By supplying capabilities such as

digital video effects, these applications can surpass the manual abilities of the user. In general, multimedia applications create, edit and store multimedia documents.

The display of a video segment or the playback of an audio segment can be described as the *delivery* of the data. This delivery normally occurs from a storage resource, such as a magnetic disk drive, to a display device, such as a computer monitor. For example, a video can be played from a hard disk drive through the system bus to the video monitor. Alternatively, the video can be played from a CD-ROM mounted in a remote computer over a *wide-area network* (WAN) to a video monitor.

Multimedia applications are designed to exploit existing networks and storage resources. Future plans for faster and wider-bandwidth networks and rapidly increasing quantities of storage resources should be taken into consideration. But multimedia applications need to be useful in the present day, not five years from now. As new resources surface in the marketplace, applications will begin to take advantage of them. However, the majority of computer applications must still run on computers, networks, and storage resources that were designed several years ago. This is why building scaleability and adaptive delivery techniques into multimedia applications today will lengthen their useful lifetime.

The process of creating a multimedia application can be subdivided into three steps:

- Planning the Application Development
- Content Acquisition and Storage
- Content Delivery

Application design encompasses both user interface design and system design. These two areas must be addressed together in order to provide a consistent, useful and successful application. The complete process of planning a multimedia application is analyzed in *Chapter 1: Planning the Application Development.*

Once the concept for the application has been finalized, the creator must decide what multimedia data will be used by the application. This multimedia data is commonly termed *content*. Content is generally acquired from a source such as a video camera or a microphone. The process of acquiring content from various sources and their associated storage formats is described in *Chapter 2: Content Acquisition and Storage*.

Once content is acquired and stored, the application creator must decide how the data will be delivered to the user. Also, the creator must decide how and on what platforms the content will be used. These issues plus inter-media synchronization and networking are discussed in *Chapter 3: Content Delivery*.

As an example of delivering network-based content, *Chapter 4: Case Study* illustrates the creation of a video playback application which supports the delivery of video data to inter-networked computers.

1. PLANNING THE APPLICATION DEVELOPMENT

Before beginning any task, one must decide what is to be accomplished, and what the end product should be. In this way, the process of creating multimedia applications is no different from any other task. The creator begins by creating a design for the application. The first attempts will likely be crude "back of the envelope" designs. But as the design process proceeds they will become more well-defined and specific. In addition, a set of specifications is created for most applications. These specifications will be the "recipe book" which the application's development will follow. Each application will have different specifications, and different application creators will design specifications differently. The degree of detail in the specifications will depend on the type of application being created and on the time allotted for its completion.

Before generating specifications, it is common to generate prototypes of the final application. A few examples of prototyping methods are storyboards, authored prototypes, and visual programming. Storyboards can be hand-drawn or computer-generated visual representations of an application's user interface and program flow. An authored prototype will normally provide a detailed example of a user interface but will only support a limited amount of programming logic, if any. The next step past an authored prototype would be to use a visual programming tool such as Microsoft Visual Basic or Borland Delphi. This type of tool allows the creator to layout an example of the user interface, but also include a subset of the programming logic for usability testing and demonstration purposes. Prototypes allow the application creator to brainstorm and validate various design principles with a minimum of development effort. Some tools even

allow visually programmed prototypes to be leveraged towards the final application by translating it into a low-level programming language such as C or C++.

1.1 User Interface Design

For applications which incorporate user interaction, the design of the user interface is of primary importance. It will affect the user's efficiency, as well as his satisfaction with and understanding of the application.

User interface design is a well-researched area, and as a result there are many opinions on user interface requirements. For example, it has been suggested that all interaction must be done via a simple device with intuitive operation and requiring little manipulation [Rosenberg92]. Devices such as touch screens are commonly used for inexperienced users because of their uncomplicated design. Typically, input devices such as mice and keyboards are not used when the application is aimed at inexperienced users, because these types of input devices require a substantial learning curve before efficient use is possible.

Intuitive operation is important for the inexperienced user because the best applications do not require instruction manuals or training. If the user can understand an application within seconds rather than minutes or hours, its usefulness is greatly enhanced. Similarly, applications which use simple manipulation methods do not require as much training as applications which have complex manipulation methods. Using a mouse to double-click an icon, in a *Windows-Icon-Mouse-Pointer* (WIMP) interface, requires more dexterity and skill than simply using one's finger to push a button on a touch screen (see Figure 1).

Figure 1: Example of icon-based user interface

This icon-based interface shows a Microsoft Windows 95 Explorer window containing five icons. The icons act as shortcuts to software applications and launch the referenced application when double-clicked.

It has been stated that application interfaces must present simple choices that do not require navigation [Rosenberg92]. An interface should present options to the user in such a way that, given the visible information, a decision can be made by the user. An example of this is an information kiosk with a touch screen, in which all available options are visible to the user at once on a single screen.

One interface technique which does not follow this guideline is the hierarchical menu structure (see Figure 2). With hierarchical menus, an option that a user desires will be hidden from view until its parent option is selected. Some training is needed to inform the user as to how to view the hidden options and how to choose the desired option. This is the navigation requirement. In hierarchical menus, an option is normally chosen with a

single-click at each level of the hierarchy. This interface technique is useful when many options must be provided to the user, but it has the drawback of being more complex than a “one finger” approach.

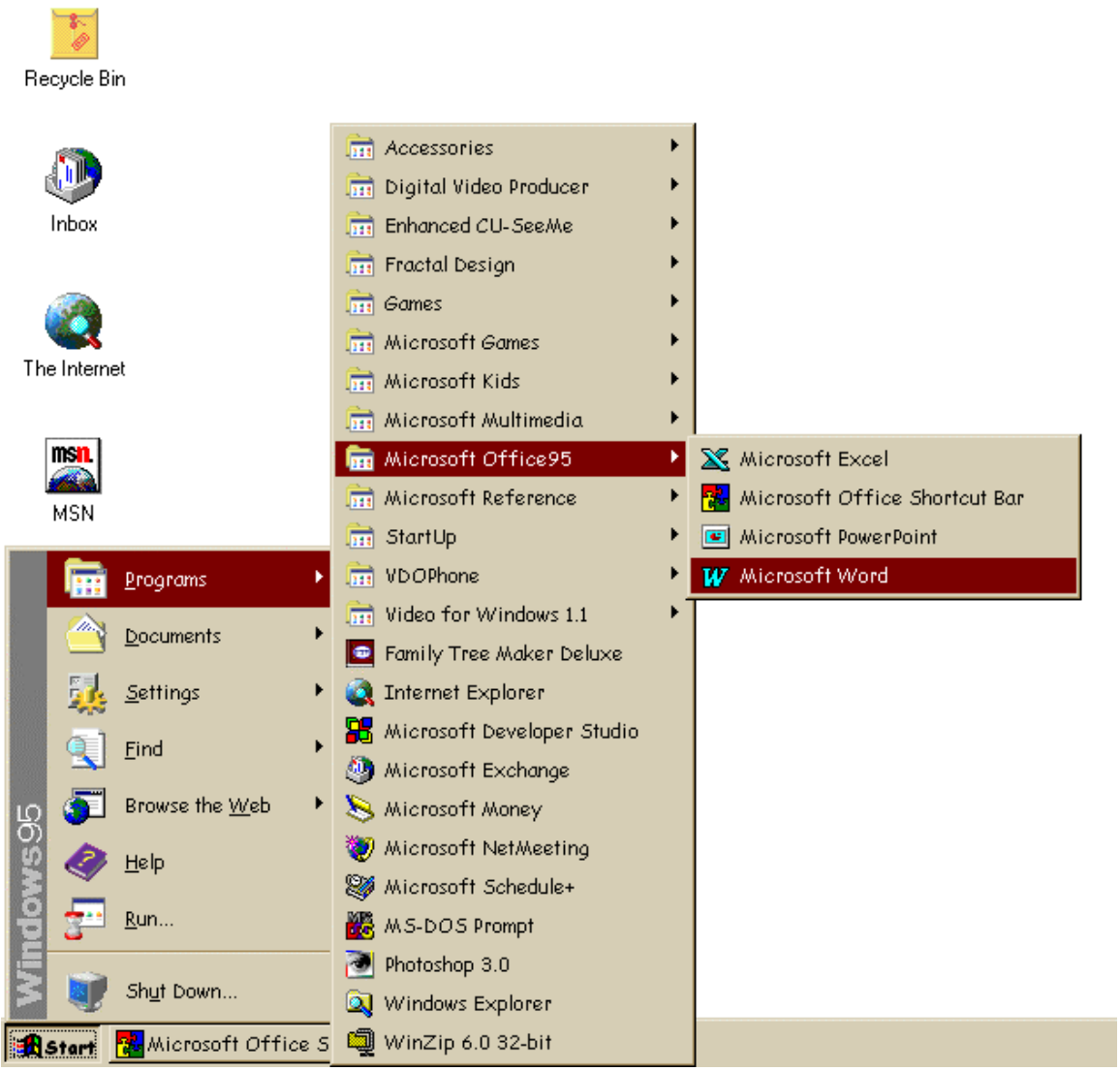

Figure 2: Example of hierarchical menu user interface

This hierarchical menu interface contains multiple sub-menus each of which contain a set of icons. The icons act as shortcuts to software applications and launch the referenced

application when single-clicked. By selecting the *Programs* item from the *Start* menu, and then selecting *Microsoft Office95*, the same five applications from Figure 1 are accessible. By highlighting *Microsoft Word* in this sub-menu and clicking on the item with the mouse, the application is launched. This type of menu interface allows access to many more applications in a smaller screen space than in an icon-based interface.

For inexperienced users, it is often a difficult task to navigate through the interface to find the desired option, so it follows that users will often make unintended selections. A well-designed application interface can help by limiting the user's chance of making incorrect selections [Rosenberg92]. For example, an interface can help users recover from mistakes by allowing them to undo mistaken selections. The application creator can focus the user's attention by only displaying currently available options. The interface can also aid the user in making the correct choice by not littering the screen with useless information.

Other user interface features that aid the user are real-time feedback, favorites, history lists and navigation context. As the user is navigating the interface, it helps to have real-time feedback of available options, help text, or shortcuts the user can take the next time to complete the same task more quickly. Favorites (a.k.a. bookmarks) are references to locations in an application which the user can save. This allows the user to quickly return to the same location in the application. A history list enumerates the previous locations the user has navigated to and allows him to return directly to a previously visited location. Similarly, navigation context provides the user with information about where in the application he is located, with respect to an overall application hierarchy or structure.

1.2 System Design

When designing an application, the application creator has to look beyond the user interface. The creator must consider the underlying technologies which will be used to deliver the media to the user, as well as the technology to deliver different media types. The various media types are discussed in *Chapter 2: Content Acquisition and Storage*, and delivery mechanisms are discussed in *Chapter 3: Content Delivery*.

In addition, one must consider the time vs. space trade-off when comparing the available delivery bandwidth to the available storage space. The delivery bandwidth may be supplied by a computer's system bus, an Ethernet LAN, an analog telephone line, a high-speed network connection such as *ATM* (Asynchronous Transfer Mode), or a WAN such as the Internet. The storage space may consist of primary, secondary or tertiary storage. The storage is categorized by its relative access time. Primary storage is a computer's resident *random access memory* (RAM). Secondary storage may be a computer's magnetic hard disk drive or removable optical disk drive. Tertiary storage is normally a slower device, such as an 8-mm magnetic tape drive.

When designing a multimedia application, security issues may be important if the application needs to protect its storage resources [Smith92]. When the storage resources are connected to a network, security is always an important concern. Password protection, encryption, and physical separation from outside networks are solutions which strengthen system security. Another concern is the difference between publishing security and viewing security. Limiting the users who can publish (write) data to your storage

resources and limiting the users who can view (read) the data on your storage resources are two different problems.

Two other goals are hardware independence and scaleability. Scaleability can lengthen the useful lifetime of an application. Building ad-hoc solutions to problems, and building an application tightly bound to a particular hardware platform are scenarios an application creator should avoid. An intermediate ad-hoc solution might solve a problem but will not provide for scaleability or hardware/software evolution. Similarly, building a hardware-dependent application might severely limit the lifetime of the application [Williams91].

An example of the latter mistake took place around 1991 when several multimedia applications were built which relied on the Intel/IBM ActionMedia II video capture and playback card. These applications relied on the hardware support for digital video which this card supplied. The ActionMedia II card had the capabilities of digitizing analog audio and video then playing back digitized video in real time. The card provided very high quality digital video but it cost more than US$1,500. As a result of its high price, sales of the capture card and of its complementary applications were poor.

Beginning around the same time, with the release of Apple QuickTime and Microsoft Video for Windows, software-only digital video became a very popular option. Users could play digital video at acceptable speeds without purchasing an expensive piece of hardware, because the playback was done solely by the CPU of the user's computer.

Applications which were dependent on the ActionMedia II card for digital video support could not adapt to use software-only digital video. Users wanted applications that could support both modes of playing back digital video (in software and in hardware) and the dwindling sales of these complementary applications reflected the users' reactions. Soon new applications were designed explicitly to take advantage of the new software-only digital video technology while still supporting the pre-existing hardware. Sales of these applications grew dramatically as a result.

1.3 Types of Applications

Multimedia applications can be creation tools, presentation tools, or editing tools. Examples of these classes of applications are multimedia authoring, multimedia conferencing, multimedia document browsing, and video annotation (see Table 1).

Application	Description
Authoring	Used to build end-user applications from a set of multimedia data.
Conferencing	Used for one-to-one, one-to-many or many-to-many communications using real-time audio or video data transmission.
Document browsing	Used to view pre-published multimedia content over a network.
Video annotation	Used for the real-time annotation of video data and for the detailed analysis of the collected data.

Table 1: Classes of multimedia applications

In the following sections, for a variety of applications, we analyze the user interface, the types of multimedia data used, the networking and storage resources used, and the manner in which the multimedia data is delivered to the user.

1.3.1 Multimedia Authoring

One of the most common uses of multimedia is in the creation of *authored applications*. Examples of these applications are information kiosks, consumer entertainment software, and educational courseware. These authored applications are termed *titles*.

A title is built from a set of multimedia data using a *multimedia authoring application*. A typical multimedia authoring application allows the title creator to display multimedia data, to accept user input, and to create navigation paths for the user. Examples of multimedia authoring applications are Macromedia Director and AimTech IconAuthor. A user views a title by moving from one visual display to another, a process commonly called *navigation*. Most authoring applications allow the creation of titles in which the user can randomly move between visual displays, also called *hyperlink navigation*. Highlighted words, buttons, and areas within images are used as *hyperlinks* in titles. When a user clicks on a hyperlink, the title navigates to another location within the title.

A common type of authored application is *courseware*, which is customized software for a particular instructional or research domain [Drapeau91]. It can include *Computer-Based Training* and *Computer-Adaptive Training*. The computer version of the *Graduate Record Exam* (GRE) is an example of Computer-Adaptive Training courseware [Educational95]. As the user completes the questions in the examination, the courseware generates a user model which provides an extrapolated final examination score. The courseware chooses the upcoming questions based on the user model in order to tailor the examination to the user's ability. Thus it is designed to generate a valid final score from a smaller number of questions than the written test .

There also exists another type of authoring application which allows only a *linear navigation* of the title. Linear navigation permits one-dimensional navigation of a title, so the user can only move forward or backward in the title as with a slide show. These applications are typically called *multimedia presentation applications*. Microsoft PowerPoint and Aldus Persuasion are examples of this type of application. Titles made with multimedia presentation applications are known as *presentations*.

A feature normally not found in multimedia presentation applications is the processing of user input. Presentations usually contain no interactive elements other than what is used for linear navigation. Other types of titles can process user input for various tasks: an information kiosk may accept a user's choice of which stored musical selection to play, a courseware title may accept a user's answer to a question posed by the title, or a consumer entertainment title may accept a user's input for determining which virtual world the user would like to explore next.

Multimedia authoring applications also allow the incorporation of programmatic logic into the title. This allows the title to perform calculations and make decisions during the execution of the title. The multimedia authoring application will usually support some form of programming language. This may be a proprietary script-based language, such as Macromedia Director Lingo, or a common programming language, such as Microsoft Visual Basic. At most, multimedia presentation applications will give the user an 'if-then-goto' type of logical capability. The logical capabilities of multimedia authoring applications greatly exceed the limited logical capabilities found in multimedia presentation applications.

Each individual visual display can be thought of as the temporal partitioning of the application. These displays are commonly called *slides*, *frames*, or *scenes*. Most multimedia presentation applications use the term slide, while some multimedia authoring applications use the term scene. The collection of slides, frames, or scenes in a multimedia application can be termed a *movie*, *title*, or *presentation*.

An alternate form of temporal partitioning is based on the *segment*, which is a temporal element which contains some semantic significance. A *chunk* is a segment of arbitrary length. The smallest addressable unit which represents continuous action in time and space is termed the *shot*. The shot consists of one or more frames generated and recorded contiguously. A *sequence* is a collection of shots which contain temporal, spatial, and perceptual continuity and form a natural unit [Davenport91]. It should be noted that these terms are drawn from cinematic techniques: cinematography, directing, and editing.

A multimedia authoring application usually incorporates several components: media manager, media editors, title editor, and title viewer. The *media manager* provides a repository for the multimedia data used by the title. This can be thought of as a database which stores the audio, video, and image data the title creator uses while authoring the title. *Media editors* are used for editing and preparing the multimedia data. The *title editor* is used for editing the spatial and temporal layout of each scene. This involves placing the multimedia data within the scene as well as providing the ability to manipulate the multiple scenes in the title. The *title viewer* provides the ability to view a finished title, but not to edit it. This component is shipped with the title in cases where the title creator does not want the user to be able to change or alter the title. In some multimedia

authoring applications, the title can be optimized for playback speed and storage size. But, after optimization, the title will lose the capabilities for further editing.

Some multimedia authoring applications dictate a managerial approach to development. They provide a single method of managing multiple forms of media, which gives the benefits of a common user interface and ease of use, but limits the flexibility and power of managing any one media type. Because of this common approach to authoring, these applications often only support a limited set of media. In addition, this managerial approach will usually be linked to a single style of authoring and will not give the option of authoring in other styles [Drapeau91].

Some other key issues for a multimedia authoring application are extensibility, portability, and interoperability [Drapeau91]. *Extensibility* is as important for the progressive development of the applications as it is for its adjustment to changing software and hardware platforms. A component-based approach is optimal for logically separating the hardware- and software-dependent portions of the application and allowing the application programmer to redesign and/or rewrite out-of-date portions of the application.

In the current computer industry, *portability* is an important issue for multimedia authoring applications. The ability to author a single title and, without any recoding, view the title on multiple software and hardware platforms is a major selling point. Porting a title to other platforms takes time away from refining the application design and its content. Also the title loses any platform-specific capabilities by being designed to the lowest common denominator.

In networked applications, *interoperability* can be acquired by defining strict protocols by which multiple machines will communicate. The definition of standard protocols allows multiple machines with differing hardware and software platforms to interoperate and cooperative. Examples of this are multimedia conferencing (T.120 and RTP), the World Wide Web (HTTP and HTML), and Internet electronic mail (MIME, SMTP, and POP3) and news delivery (NNTP).

1.3.2 Multimedia Conferencing

As the available bandwidth between network-connected sites grows, the use of multimedia conferencing has expanded. It began as early as 1988 with the Etherphone system [Terry88]. Multimedia conferencing can be as simple as Internet Voice Chat which transmits pre-recorded audio segments between a pair of participants using Windows-based computers [Ahrens94], or as complex as CU-SeeMe which supports multi-party video and audio conferencing between both Macintosh and Windows-based computers [Cogger94] (see Figure 3). Other similar applications are SGI InPerson and Microsoft NetMeeting.

Many multimedia conferencing applications are built upon the TCP/IP networking protocol, which is the *lingua franca* of the Internet. Other applications support network connections using physical links such as *POTS* ("Plain Old Telephone Service"), *ISDN* (Integrated Service Digital Network), Ethernet cable, or optical fiber.

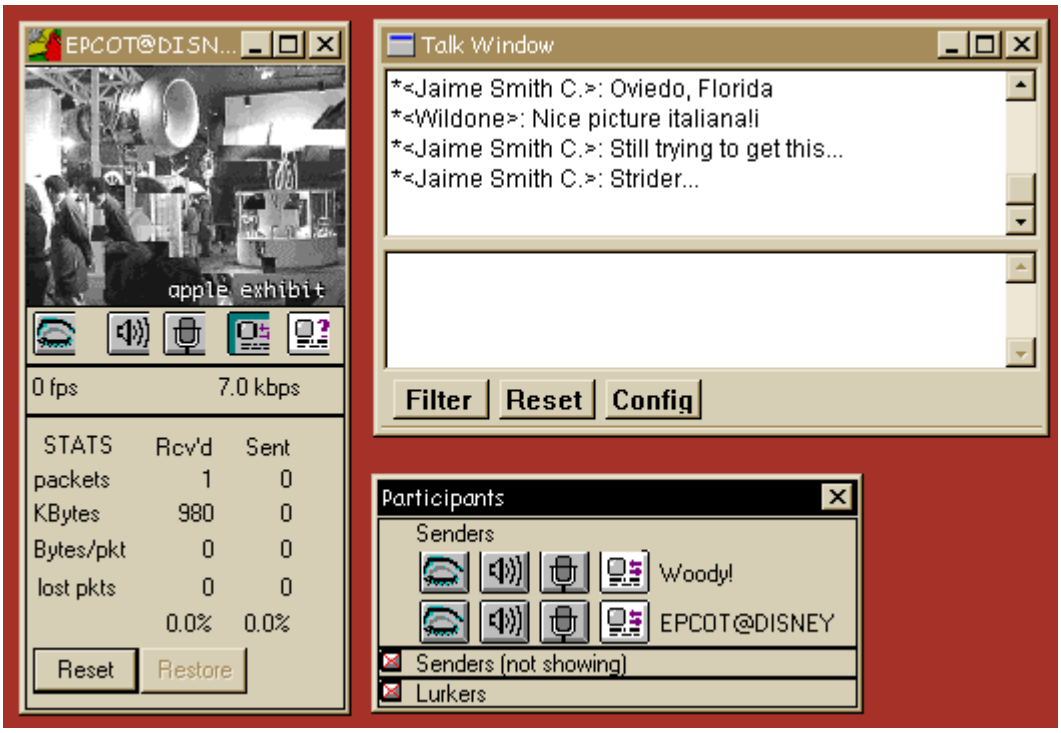

Figure 3: CU-SeeMe conferencing application which renders compressed video and audio streams and out-of-band text "chat" messages.

Key factors in the movement of multimedia data, especially when it is used for multimedia conferencing, are the asynchronous exchange of the data, real-time constraints on the movement, and the fact that the communication system must provide for the timely transmission of the data [Ahuja92]. The pipelining effect of asynchronous data exchange is key to the timely transmission of multimedia data. Newer advances in multimedia data transmission bypass the send/receive/acknowledgment cycle of the TCP/IP protocol by not requiring the acknowledgment of sent multimedia data packets. In this situation, alternate provisions are made for the reliable transmission of the data and packet retransmission that TCP/IP would typically provide. More discussion of these techniques occurs in *Chapter 3: Content Delivery*.

Three features of most multimedia conferencing systems are the user interface, call control, and multi-point communications [Ahuja92]. Every application generally will have a user interface and call control, by which the user can specify the endpoint of the conference. Some applications support point-to-point communications, which is analogous to a private phone call, while others support multi-point communications, which is analogous to a conference phone call.

Multimedia conferencing systems can be used in conjunction with other network-aware applications, such as word processing applications. If the word processing application possesses concurrent multi-user capabilities, it can be used with a multi-point video (and/or audio) conferencing application for cooperative document authoring [Baecker92].

1.3.3 Multimedia Document Browsing

As a result of the explosion of activity on the Internet, there recently has been a corresponding explosion in both the supply and demand of multimedia documents. The *World Wide Web* (WWW) is a global publishing system which uses the resources of inter-networked computers on the Internet. Users can create their own multimedia documents (*Web pages*) for publishing. These documents can be published on a *Web server* - a computer which is directly connected to the Internet and which supports *HTTP* (HyperText Transfer Protocol). A *Web browser* is used to access Web pages published on remote computers. The World Wide Web Initiative, based at *CERN* (European Particle Physics Laboratory) in Geneva, Switzerland, was begun in 1990 and is responsible for the protocols that make up the Web [Lemay95].

Web pages support textual information, images, and other media types by the use of *viewers*. Viewers are applications whose sole purpose is to display multimedia data. Viewers commonly exist for displaying video and sound data. Textual information must be formatted using *HTML* (HyperText Markup Language, which is derived from SGML). Images are typically in *GIF* (Graphics Interchange Format) or *JPEG* (Joint Photographic Experts Group) format. Other media types, such as *MPEG* (Motion Picture Experts Group) or Apple *QuickTime* format video, can be downloaded and viewed on the user's local machine if an appropriate software player is present.

A Web page can be authored by inserting the HTML codes by hand into an ASCII document, or by using a HTML editor to format the page in a *WYSIWYG* ("what you see is what you get") fashion. Several WYSIWYG HTML editors are currently available such as Microsoft Internet Assistant for Word and SoftQuad HoTMetaL.

Images can be inserted directly into a Web page (*inline images*) by specifying a dynamic link to the image file. The Web Browser is responsible for displaying these inline images. Audio and video files can also be inserted, but these data types will use an external viewer to render the data. Links to another location in the current Web page, to another Web page on the current machine, or to another Web page on a remote machine can be specified in the HTML document. These links are all dynamic links - i.e. a reference to the location of the required file is embedded in the Web page itself, not the actual data from the file.

Figure 4: Microsoft Internet Explorer WWW browser showing inline images, formatted text, and hyperlinks embedded in the HTML page

Web pages can be browsed on Microsoft Windows, Apple Macintosh or various UNIX platforms with Microsoft Internet Explorer (see Figure 4), Netscape Communications Netscape Navigator, NCSA (National Center for Supercomputer Applications) Mosaic, Lynx, or other freely available public domain browsers. These applications provide methods for navigating the hyperspace created by Web pages and their hyperlinks. Common features are: history lists, so the user can see past links; configurable viewers, so the user can add viewers for other data types at a later time; and bookmarks, so the user can easily return to an interesting page in future.

1.3.4 Video Annotation

One multimedia application that is designed to assist users is the *video annotation system*. A video annotation system provides for real-time annotation of video data and for the

detailed analysis of the collected data [Harrison92]. The basic functions of a video annotation system include the recording of events such as mouse and keypress logging, symbolic annotation of multimedia data (video snapshots, sound bites), spatial viewing of temporal events, synchronization of streams of different media, and the reordering of video segments. Spatial viewing of temporal events refers to the on-screen display of multiple video snapshots using techniques such as video icons. Commercial digital video editors provide several of the desired features of a video annotation system [Mackay89].

The user interface of the system must allow for continuous visual attention in order to provide consistent analysis. The use of non-speech auditory and visual feedback enhances the analysis process. The user interface should also give control over the source media by direct manipulation. Often this is done using a virtual VCR-type control [Harrison92].

2. CONTENT ACQUISITION AND STORAGE

All multimedia applications have at least one feature in common: the ability to display one or more multimedia data types. This image, audio, or video data is called *content*. A multimedia presentation application can integrate content such as bitmapped images, formatted text, a *MIDI* (Musical Instrument Digital Interface) audio stream, or live video streams. A multimedia conferencing application, for example, supports only networked audio and video streams as content.

Content can be separated into static or dynamic media types, and into digital or analog media types. *Dynamic data* changes over time (i.e. the notes played by a MIDI audio stream change over time), while *static data* remains constant (i.e. an image which does not change during a multimedia presentation). *Digital media* is maintained in a digital form and the data can be accessed and stored by devices such as a computer file system. *Analog data* can only be accessed and stored by an analog storage device such as a VCR or an audio cassette recorder.

Media types can be converted from analog to digital by a process called *digitizing*. Video can be acquired from an analog video source, such as a video camera, and stored in a digital format, such as MPEG. It is also possible to convert digital media to analog. A digital audio file can be played by a computer's sound card and output through an analog cable to speakers.

2.1 Finding Content

There are many ways to find content for a multimedia application. For examples of sources for image, audio, and video content, see Table 2.

Format	Media type	Source
Analog	Image	Computer-controlled 35 mm slide carousel
	Audio	Live audio CD-Audio
	Video	Live video Video tape LaserDisc
Digital	Image	Stock footage Image capture Scanning Digital photography Kodak PhotoCD
	Audio	Stock footage Audio capture CD-Audio MIDI audio Live MIDI audio
	Video	Stock footage Video capture

Table 2: Sources of image, audio and video data in analog and digital formats

An analog media source provides the visual or auditory effect of multimedia without requiring digital storage space on the user's machine. There is still a storage requirement however, since the user's machine must support some form of analog storage. Video LaserDisc technology and magnetic video tape are analog video sources. The analog video output can be directed into the user's computer by using a *video overlay board*. Video overlay boards mix an external analog video signal with the video stream from a computer's video display board.

Any existing technology that plays audio through an analog cable, such as a radio receiver or a record player, is an analog audio source. In multimedia applications, the microphone and the compact disc are two of the most common analog sources. The microphone is useful for capturing the user's speech or ambient sounds. The compact disc is a digital technology but is most commonly used as an analog source. This is because the digital audio data is converted to analog data at the time the compact disc is played. A compact disc player will normally send the analog audio data through the same cable as a radio receiver or record player.

Digital images are the most common form of digital multimedia data. Digitized images are easily accessible from online services or can be purchased in bulk on CD-ROM or floppy disk. These prepackaged collections of digitized images or other digital media are called *stock footage*. Another term for stock footage is *clip art*. Multimedia application developers commonly need access to artwork for use in their titles. Stock footage is an efficient way for them to gain access to a wide range of media at a low cost.

Other ways of acquiring digital image data are image capture, image scanning, and digital photography. Image capture entails using an image capture or video capture card to digitize an incoming analog video signal. This analog signal will often come from a video camcorder or other video camera. The capture card will allow a user to digitize specific frames from the video stream and save them to the user's disk drive for later use in a multimedia application. An image scanner can be used for digitizing static artwork or photographs. The artwork is laid face-down on the glass surface of the scanner and, utilizing a method similar to Xerox machines, the image is progressively digitized and

stored to the user's disk drive. Digital photography removes a step from the image scanning method of acquiring digital image data. In a single step, the digital camera can photograph a scene and store the digital image data to a device in the camera itself. The digitized images can later be copied to the user's disk drive.

A recent technique of acquiring digital image data is the *Kodak PhotoCD*. With this method, the user can take photographs with existing film-based cameras and send the film to a PhotoCD film processing center. The PhotoCD center processes the film into the desired print or slide format, and simultaneously digitizes the pictures into a proprietary PhotoCD format. The digital PhotoCD images are stored onto a CD-ROM which is returned to the user with the photographs. For a relatively low cost, a user can have high-quality digital images generated without the expense of buying an image scanner or image capture card.

Next to digital images, digital audio is the most common form of multimedia content. Sound cards for computers that can record and playback digital audio are commonplace and affordable. Most personal computers come with digital audio support standard. Digital audio can be found in similar locations as digital images - online and in clip media collections. Also, a computer's sound card can digitize audio from various sources. Common sources which can be connected by an analog audio cable are microphones, radio receivers, and magnetic tape players. Another form of digital audio is MIDI. MIDI audio consists of the actual musical note data as created by MIDI-capable musical instruments such as synthesizers or digital guitars. This MIDI audio data can be stored in

digital form on a computer's hard drive or can be input live as the MIDI-capable instrument is played.

Compact discs store audio data in a digital format and typically they are played as analog audio through audio speakers. It is possible to copy the data in native digital form from a CD-Audio disc to a digital storage device, but this is not commonly done. However, this technique can be useful for preserving the original audio quality when making a copy of the audio data. Otherwise, the analog version of the played CD-Audio disc would have to be re-digitized by a sound capture card.

The popularity of digital video is blossoming. As hardware development strives to keep up with the technical requirements, it will probably get even more popular. Clip media and stock footage collections are also good sources for digital video. The stock footage houses that used to provide stock film footage are converting their collections into digital formats. This is a great source of video for the multimedia application creator. An application creator can also capture his own video using a video capture card. These cards are similar to their audio-only counterparts in that they take an analog source and digitize it for storage on a computer's hard drive. The available video capture cards affordable by the average consumer can only digitize partial TV-resolution (320x240) at 30 frames/sec, but high-end professional cards can digitize full TV-resolution (640x480) video at 60 fields/sec.

Standard NTSC television signals are *interlaced*, meaning that each video *frame* is divided into two separate *fields* of alternating scanlines. The resulting fields are displayed

sequentially, such that what was originally a 30 frames/sec refresh becomes 60 fields/sec at half the vertical pixel addressability [Scott96].

It must be noted that the capabilities of the application creator's computer platform will determine whether or not real-time capture of an analog video signal is possible. The storage device must be able to handle the incoming data rate from the capture device. For example, a captured video stream of 320x240 pixels at 24 bits/pixel and 30 frames/sec will generate 6.9 Mbits/sec of data. The latest *EIDE* (Enhanced Integrated Drive Electronics) and *SCSI* (Small Computer System Interface) hard drives can write incoming data to disk at this rate, but older *IDE* (Integrated Drive Electronics) hard drives can not.

2.2 Preparing Content

Once the application creator has selected a set of content for a multimedia application, he will need to prepare it before use. Images can be cropped, scaled, filtered, color-corrected, or even cut and pasted with other images to make new composite images. Audio can be filtered to remove background noise, have effects added (reverb, echo, etc.), and be cut and pasted with other audio segments to make new composite audio segments. Video is essentially a sequence of images, so it follows that each frame of video can be cropped, scaled, filtered, color corrected, or cut and pasted with other frames of video (or other images). In addition to image-based operations, multiple-frame effects can be applied to video. When image processing operations like sharpening or warping are applied, the parameters of the effect can change with time.

Once the selected content has been prepared, it typically will be converted into a compressed digital format for storage. The next section describes methods for digitizing and compressing image, audio and video content and lists digital storage formats for each content type.

2.3 Storing Content

Once the content for a multimedia application has been acquired and prepared, the application creator must decide on the storage format which best fits the application. Most content used by multimedia applications will be digital. Two common exceptions are CD-Audio and live video, but otherwise all content will need to be stored in a digital format. As was discussed in the beginning of this chapter, content can be acquired from an analog or a digital source. Analog sources need to be digitized before they can be stored in a digital format. As shown in Table 3, the space required for storing various types of content in a digital format differs greatly. A common way to reduce the storage space requirement is by compressing the raw digital data. The proper compression technique to be used varies widely depending on the type of content. All storage space requirements listed in Table 3 are for uncompressed data.

Content type	Description	Storage space
Encoded text	screen size: 768x512 pixels, character size: 8x8 pixels, 2 bytes/character	12 kB
Vector graphics	screen size: 768x512 pixels, 1 byte/line	2.8 kB
Bitmapped images	screen size: 768x512 pixels, character size: 8x8 pixels, 256 colors/pixel	384 kB
Digitized speech	sample rate: 8 kHz, 8 bits/sample	8 kB/sec
Digitized audio	sample rate: 22 kHz, 16 bits/sample	88 kB/sec
Digitized video	sample rate: 10 MHz, frame rate: 25 frames/second 24 bits/sample	30 MB/sec

Table 3: Storage demands of uncompressed digital media types [Williams91]

2.3.1 Image

Images are two-dimensional digital representations of a three-dimensional world. The images are broken up into discrete elements called *pixels* (picture elements). The image is a two-dimensional matrix of pixels, where each pixel contains some quantity of color data. This quantity of color data is called the *bit depth*, because it uses a fixed number of bits to represent the color information at that picture location. The bit depth will not change over the two-dimensional space of the picture. The dimensions of the matrix are called the *width* and *height* of the image [Foley90]. The color information in images normally fits into the *RGB* (red-green-blue) color space, but images can also use the *YUV* (luminance-chrominance) or *CMYK* (cyan-magenta-yellow-black) color spaces. Common bit depths are 1 bit (bilevel), 8 bit (greyscale or palettized), and 24 bit (true color). One-bit images represent two colors (normally black and white). Eight-bit images can represent 256

colors, where the 256 colors are shades of grey or indices into a color palette. By using a color palette, images can store a reference to 24 bits of color information in only 8 bits of data. Twenty-four-bit images can represent 16,777,216 colors.

Digital image content can be stored in numerous formats. Each format contains various information about the image in addition to the image data itself. These formats are standardized so that an image created by one application can be read and displayed by any other application which supports the same format. Some of the most common formats are GIF, *TIFF* (Tagged Image File Format), and JPEG. Each of these formats stores the image data in a different way. Table 4 provides an overview of many of the common image formats.

Name	Owner	Bit depths supported	Compression types supported
BMP	Microsoft Corporation	1, 4, 8, 24	none, RLE4, RLE8
GIF	CompuServe Information Systems	1, 2, 4, 8	LZW
JPEG	Joint Photographic Experts Group	8, 24	none, JPEG
PCX	ZSoft Corporation	1, 4, 8, 24	RLE
TGA	Truevision Corporation	8, 16, 24, 32	none, RLE
TIFF	Aldus Corporation	1, 4, 8, 24	none, LZW, RLE, Macintosh PackBits, CCITT 1D, CCITT group 3 and 4

Table 4: Compressed digital image storage formats [Media91] [Murray95]

Three of the most common compression techniques are *RLE* (Run-Length Encoding), *LZW* (Lempel-Ziv Welch), and JPEG. Run-length encoding takes advantage of redundancy across scanlines of an image. Repeated pixels are replaced by a repeat count and the repeated pixel value [Foley90]. RLE compression is most efficient on “artificial”

images, rather than “real-world” images. Real-world images exhibit more randomness and a wider variation in pixel values. A frame of cartoon animation which uses solid colors is most appropriate for RLE.

LZW uses an entropy reduction technique known as *string encoding*. String encoding is a process which assigns codes to groups of data items repeated in the data stream. LZW is a *lossless*, adaptive compression algorithm that works well with various types of images. A lossless compression algorithm means that none of the original image data is lost in the compression process. Palettized and bilevel images typically exhibit high compression ratios (10:1), while true color and greyscale images generally get only minimal compression (3:1) [Media91].

JPEG encoding is designed for compressing full-color or greyscale images of real-world scenes. It works well for photographs, but does not work well for line drawings or cartoons. It is a *lossy* form of compression, which means that some of the original image data is lost in the compression process. The degree of lossyness can be varied by adjusting compression parameters. JPEG stores 24 bits per pixel of color information, but can generally achieve a compression factor between 10:1 and 20:1 without a loss of visual image quality. A compression factor between 30:1 and 50:1 is possible with small to moderate quality loss [Lane95].

2.3.2 Audio

Digital audio data represents a time-sampled version of an analog audio waveform. The analog waveform is sampled in order to produce a sequence of values which correspond

to the amplitude of the waveform at precise points in time. The stream of digitally encoded amplitude data is generated by an *analog-to-digital* (A/D) converter. Multiple channels of audio waveforms can then be multiplexed into a single data stream. For playback the stream of digitally encoded audio data is decoded to recover the amplitude information at each sample point. An analog waveform is reconstructed from this amplitude information by a *digital-to-analog* (D/A) converter [Pohlmann89].

There are many techniques available to encode analog audio signals digitally. They vary widely in terms of required bandwidth, signal-to-noise ratio and accuracy. Techniques exist such as *Pulse Amplitude Modulation* (PAM), *Pulse Position Modulation* (PPM), and *Pulse Width Modulation* (PWM). These techniques use variations in pulse amplitude, time position and width, respectively, to represent the analog signal's sample value. Another technique, *Pulse Number Modulation* (PNM), is similar to *Pulse Code Modulation* (PCM), but PNM generates a string of pulses where the pulse count represents the amplitude, while PCM encodes the pulse chain in order to greatly reduce the bandwidth required to store the data. PCM is a very common form of digital encoding and is used in the CD-Audio standard format [Pohlmann89].

Adaptive Delta Pulse Code Modulation (ADPCM) builds on the strengths of PCM by adding excellent data compression. ADPCM is a differential encoding system, which means that only the differences between successive samples are stored. By encoding only the differences between samples, a high level of data compression is achieved. In this technique, a fast sampling rate is important so the differential encoding can closely track the analog signal. Simple Delta Modulation techniques use a 1-bit correction code to

predict whether the signal will rise or fall at the next sample point. ADPCM expands on this technique by using a 4-bit or 8-bit code to represent correction information. This allows for 16 or 256 levels of correction information to be encoded. More control over the correction information results in greater accuracy in encoding the analog waveform [Pohlmann89].

Two audio encoding techniques that are commonly used for speech encoding are μ-Law and A-Law. In PCM encoding, the amplitude levels are quantized at uniform intervals. But for some signals, such as speech, it is preferable to have quantization levels for high-amplitude signals spaced far apart and low-amplitude signals spaced closer together. μ-Law and A-Law encoding systems both use a logarithmic encoding system to provide these characteristics. Although very similar, these two encoding systems use different logarithm-based equations for their encoding. As a benchmark, an 8-bit implementation of μ-Law encoding can achieve a small signal to noise (S/R) ratio and dynamic range equivalent to that of a 12-bit PCM encoding system [Pohlmann89].

Table 5 shows the data rate for various encoding methods at different sample rates. The data comes from the *Sound Recorder* application which ships with the Microsoft Windows 95 operating system.

Different computer platforms have different standards for storing digital audio data. On the Microsoft Windows platform applications use the WAV format. On the Macintosh the AIFF format is used, and on UNIX platforms the AU format is common. The WAV and

AIFF formats can store PCM, ADPCM, μ-Law and A-Law encoded audio data, but the AU format stores μ-Law data only [van Rossum94].

Encoding method	Bits per sample	Sample rate	Mono data rate	Stereo data rate
A-Law; μ-Law	8 bits	8.000 kHz	7 kb/sec	15 kb/sec
		11.025 kHz	10 kb/sec	21 kb/sec
		22.050 kHz	21 kb/sec	43 kb/sec
		44.100 kHz	43 kb/sec	86 kb/sec
ADPCM	4 bits	8.000 kHz	3 kb/sec	7 kb/sec
		11.025 kHz	5 kb/sec	10 kb/sec
		22.050 kHz	10 kb/sec	21 kb/sec
		44.100 kHz	21 kb/sec	43 kb/sec
PCM	8 bits	8.000 kHz	7 kb/sec	15 kb/sec
		11.025 kHz	10 kb/sec	21 kb/sec
		22.050 kHz	21 kb/sec	43 kb/sec
		44.100 kHz	43 kb/sec	86 kb/sec
	16 bits	8.000 kHz	15 kb/sec	31 kb/sec
		11.025 kHz	21 kb/sec	43 kb/sec
		22.050 kHz	43 kb/sec	86 kb/sec
		44.100 kHz	86 kb/sec	172 kb/sec

Table 5: Compressed digital audio storage requirements [Microsoft95]

A common storage medium for digital audio data is the compact disc. Compact discs store a 16-bit stereo PCM signal which is sampled at 44.1 kHz. This format is commonly called *CD-Audio*. An alternative form of compact discs is called *Compact Disc-Interactive* (CD-I). This format can store audio and video information, as well as computer data (text or binary). The audio data can be stored in one of five modes, depending on the fidelity required and the storage space available [Pohlmann89] (see Table 6).

Audio mode	Encoding format
CD-Audio	16-bit PCM, 44.100 kHz, stereo
Hi-Fi (level A)	4 bit ADPCM, 44.100 kHz, stereo (LP quality)
Mid-Fi (level B)	4 bit ADPCM, 44.100 kHz, mono (FM quality)
Speech (level C)	4 bit ADPCM, 22.050 kHz, mono (AM quality)
Text-to-speech	phonetic coding (synthesized speech quality)

Table 6: CD-I audio modes [Pohlmann89] [Luther89]

2.3.3 Video

Digital video is encoded in a similar manner to digital audio. An analog video waveform is time-sampled to produce a digitally encoded representation of the original waveform. Video waveforms may contain one or more signals, depending on the video format.

Broadcast video waveforms consist of two signals: luminance and chrominance. *Luminance* is the brightness of the signal, which represents a gradient from black to white. *Chrominance* is the color part of the signal, which represents the hue and saturation. The original black-and-white method of television broadcast transmitted the luminance signal only. When color television was invented, the chrominance signal was piggybacked on top of the luminance signal [Rubin91].

The *Y/C* format transmits the luminance and chrominance in two separate video signals. The *RGB* analog format separately transmits the red, green, and blue components [Luther89]. The RGB color information can be converted into other equivalent color spaces, such as *YUV* or *YIQ*, which are transmitted in three separate video signals.

The RGB, YUV, and YIQ formats are called *component video signals* because they transmit three separate video signals. The Y/C format is called a *pseudo-component video signal* because two of the component signals (U and V) are combined into one signal (C).

Another video signal format is the *composite video signal*. In this format, all three component video signals are mixed into one signal. This format is used in the *NTSC*, *PAL*, and *SECAM* broadcast video standards that are used throughout the world [Rubin91] [Focal69].

As previously described for audio, video signals are also digitized using an A/D converter. The output sample values from the digitizing process are converted into color values. At each sample point, or pixel, the color value can be encoded using a varying number of bits per pixel. This provides a method of varying the accuracy of the color reproduction of the original analog video signal. In other words, more bits per pixel provide a more accurate representation of the source video signal.

Digital and analog video signals are frequently stored on magnetic tape. Table 7 shows a listing of common video signal formats with their corresponding storage formats. Each storage format is listed with its commonly known product name and the type and width of the tape used.

Video signal format	Storage format
Component	D1 (19mm digital) BetaSP (1/2″ analog) MII (1/2″ analog) Betacam (1/2″ analog)
Pseudo-component	S-VHS (1/2″ analog) Hi-8 (8mm analog)
Composite	D2 (19mm digital) D3 (19mm digital) 1″ type C (1″ analog) 3/4″ U-matic (3/4″ analog) 3/4″ SP (3/4″ analog) 8mm (8mm analog) VHS (1/2″ analog)

Table 7: Video signal formats with corresponding storage formats [Rubin91]

In order for an application creator to use digital video, it must be stored in a form appropriate to be displayed on the application's computer platform. Most computer platforms have their own standard format for digital video files. On the Microsoft Windows platform, the standard is called *Audio-Video Interleaved* (AVI), and on the Macintosh platform, the standard is called QuickTime. Each of these standards can contain various types of video data: different bit depths, resolutions, and compression types. File format and compression type are not necessarily linked. There exists an MPEG file format which only holds MPEG compressed data, but AVI and QuickTime formats can also contain MPEG compressed data.

The storage requirements of digitized video are more rigorous than any other data types. For example, the storage space required for digitized NTSC format video is 45 Mb/sec. CCIR 601 format video requires 216 Mb/sec, CIF format requires 36 Mb/sec, and QCIF format requires 9 Mb/sec. Standard NTSC format video uses a resolution of 525

horizontal lines and 360 pixels per line. CCIR 601 format uses 720 horizontal lines and 480 or 576 pixels per line (depending on NTSC or PAL format). Primarily utilized for video telephony, the CIF and QCIF formats use resolutions of 360x288 and 180x144, respectively [Fox89].

Because of the extreme storage requirements of uncompressed data, data compression is used to reduce the data rates. The inherent redundancy present in video streams means that two methods of compression can be used. *Intraframe compression* can be applied to reduce the spatial redundancy within a single frame of video, and *interframe compression* can be applied to reduce the temporal redundancy between frames of video.

Intraframe compression encompasses the preprocessing steps of filtering the image data, color space conversion (typically RGB to YUV), digitizing, and scaling. In addition, transformations, quantization, and encoding are applied. Filtering the image data removes high-frequency noise and averages the image pixels to achieve more spatial redundancy. This step does not provide any direct compression but prepares the data for the later steps in the compression process. The RGB to YUV color space conversion provides a 1.5:1 compression ratio. When the video is digitized, the U and V channels of the color information are subsampled at a 2:1 ratio. After the video is digitized, it can be scaled to the output resolution. For example, a 320x240 resolution output from a 640x480 video stream provides a 4:1 compression ratio. The scaled and digitized video frames are transformed into different spatial representations, depending on the compression algorithm used. The transformed data is quantized such that the video data uses a smaller number of bits per pixel of color information. This quantization can provide up to a 3:1 compression

ratio. The final step in intraframe compression is to compact the quantized video data by using a encoding scheme such as RLE, Huffman coding (also called variable-length or entropy coding), or arithmetic coding. This last coding step provides a 1.5:1 compression ratio [Doyle91].

In typical video streams, the content of successive video frames does not change considerably, and as a result interframe compression can supply a generous compression ratio. Three types of interframe compression exist: predictive coding, motion estimation, and picture interpolation. Interframe compression can provide a 5:1 compression ratio, in addition to the more than 50:1 compression ratio delivered by intraframe compression.

Compressed video streams can be described as *constant-rate encoded* or *variable-rate encoded*. Constant-rate encoding schemes accept variable sized video frames but output constant sized compressed frames, while maintaining constant perceptual picture quality. Constant-rate encoding is typically used when the compressed stream is being output into a fixed-rate communications channel. Variable-rate encoding schemes accept variable sized video frames and output variable sized compressed frames, while maintaining constant picture quality [Fogg96].

Some of the most common video data compression standards are MPEG, CCITT H.261, and the *DVI* (Digital Video Interactive) technology with its *RTV* (Real-Time Video) and *PLV* (Production Level Video) compression algorithms. The implementation of a compression algorithm is termed a *codec* (compressor-decompressor). Currently, these compression schemes are only feasible in real-time with hardware support. The compute-

intensive nature of these compression algorithms overwhelms most computer systems. Only with custom hardware do these schemes become worthwhile.

The MPEG standard consists of three related standards: *MPEG-Video*, *MPEG-Audio*, and *MPEG-System*. MPEG-Video provides a standard for compression of digital video signals with a resulting data rate of about 1.5 Mb/sec. MPEG-Audio provides a standard for compression of digital audio signals at 64, 128, and 192 kb/sec per channel. MPEG-System describes the synchronization and multiplexing of multiple compressed audio and video bitstreams. Some of the features of the MPEG standard are random access to video frames, fast-forward and reverse searches, reverse playback, audio/video synchronization, and editability [LeGall91].

The MPEG-I standard specifies a 640x480 video resolution at 1.5 Mb/sec, while the more recent MPEG-II format specifies a 704x480 resolution at rates up to 10 Mb/sec. The MPEG-II standard is aimed at interactive television, while MPEG-I is suited more for CD-ROM distribution. Using MPEG-I, up to 72 minutes of VHS-quality (30 frames/sec, 640x480 resolution) video can be stored on a single CD-ROM.

The MPEG-Video compression algorithm uses block-based motion compensation for the reduction of temporal redundancy. Predictive and interpolative coding methods are used, followed by *Variable-Length Code* (VLC) compression. It also uses a *Discrete Cosine Transformation* (DCT) based compression technique for the reduction of spatial redundancy [LeGall91].

Motion compensation uses prediction and bi-directional interpolation to provide compression. Prediction exploits the temporal redundancy of video signals. This means that each frame can be represented as some transformation of a previously encoded frame. Bi-directional interpolation results in the reproduction of a full temporal resolution stream from a partial temporal resolution stream (1/2 or 1/3 of frame rate). The motion is represented by one or two motion vectors per 16×16 pixel block (*macroblock*). Pixel blocks are matched with blocks in previous and future pictures. If a closely matching block is found, the motion of the macroblock is encoded into a *motion vector*. As used in JPEG image compression, MPEG uses a DCT compression technique for the reduction of spatial redundancy. The DCT is a fast algorithm and results in good visual quality while providing the desired spatial compression [LeGall91].

MPEG-Video compression results in three types of encoded video frames: intrapictures (*I-frames*), predicted pictures (*P-frames*), and bi-directionally predicted pictures (*B-frames*). I-frames provide reference points for random access, while only supplying moderate compression. P-frames are coded with reference to past I-frames or P-frames, and are used as reference frames for predicted pictures. B-frames provide the highest amount of compression and require past and future reference frames for prediction. B-frames are not used as reference frames for other predicted frames [LeGall91].

The Motion-JPEG (*M-JPEG*) format uses intraframe compression, in the form of JPEG compression, for each frame in a video sequence. Every frame is a reference frame in this format. Because of this, M-JPEG is commonly used in non-linear video editing since every frame can be used as an starting or ending point for edits.

The H.261 video coding algorithm provides a compression solution at rates of p×64 kb/sec (p = 1, 2, ..., 30). These rates cover the ISDN channel capacity. This algorithm is designed for video-phone and video-conferencing applications for which a real-time algorithm with minimum delay is a requirement. For values of p = 1 or 2, this algorithm can support desktop face-to-face visual communication (video-phone), and for values of p ≥ 6, true video-conferencing is possible [Liou91].

H.261 supports the CIF and QCIF video resolutions. At 15 frames/sec, the algorithm provides a compressed data rate of 320 kb/sec for CIF resolution video, and 64 kb/sec for QCIF resolution video. The compressed data rate of CIF resolution video at 30 frames/sec is 1.472 Mb/sec and at 10 frames/sec is 64 kb/sec [Liou91].

Both interframe and intraframe encoding are used by the H.261 algorithm. It uses a hybrid of DCT and *DPCM* (Delta Pulse Code Modulation) coding schemes with motion estimation. Each frame is split into a luminance channel and two chrominance channels and each of these channels is split into 8x8 pixel macroblocks. The motion estimation is done by comparing every luminance macroblock in the current picture with the nearest luminance macroblocks in the previous picture. If the difference between any macroblocks in the current and previous picture is greater than a predefined threshold, the difference is processed and stored with the calculated motion vector information [Liou91].

Intel's DVI technology supports both *symmetric* and *asymmetric* compression schemes. In a symmetric compression scheme, the time to compress and decompress a video segment are similar. While with an asymmetric compression scheme, the time to compress

a video segment is considerably greater than the time to decompress the video segment. Asymmetric compression schemes typically do not support real-time compression. The compression will take place off-line after the video stream has been digitized.

PLV is an asymmetric compression scheme which requires the analog source video to be sent to a central digitizing and compression facility which uses parallel supercomputers for compressing the video. The resulting video can be played back in real-time on any DVI-capable computer system. PLV supplies a much higher video quality than its similar, although symmetric, compression scheme, RTV. RTV can play back and compress video in real-time with hardware support [Luther89].

With PLV compression, the analog source video is played at a full frame rate to minimize frame storage requirements and possible quality loss. The video is digitized, filtered and color sampled before being stored digitally. This preprocessed video requires about 2 Mbytes/sec of storage space. In non-real time, the preprocessed data is compressed frame-by-frame on a parallel processing transputer-based computer. In 1989, compression took about 3 seconds per frame, or 90 minutes of compression time for one second of digitized video. The PLV compressed video uses a picture interpolation method where the difference between successive frames (*delta frames*) is compressed relative to *reference frames* [Luther89].

RTV compression provides a method by which an application developer can compress his own source video in real-time without the delay or expense of PLV compression. RTV can compress video frames to the same size as PLV compression, but sacrifices picture

quality for speed. RTV compression is done with a lower resolution and frame rate, and with a simplified video compression algorithm. This compression scheme does allow the user to adjust the balance between picture quality and compression ratio [Luther89].

There are also many software-only video codecs available. These compress video data after it has been digitized by a video capture device. Some can process the incoming video data in real-time and others require the data to be pre-stored in a raw digital format on the user's computer. Some of these compression schemes can also make use of available compression hardware to provide better results. Table 8 describes a selection of the available software-only video codecs.

Name	Developer	Type	Platform	Maximum frame rate
Captain Crunch	MediaVision	Wavelet	PC	320x240 @ 30 fps
Cinepak	SuperMac	VQ	Mac, PC	320x240 @ 15 fps
DVI-RTV	Intel	VQ	PC	245x240 @ 15 fps
DVI-PLV	Intel	VQ	PC	640x480 @ 30 fps
Indeo	Intel	VQ	Mac, PC	320x240 @ 15 fps
Motion-JPEG	N/A	DCT	Mac, PC	640x480 @ 60 fps
MotiVE	MediaVision	VQ	PC	160x120 @ 12 fps
MPEG-1	N/A	DCT	PC	320x240 @ 15 fps
MPEG-2	N/A	DCT	PC	704x480 @ 60 fps
H.261 (p×64)	N/A	DCT	Mac, PC	352x288 @ 15 fps
Pro-Frac	TMM	Fractal	PC (DOS)	320x240 @ 30 fps
SoftVideo	TMM	RLE	PC	640x480 @ 15 fps
Ultimotion	IBM	N/A	PC (OS/2)	320x240 @ 30 fps
Video	Apple	VQ	Mac	160x120 @ 15 fps
VideoCube	ImMIX/Aware	Wavelet	Proprietary	640x480 @ 60 fps

Table 8: Software-only video compression algorithms [Doyle93]

Two of the most often used software-only codecs are Intel Indeo and SuperMac Cinepak. The Indeo compression scheme uses color subsampling, pixel differencing, vector

quantization, and run-length encoding. The codec is scaleable by providing higher frame rates, but uses a fixed video resolution (160x120 or 320x240). Indeo is supported by Apple QuickTime and Microsoft Video for Windows, and it provides a data rate between 90 and 300 kB/sec. It can also use the Intel i750 processor for hardware-assisted decompression when the Intel/IBM ActionMedia II board is employed. With the ActionMedia II board or the Intel Smart Video Recorder, this codec can be used to capture and compress in real-time [Perey94].

SuperMac's Cinepak codec uses a *vector quantization* (VQ)-based algorithm to provide data rates similar to that of the Indeo codec. It supports video resolutions of 160x120 or 320x240 at 24 bits/pixel, while delivering a data rate between 90 and 300 kB/sec. When used with Creative Labs Video Spigot board, video can be captured and compressed in real-time. This is a highly asymmetric algorithm. In other words, if this algorithm is executed without hardware support, the compression time will greatly exceed the decompression and playback time. As with Indeo, Cinepak is found in Apple QuickTime and Microsoft Video for Windows [Perey94].

Two other interesting software-only codecs are Iterated Systems/TMM Softvideo and MediaVision Captain Crunch. Softvideo is a fractal-based compression scheme which provides a scaleable window size (from 320x200 to 800x600) at data rates ranging from 30 to 120 kB/sec. The asymmetric nature of Softvideo exceeds all other codecs, requiring 15 hours of compression time for every 1 minute of source video.

Captain Crunch provides high-quality compressed video at low data rates by using a combination of *DWT* (Discrete Wavelet Transformation), tree-based encoder, vector quantization, and a Huffman encoder. It is a nearly symmetrical compression scheme that supports scaleable frame rate, window size and bit depth.

Various hardware exists for assisting video compression. The individual steps of the compression algorithms, such as motion estimation, quantizing, and DCT calculation, have silicon-based equivalents. The SGS-Thomson STI3220 is a motion estimation processor, and the IMS A121 is an 8x8 pixel block DCT processor [Kim91]. For JPEG image compression, C-Cube Microsystems has the CL550 and CL560 chips available. The CL550 contains a DCT/Inverse-DCT processor, quantizer, and VLC encoder. It supports JPEG images at a 640x240 resolution, and interpolates vertically to provide a 640x480 resolution. The CL560 chip provides Motion-JPEG video compression at a 640x480 resolution. For MPEG compression, C-Cube has the CL450 which supports MPEG-I standard video compression [Brown93].

The Intel/IBM ActionMedia II board and the Intel Smart Video Recorder both use the Intel i750 chip set for video compression and playback. The i750 chip set contains a pixel processor and separate display processor. The 82750PB pixel processor supports compression, decompression and VLC decoding, while the 82750DB display processor is responsible for timing signals and YUV to RGB conversion. The chips work in an 8 bit YUV color space where the chrominance is subsampled at 1/2 or 1/4 the luminance resolution in both the U and V directions. The i750 chip set is programmable and can be

programmed to decode JPEG images and compressed motion video. JPEG images at a 640x480 resolution can be decoded in less than one second [Harney91].

3. CONTENT DELIVERY

When a multimedia application requests the playback of video or audio data, the request is satisfied by the application's host computer platform. The operating system of the computer manages the transfer of data from the source to the display destination. The source of the data can be on the local machine or on a remote machine connected by a network. The destination of the data is generally the video display and/or the audio playback device on the local machine. The action of transferring the data involves the CPU of the destination machine, the storage device (i.e. magnetic disk or CD-ROM) at the data source, and possibly the network interconnection devices at both the source and destination.

The media that is delivered from the source machine to the destination machine can be either static or dynamic. Static media, such as text or images, have no time element, while dynamic media, such as audio or video, do possess a time element. Normally static media is delivered in one contiguous chunk, but it is also possible to progressively render static media. Progressive rendering of images means that over time the image is displayed in several "progressions." The image is stored as multiple resolutions of increasing quality. These resolutions are incremental such that the later resolutions build on the earlier resolutions to produce an image of increasing quality as with interlaced GIF and progressive JPEG images.

The individual chunks of dynamic media data are not normally rendered progressively. The granularity of video playback is typically limited to the video frame. Some video

decompression schemes use a sub-frame granularity where pixel blocks of the frame are decompressed independently and composited into the video frame.

Another factor in delivering media is the delivery mode: real-time or non-real-time. The real-time delivery of a video segment means that the user begins watching the video as soon as it is sent from the source. Audio can be played back in a similar manner. Examples of real-time delivery are video conferencing (CU-SeeMe) [Cogger94], or networked audio playback (RealAudio Player) [Progressive95]. In non-real-time delivery mode, however, the media is fully delivered to the destination before it is played back. An example of this is an audio file embedded in an e-mail message. The e-mail message must be fully delivered before the audio file can be separated from the original message and played back.

The delivery of media is an asynchronous exchange of data between the source and destination machines. There are normally two parallel flows of information involved in media delivery: the data flow and the control flow. The data flow is a single duplex channel over which the media data travels, while the control flow is a full-duplex communication channel over which the source and destination sites negotiate the characteristics of the transmission of the data. In real-time delivery mode, such as with multimedia conferencing, there are real-time constraints which the data flow must meet. If a video frame can not arrive at the specified time, the constraints will not be met and some delivery parameters must be adjusted. Possible parameters for adjustment are the frame rate of the video stream, the visual quality of the video stream, and the buffering ability of the source and destination sites [Ahuja92] [Uppaluru92].

3.1 Using Multimedia Data Delivery in Applications

Different applications use different methods of data delivery. For example, multimedia authoring applications and video-conferencing applications use multimedia data in distinctly different ways. A multimedia authoring application will play back audio and video data from a storage device to the video display and audio playback devices. A video-conferencing application generates compressed video data at the source and transfers the data over a network to the destination site. At the destination site, the compressed data is immediately decompressed and played back, or it can be archived to a local storage device. The decompression is normally done by custom hardware but new algorithms and faster computer processors allow it to be done in software. Most audio playback devices have decompression hardware, and video decompression hardware is becoming a common feature on video display devices.

A multimedia application must make several design choices: whether to have single or multiple data delivery channels, delivery over a network or direct from disk, data used in a design or runtime mode, and data delivered in a continuous mode or as individual chunks. Whether to synchronize between various media data elements is another choice which must be made.

A single application can open multiple concurrent data channels, for example audio data streaming from a CD-Audio device, video data streaming from a magnetic storage device, and textual data streaming off the Internet. A typical multimedia application will use multiple concurrent data channels to assemble the multimedia presentation.

Multimedia authoring and presentation applications commonly have a design mode and a runtime mode. The data access methods between these two modes can differ greatly. At design-time, data is normally accessed in a random-access method while the application developer searches for the proper segment of multimedia data. He may choose to edit or pre-process the data at this time. At run-time, the same selection of multimedia data would normally be accessed in a sequential manner and be played back from start to finish.

The chunks of information accessed by an application can be termed *presentation objects* or *P-objects*. These P-objects can be classified according to their size, media composition, and links to other P-objects. For optimal networked data delivery, the application must provide the network with information that indicates the P-object's current bandwidth allocation demands and its dependence on time [Loeb92].

P-objects can be thought of as the individual media elements of the multimedia documents which multimedia applications create, edit, and display. Each P-object has its own unique display characteristics (size, bandwidth demands, etc.) and the multimedia application must take these characteristics into consideration for the optimal display of the P-object.

Similarly, by using a technique called *application-level framing*, an application can specify its own transmission units (i.e. packet size) which can be termed *application data units* or *ADUs*. In networked data delivery simulations, it has been shown that a significant source of delays is a fixed packet size, which is not optimal for the various data types within an application [Loeb92].

The amount of text to be delivered may vary with the size of the video display device. In addition, the packet size of a video stream will be determined by the size of a compressed video frame, and bitmapped images may be delivered a scanline at a time.

For interactive multimedia applications, the non-sequential access of media data defeats typical caching and pre-fetching mechanisms. One solution is to have the application dictate the data delivery demands in accordance with the data access method.

A multimedia application can specify that a selection of media data will be used only once or that it will be used repeatedly to provide hints to the caching mechanism. It can also specify that the data will be used either sequentially or randomly in order to provide a hint to the pre-fetching mechanism (see Table 9).

Caching	
Normal	unknown page access order, caches needed pages upon access
Random	pages are likely to be accessed in random order, caches minimum amount of pages
Sequential	pages are likely to be accessed once and in order, caches maximum amount of pages
Pre-fetching	
Normal	read pages on demand
Will need	read pages in immediately
Do not need	free pages immediately

Table 9: Types of application data delivery advisement [Loeb92]

It is preferable to pre-fetch data when the data delivery channel is lightly loaded. Basically, this is an example of overlapping the data delivery with any application-level delays. The time during which an application is loading can be used for pre-fetching data that is expected to be used. For an authored title which uses linear navigation, data for

subsequent sections of the title can be pre-fetched because it is likely the data will be needed at a later time.

Other heuristics can also be applied to optimize the pre-fetching operation. At design-time, the user will most likely be cycling through various pieces of media data trying to find the appropriate one, editing and pre-processing the data, and linking it into a multimedia document. The data access method will likely be random, therefore no pre-fetching should be used and only the minimal set of accessed data should be cached as there will be a limited locality of reference. The data also may be accessed in both read and write modes while editing.

At run-time, the multimedia document is already linked to the data and can be pre-fetched if desired. In most cases, the data will be read sequentially and should be cached as such. Depending on the type of navigation available, other linked media data may be pre-fetched to optimize the latency for starting playback. Also, the data will most likely be accessed in read mode only, since no run-time editing will occur.

3.2 Platform Issues

Many multimedia applications are designed to deliver multimedia data, but not every application needs to access the low-level delivery mechanisms directly. For this reason, system-level support for multimedia data delivery is found in many operating systems or in multimedia extensions placed on top of operating systems.

3.2.1 Apple QuickTime

The Apple Macintosh operating system uses *QuickTime* multimedia extensions for multimedia support. QuickTime manages the delivery of various classes of multimedia data: digital audio and video, MIDI audio, or SMPTE timecodes. The basic data object in QuickTime is the "movie" and it consists of one or more "tracks." Each track contains references to chunks of raw data, called "media" [Hoffert92].

QuickTime consists of the *Movie Manager*, the *Image Compression Manager* (ICM), and the *Component Manager*. The Movie Manager is responsible for the synchronization of the media data and delivery of the data from the storage device. The ICM manages the various compression algorithms and is used to compress and decompress media data. The Component Manager provides a hardware abstraction layer to determine hardware resource availability. QuickTime is not tied to any specific hardware devices and is extensible as new hardware and new compression algorithms become available [Wayner91].

3.2.2 Intel Audio Video Kernel

Intel's *Audio-Video Kernel* (AVK) provides multimedia services to IBM OS/2 and Microsoft Windows operating systems [Donovan91]. It is tied closely to the Intel/IBM ActionMedia II video capture and display device, and uses a real-time kernel to manage the multimedia data delivery. The AVK uses a production studio model with separate managers for analog device control, visual display, digital audio and video playback, and digital effects and mixing. The image data flow pipeline, for example, loads compressed data input from a storage device into a compressed data buffer, decompresses the data

into a separate decompressed data buffer, and renders the decompressed data to the screen. Each step in the pipeline is managed by a separate real-time task [Donovan91].

3.2.3 UC Berkeley Comet

Comet, from the University of California at Berkeley, is a multi-user multimedia toolkit which provides an abstraction to hardware and network resources, and provides an architecture to connect media data sources to sinks in a dataflow-style graph [Anderson91b]. Comet is an object-oriented toolkit which abstracts devices, storage devices, mixers, and the network interconnections. The Comet objects support a protocol by which data types and resources are negotiated before connection. Comet runs on UNIX workstations with X-Windows and a connection to an ACME continuous media server (see [Anderson91]). Multiple processes are used to manage the delivery of media data from source to sink and to filter the data during transmission (i.e. audio and video mixing) [Anderson91b].

3.3 Synchronization Issues

Synchronization has been defined as the "ordering of processes and their coordinated access to shared resources." Each process can manage an individual data stream or a set of data streams. It is possible to define a global synchronization system which imposes synchronization on multiple data streams across multiple processes. This synchronization system can exist in the local or distributed environment, i.e. it can manage media on one computer or media distributed across many interconnected computers [Steinmetz90].

When a multimedia application begins to use multiple streams of media data, the problem of synchronization arises. There are several solutions to this problem which have been implemented, but the solutions are usually application-specific. Synchronization can be handled in the operating system or in an application-specific manner.

When developing a multimedia document, the multimedia objects must be assembled both in space and in time. This can be termed *spatial composition* and *temporal composition.* Temporal composition can be *continuous* or *synthetic*. With continuous synchronization, absolute synchronization exists only at the beginning and end of data stream playback. The source and destination of a data stream will differ in synchronization during playback.

Synthetic synchronization employs a coarse synchronization technique with object-level granularity. Data streams can be started or ended based on user interaction, application execution or termination, or pre-defined relationships. Objects can have sequential or parallel time relationships. When these multimedia objects are presented, the synthetic relationship, minimum and maximum delay, and the desired performance constraints can be taken into consideration for optimal playback performance [Little90].

A coarse-grain synchronization technique would define the interconnection of objects in a multimedia document, while a fine-grain technique would illustrate the binding of objects to a reference timeline for playback timing [Little90a].

Complex temporal relationships can be created between multimedia objects. Some examples of object relationships can be seen in Table 10.

Relationship	Description
Pa before Pb	[Pa] <delay> [Pb]
Pa meets Pb	[Pa] [Pb]
Pa overlaps Pb	[Pa] <delay> [Pb]
Pa during Pb	<delay> [Pa] [Pb]
Pa starts Pb	[Pa] [Pb]
Pa finishes Pb	<delay> [Pa] [Pb]
Pa equals Pb	[Pa] [Pb]

Table 10: Examples of temporal object relationships [Little90a]

3.3.1 Continuous Synchronization

The *ACME* (Abstractions for Continuous Media) server provides network-transparent access to hardware devices, manages multiple concurrent data streams, and allows clients to specify synchronization requirements while providing mechanisms to enforce them. Each client application can request multiple data streams (*ropes*) which consist of individual per-device streams (*strands*). These strands are multiplexed into ropes and are transmitted over the client-server network connection. The ACME system uses a *logical time system* (LTS) for synchronization support. An LTS can have a variable unit of time and can be either device driven or connection driven [Anderson93].

The system can use a skip-pause LTS, which skips or pauses the transmission depending on transmission latency. Another alternative is the low-delay LTS, which acts to minimize the transmission delay on its streams. A multimedia document browser would use a skip-pause LTS to manage the synchronization of multiple data streams [Anderson91]. The LTS would be driven from a single device, normally the audio playback device, so that all

data streams (such as video and text captioning) would be synchronized to the audio stream. An audio conferencing system would use a low-delay LTS to support low end-to-end delay between clients. The LTS would be driven by the network connection.

An LTS has three modes of operation - startup, normal, and starvation/overrun. During the startup period, the LTS waits for the device and connection buffers to fill up before proceeding. In normal operation, the LTS handles rate mismatches and manages the intrastream synchronization. When buffer starvation or overrun occurs, the LTS may temporarily stop to let the device or connection catch up [Anderson91].

When the LTS must wait for a device or connection to catch up, a data stream may stop playback. Normally when this happens no media data is shown or heard since playback has ceased. A technique called *restricted blocking* can be used to provide some continuity when playback is delayed. For example, video streams can leave the last frame visible during delays in playback rather than showing no video signal at all [Steinmetz90].

3.3.2 Synthetic Synchronization

Explicit synchronization control can be applied by the use of scripts. Scripts can be created to define the set of data streams, the sequence of data units (samples, frames, etc.) that will be played, and the time at which the streams will begin and end. In this case, the script playback engine itself would use an LTS for synchronization which will be driven by the master data stream, as specified in the script [Rowe92].

Script-based synchronization is specified by the application developer and defines the relationship between the media data in the application. For multimedia presentations, the

presentation timing is dependent on the availability of required media data, latency of playback, and the playback algorithm. A scheduling policy may also be put in place to describe which playback deadlines to meet and which playback tasks to defer or drop [Staehli93].

In addition to script-based synchronization, an application can support event-based synchronization. An example of this is the display of subtitles triggered by video playback events (i.e. playback of a specific frame) [Blair91].

Synchronization protocols can be used to manage the synchronized data delivery of multimedia objects with inter-object relationships. An *application synchronization protocol* (ASP) or *network synchronization protocol* (NSP) may be utilized. An NSP allows the playback of complex multimedia presentations from distributed sources to a single site, while an ASP manages the playback to and from a single site [Little91].

An example of a synchronization protocol consists of these steps. First, the temporal relationships between objects in the presentation are retrieved. Next, the relationships are evaluated and a playback schedule which incorporates the differing playback requirements of the objects is created. Then, the overall playback schedule is determined in coordination with an NSP. Finally, the synchronous data transfer of the presentation occurs [Little91].

3.4 Networking Issues

A common concern of multimedia developers involves delivering media data over a network. Networked video may be used in businesses for user training and support, sales

videos, or employee information. Advertising agencies frequently use archives of television commercials [Tobagi92]. The storage resource and delivery mechanism of video data is termed the *video server*. Through the use of a video server, numerous streams of video data can be served concurrently with random access capability.

Video servers must maintain a continuous high data rate with real-time processing requirements. One type of video server may use a *session manager* to service multiple clients. Each *session* can be described by two concurrent client/server streams, or *channels*: command and data. The command channel is used for a bi-directional protocol for managing the data transfer rate. The server monitors the client's buffering ability and can adjust its data transfer rate accordingly by notifications sent on the command channel [Uppaluru92].

The command channel can be implemented by a reliable stream-based protocol (e.g. TCP/IP). The data channel can be implemented by a stream-based protocol or a packet-based protocol (e.g. UDP/IP). Reliable stream-based protocols will retry lost packets during data transmission. The latency for packet retransmission is not acceptable for continuous media data transfer. As a result, specialized packet-based protocols have been invented. In these protocols, no acknowledgment or retransmission techniques are employed. Data packet loss is acceptable and must be dealt with by the application.

Both static and dynamic media types can be delivered over a network. The network may be a Local-Area Network or a Wide-Area Network, such as the Internet. The mode of delivery may be real-time, where the user watches the data stream as it is delivered, or

non-real-time, where the data stream is archived at the user's site and watched later [Blackketter92]. Real-time delivery over WANs is where most synchronization problems occur because of the inherent delays in data transmission.

Network types vary widely in delivery bit-rates (see Table 11), but delays in network routing as a result of network load can diminish the data rates drastically.

Interface Type	Bit-rate
POTS	0.3 - 56 kb/sec
DS-O (used by telephone companies)	56 kb/sec
ISDN	64 - 144 kb/sec
LocalTalk	230 kb/sec
T-1	1.5 Mb/sec
Ethernet	10 Mb/sec
T-3	45 Mb/sec
Fast Ethernet	100 Mb/sec
FDDI (Fiber Distributed Data Interface)	100 - 200 Mb/sec
SONET (Synchronous Optical Network)	N × 51.84 Mb/sec

Table 11: Network interface types and corresponding bit-rates [Liebhold91]

Clients can use the video data in applications, but may need custom networking protocols for streaming data delivery [Tobagi92]. These protocols can provide a scaleable data transfer rate by responding to the network load and application requirements. The data streams may be scaled temporally or spatially to maintain acceptable rates. Temporal scaleability dynamically varies the frame rate of the video stream, while spatial scaleability provides the ability to reconstruct a frame from partial information [Uppaluru92].

A data stream is transmitted from a source to a destination site and will contain a sequence of media objects - video, audio, and so on. The current logical time of the source or destination of the data stream is termed the *presentation time* [Little90]. The presentation

time may be described in various units of time (samples, frames, etc.). For example, the source of the data stream may have transmitted frame 105, but the destination has only received frame 101. A loss of synchronization means that the presentation times at the source and destination of the data stream are different.

The instantaneous difference of presentation times is termed *jitter*. The average delay over the duration of the data stream playback is termed *skew*. The set of performance parameters for data stream playback - including skew and jitter - can be termed *quality of service* (QoS). Object presentation can be constrained by the QoS specification, which could affect the playback data rates, buffering parameters, and network transmission policy [Little90].

Data with a real-time delivery requirement that is transmitted in a single direction can incur large end-to-end delays from data delivery to arrival. However, the arrival times of data packets and jitter need to be constrained. Bi-directional real-time data delivery requires constraints both on jitter and total round-trip delay. As a solution, ATM networks control delay and jitter by using fixed-size data packets. This minimizes buffer sizes, and eases routing delays during data transmission [Hopper90].

3.4.1 Continuous Media Player

The *Continuous Media Player* (CMPlayer) from UC Berkeley supports a continuous media support library and portable user interface. CMPlayer takes as input a script describing a series of audio or video streams, and the start and end frame of each stream. Each script has a logical time system (LTS) for driving synchronization [Rowe92].

CMPlayer is composed of one or more CMSources and a CMServer. Each CMSource resides on a video file server and sends continuous media data in packets over a UDP network connection. The CMServer is an event-driven process which manages a time-ordered playback queue for synchronizing the playback of audio and video packets. It receives packets from CMSources, assembles the CM data from the packets, and queues the data for playback. Control information is sent between the CMSource and the CMServer over a TCP network connection. CMServer requests retransmissions if a packet is lost, and deals with out-of-order data [Rowe92] [Rowe94a].

The CMSource sends data to the CMServer at the appropriate time, and the data is marked with a valid time interval for display. But both the CMSource and CMServer can skip or drop frames to maintain synchronization with the LTS. A timer event notifies the CMSource to send a frame. It then decides which frame to send and calculates the start and end times for display. If the CMServer decides a frame can be decoded and displayed before its logical time, the frame is put on a queue of display requests. If not, the CMServer drops the frame [Rowe94a].

The CM network protocol uses an adaptive feedback algorithm to match packet flow to available resources. CMServer uses a penalty method to maintain a constant playback rate. Frames that are queued but not played, multiple missed frames, and frames lost in the network increase the penalty. This penalty is sent to the CMSource which is transmitting the stream. Every 300 msec, the playback frame rate is calculated based on the current frame rate, the penalty, and the frame rate constraints on the stream [Rowe92].

3.4.2 Rate-based Flow Control Protocol

Researchers at Lancaster University (UK) have proposed a system which uses a relaxed rate-based flow control protocol. This protocol does not support retransmissions when transferring real-time data. The system is composed of two source processes - writer and send - and two destination processes - receive and reader. The writer process gathers media samples into the required buffer size, also called the *burst size*. When the buffers have been filled, the send process fragments the buffer into network packets and adds the appropriate packet headers. Upon packet receipt at the destination, the receive process strips the header information and fills data buffers. The reader process passes the filled buffers to the media playback engine [Shepherd91].

As the first step in the flow-control protocol, the source and destination must agree on the burst size. Next, they must agree on a suitable *burst interval* which allows the receiver to consume and process a negotiated full burst in that interval. When a burst is consumed and processed by the destination, it informs the source of its current buffering capabilities. The source uses this information to decide whether to send the next burst or block at the next burst interval. By increasing or decreasing the burst size, this protocol allows delay jitter to be smoothed out. Synchronization is therefore provided at the buffer level not the packet level [Shepherd91].

3.4.3 Multimedia Virtual Circuit

An alternative delivery mechanism developed at Boston University is the *Multimedia Virtual Circuit* (MVC). An MVC contains multiple channels of synchronized media which are multiplexed onto a single *virtual circuit* (VC) with variable bandwidth. Packet order

is guaranteed by the VC, and there is no connection overhead once the VC is established. The client application requests the number of channels and the QoS per channel, but channels can be added to the MVC during transmission. An MVC only supplies point-to-point connections [Little90].

3.4.4 Continuous Media Transport Service and Protocol

In order to provide better service for continuous media applications, researchers at UC Berkeley have developed the *Continuous Media Transport Service* (CMTS) and *Continuous Media Transport Protocol* (CMTP). CMTS is designed primarily for CM clients, but can coexist with message-oriented clients. It provides a better traffic model for characterizing CM traffic and for specifying performance requirements. CMTS defines an abstraction of logical streams, provides CM-specific error handling, and eliminates the need for the client's acknowledgment of each data transmission [Wolfinger91].

CMTP is responsible for unidirectional communication between CMTS processes at the sender and receiver. Real-time applications cannot wait for retransmissions of data and will handle lost packets, but in cases of serious error, the connection can be torn down and reestablished with a new stream [Wolfinger91].

The conversation between sender and receiver consists of a sequence of logical streams with intervals of silence between the streams. The sending client must notify its CMTS at the beginning and end of each logical stream. The sending client and its CMTS share a circular buffer. The sending client must place all outgoing packets in the shared buffer before the end of the transmission period. This buffer may be prefilled by the client by a

predetermined number of bytes (called *slack*). The traffic and performance parameters of the stream can be redefined at the start of each logical stream, but only a decrease in data rate is allowed [Wolfinger91].

The CMTS at the sender and receiver coordinate to transfer all buffered data packets for each transmission period. The receiving CMTS must inform its client of the beginning of a new logical stream. The receiving client and its CMTS also share a circular buffer. Similar to the sending process, the receiving CMTS must place incoming packets into the shared buffer before the beginning of the playback period in which they will be needed. The receiving client is responsible for removing these packets from the shared buffer before the end of that period, but can fall behind by the slack number of bytes. The receiving buffers are relatively large to smooth out the fluctuations in the arrival of data packets caused by delay jitter [Wolfinger91].

3.4.5 AudioFile

Digital Equipment Corporation has developed *AudioFile*, a network-transparent system for distributed audio applications. AudioFile allows multiple simultaneous clients, supports a variety of audio hardware, supports multiple audio data types and sample rates, and permits transparent access through the network. It can be used for applications such as audio recording and playback, answering machines, voice mail, telephone control, and speech synthesis and recognition [Levergood93].

AudioFile operates on blocks of audio data rather than streams. It does not provide a complete synchronization protocol, but does supply low-level timing information.

AudioFile clients manage end-to-end delay and multiple synchronization clocks. The AudioFile server consists of *device independent audio* (DIA), *device dependent audio* (DDA), and operating system components. The DIA component uses a control loop which waits for client connections and open devices. Client requests are processed by a dispatcher. The DDA component manages the play and record buffers for each device while audio device I/O is performed by reading and writing shared memory buffers [Levergood93].

The AudioFile protocol is modeled after the X-Windows protocol and uses a transport protocol which is reliable and does not reorder or duplicate data. The reliable protocol (TCP) was found to be sufficient as the delay caused by the retransmission of lost packets was small compared to the buffering. Teleconferencing applications over intercontinental TCP links suffered poor performance, and the researchers decided TCP was the wrong protocol for low-delay applications which need guarantees on bandwidth and latency and can accept lost packets [Levergood93].

3.4.6 Etherphone

The *Etherphone* multimedia conferencing system was developed by researchers at Xerox PARC. The *Phoenix* subsystem manages audio conferencing and supports multiple simultaneous conversations in various configurations: one-to-many, many-to-one, and many-to-many. Also, Phoenix supports best-effort conferencing, in which the conversation participants use a media format supported by every participant [Vin91].

Phoenix uses a multicast packet protocol for audio transmission. The audio transmission protocol exhibits a small end-to-end delay (< 50 ms). Data transmission alternates "talk spurts," or segments of continuous speech, with silence. Each audio packet consists of up to 160 μ-Law encoded bytes, which is equivalent to 20 ms of audio at 8000 bytes/sec. At most, 50 packets/sec are transmitted per conference participant. An energy value is computed for each packet. If this value is below a predetermined energy threshold, the packet is considered silent and is not transmitted. To smooth the transition from speech to silence and from silence to speech, low volume packets are transmitted at the end of the preceding talk spurt and at the start of the following talk spurt [Vin91]. For this to be possible, the audio transmission must buffer enough data to be able to calculate the silent portions of the stream. Otherwise, the silent packets at the transition points could not be replaced with low volume packets since they would already have been sent.

3.4.7 X-MOVIE

Researchers at the University of Mannheim have produced a system for transmitting and displaying digital movies called *X-MOVIE*. X-MOVIE consists of the Movie Server, the Movie Client, and the X-Client. The Movie Server maintains a directory of movies available for playback. Also, it accepts playback requests from, and delivers the movie over the network to, the Movie Client using an application-level protocol named *Movie Transmission Protocol* (MTP). The Movie Client extends the standard X-Windows System X-Server to provide digital movie playback. Similarly, the X-Client extends the standard X-Client to provide this same functionality [Lamparter91].

The Movie Server provides services to the Movie Client such as Play, Stop, Step Forward/Step Backward, Show Picture *n*, and Slower/Faster. When using the MTP protocol, all data packets from the Movie Client to the Movie Server and all control packets from the Movie Server to the Movie Client are acknowledged. All movie data packets sent from the Movie Server to the Movie Client are unacknowledged in order to provide a continuous data flow. When run over TCP/IP, the data rate from Movie Server to Movie Client was found to be about 150 kb/sec, but when run over UDP/IP, the data rate jumped to approximately 2 Mb/sec with 10% packet loss. TCP was found to be a poor transport protocol for real-time data because of the flow control and packet retransmission protocol mechanisms [Lamparter91].

4. CASE STUDY

The thesis project that was undertaken illustrated an example of networked playback of video and audio streams in a client/server environment. The software application that was developed can act as both client and server. The application can serve multiple concurrent video and audio streams, and also can render multiple incoming video and audio streams. The server plays the audio and video streams off a local storage device such as a hard drive or CD-ROM. It accesses pre-compressed video and audio files and transmits each individual video frame (or buffers of audio samples) over the network. The compressed video data is decompressed on the client and rendered to the local display device. The compressed audio data is sent to the audio device (e.g. sound card) where it is decompressed and played.

A LTS-based synchronization engine was used in conjunction with an adaptive rate control algorithm. *Real-Time Protocol* (RTP) was used to provide real-time *quality of service* (QoS) information to the media server. The application can support the synchronization of multiple independent data streams. The adaptive rate control algorithm uses the QoS information which is sent from the client back to the server in order to optimize the synchronization of the data streams. The server data rate is altered dynamically to provide the highest data rate without sacrificing packet loss.

The goals of the project were to provide synchronized video playback over a network while leveraging existing technology in order to support extensibility, portability, and interoperability. The application was therefore designed to take advantage of increasing

network bandwidth, advances in data compression, as well as advanced operating system support.

Other possible uses for the application are in a multimedia document browser, such as for the World Wide Web, or in a multimedia conferencing application. The server part of the application could be used in conjunction with a Web server to provide for streaming continuous media over the Internet. Alternately, the application could be enhanced to support live audio and/or video capture in order to support live multimedia conferencing.

4.1 Related Works

The LTS-based synchronization engine was derived from the ACME server developed at the University of California at Berkeley [Anderson91] [Anderson93]. The adaptive rate control algorithm was based on the penalty method used in CMPlayer [Rowe92], while the RTP protocol implementation was derived from the Xerox PARC Netvideo application [Frederick93].

These software applications developed as a result of research projects at other universities were used as the groundwork for this project, and the choice was made to start this research project from where the other projects left off. This allowed the code that was written for this application to be unique and limited the duplication of preexisting code.

4.2 Design Rationale

As stated above, a primary goal of the project was to leverage existing technology in order to support extensibility, portability, and interoperability. Extensibility was provided by allowing the application to work with a variety of software codecs. Depending on the

bandwidth available to the client-server network link, a different codec may be used to optimize the quality of the compressed audio and video.

Under Microsoft Windows, the operating system support for video playback and capture is known as *Video For Windows* (VFW). VFW abstracts the specifics of the video compression and decompression algorithms which are used. To play back a frame of compressed video, the frame is handed to the VFW subsystem along with a bitmap header data structure which describes the video frame dimensions, bit depth, and compression algorithm. By using the compression algorithm item from the bitmap header, VFW dynamically loads the code necessary to decompress the video frame. The decompressor is called to decompress the video frame, after which the decompressed frame is rendered to the display device.

One limitation of using VFW is that the standard codecs (e.g. Intel Indeo, Radius Cinepak) only support the compression and decompression of full video frames, not sub-frames as in some video conferencing applications (e.g. CU-SeeMe). The CU-SeeMe compression algorithm encodes sub-frames along with their (x, y) position within the full frame. This allows each full video frame to be built up from many sub-frames and minimizes the effect of a lost network packet (which would contain a sub-frame rather than a full frame).

If the video compression algorithm selected uses interframe compression, and a reference frame of the video stream is dropped during its network transmission, the rendering of the video stream can contain undesirable visual results. The intermediate frames of the video

segment will not have a reference frame to use for decompression. To get around this problem, the video stream should be compressed without interframe compression when transmitting the stream over a network (when using a non-guaranteed network protocol).

The operating system support for audio playback and capture is provided by the *Windows Multimedia* subsystem (WinMM). Similarly to video decompression, buffers of audio data are handed to the audio device along with an audio header data structure. The header contains the audio compression algorithm, the sample rate, and the number of channels (e.g. mono or stereo) in the audio buffers. This application uses a fixed frame rate of 15 fps so that the each audio buffer contains 67 msec worth of audio data.

To provide portability, many standard *application programming interfaces* (APIs) were used in the development of this software application. System-level APIs for the user interface (*MFC:* Microsoft Foundation Classes), multithreading (*Win32*), multimedia services (VFW and WinMM), and network connections (*WinSock:* Windows Sockets Services) were provided by the operating system. By writing to these abstract APIs, the application can run on any hardware platform where these APIs are supported. Both the Windows 95 and Windows NT operating systems support the Win32 system APIs. Third parties have developed Win32 emulation libraries on various Unix implementations. In addition, applications written using MFC for their user interface support can simply be recompiled to work under the Macintosh operating system. Microsoft also provides a Win32 software layer for the Apple Macintosh which allows a subset of the Win32 APIs to exist on Macintosh computers.

Most of the related applications (including ACME and Netvideo) were run on the Unix platform. By writing to the high-level Win32 API, this software application is less dependent on the underlying hardware and, therefore, can transparently make use of advances in audio and video codecs, multiprocessor support, and new display devices. Using the Win32 APIs and MFC for development greatly decreased the implementation time and made the application more extensible and portable.

The networking protocol selected for this project was RTP. It is a non-guaranteed protocol which has support for the real-time transmittal of data. The RTP headers contain timestamp information which allows the receiver of the data to know immediately whether a packet has been lost (or, at least, received out of order). *RTCP* (Real-time Control Protocol) is used for media source management. This protocol is used to signal the beginning and ending of data streams and to send QoS information back to the sender of the data.

As was seen in many other networked video playback systems, a non-guaranteed protocol will exhibit much higher data throughput than a guaranteed protocol (such as TCP). The X-MOVIE system found TCP to be a poor transport protocol for real-time data because of the flow control and packet retransmission protocol mechanisms. Also, the non-guaranteed protocol showed more than a factor of ten improvement in data throughput [Lamparter91]. A non-guaranteed protocol will be susceptible to noticeable packet loss (upwards of 10%) during data transmission and the receiver of the data must be able to handle the data loss without difficulty.

The ACME server used TCP for data transmission and RPC for control information. Because of the documented inadequacies of TCP as a continuous media transport protocol, in this project these protocols were replaced with RTP for data transmission and RTCP for control information. Also, since RTP is an open protocol standard, the Win32 client application can interoperate with any other network server that adheres to the RTP protocol. The only limitation is that the client must be able to decompress and render the encoded data transmitted by the server. The RTP protocol header includes a tag for the format of the data stream which is being transmitted. If the client can not understand the data format being sent to it, it will ignore the connection with the server.

In this project, an adaptive rate control algorithm was used on the server side to alter the transmitted data rate. This algorithm used the QoS information sent from the client back to the server to adjust the rate at which data was transmitted. If too many frames were lost during transmission due to inadequate buffering or other reasons, the server would slow its transmission rate. The algorithm also can sense a surplus in bandwidth availability and will increase the transmission rate if no frames are being lost. The algorithm used is based on the CMPlayer application [Rowe92].

4.3 Implementation

The software application was written in C++ using Microsoft Visual C++ 4.2 and tested on a Gateway 2000 133 MHz Pentium desktop computer and an HP Omnibook 5000CTS 120 MHz Pentium laptop computer both running the Microsoft Windows 95 operating system. The user interface was built on the MFC class library. For system services, including multithreading and synchronization (events, critical sections, etc.), the Win32

API was used. Video for Windows was used for video decompression and image rendering. WinMM was used for audio decompression and playback. WinSock was used for socket creation and data transmittal and receipt.

C++ code from the ACME and Netvideo applications were reused for this application. The implementation of the logical, physical, and compound logical devices and the logical time systems were reused from ACME. The formatting and decoding of the RTP and RTCP headers were reused from Netvideo.

The application can be broken down into the user interface, the networking protocol, and the synchronization engine (the logical, physical, and compound logical devices, and the logical time systems).

4.3.1 User Interface

The user interface is subdivided into three sections: the *client view*, the *client information view*, and the *server information view* (see Figure 5). If the application is acting as a client for media playback, the first two views will be used. If the application is acting as the server for media playback, the third view will be used. The application can function simultaneously as a client and server and in that case all views will be utilized. In this case study, the streams of audio or video data sent from the server to a client are called *media servers*. The streams of audio or video data received by the client from one or more media servers are called *media sources*.

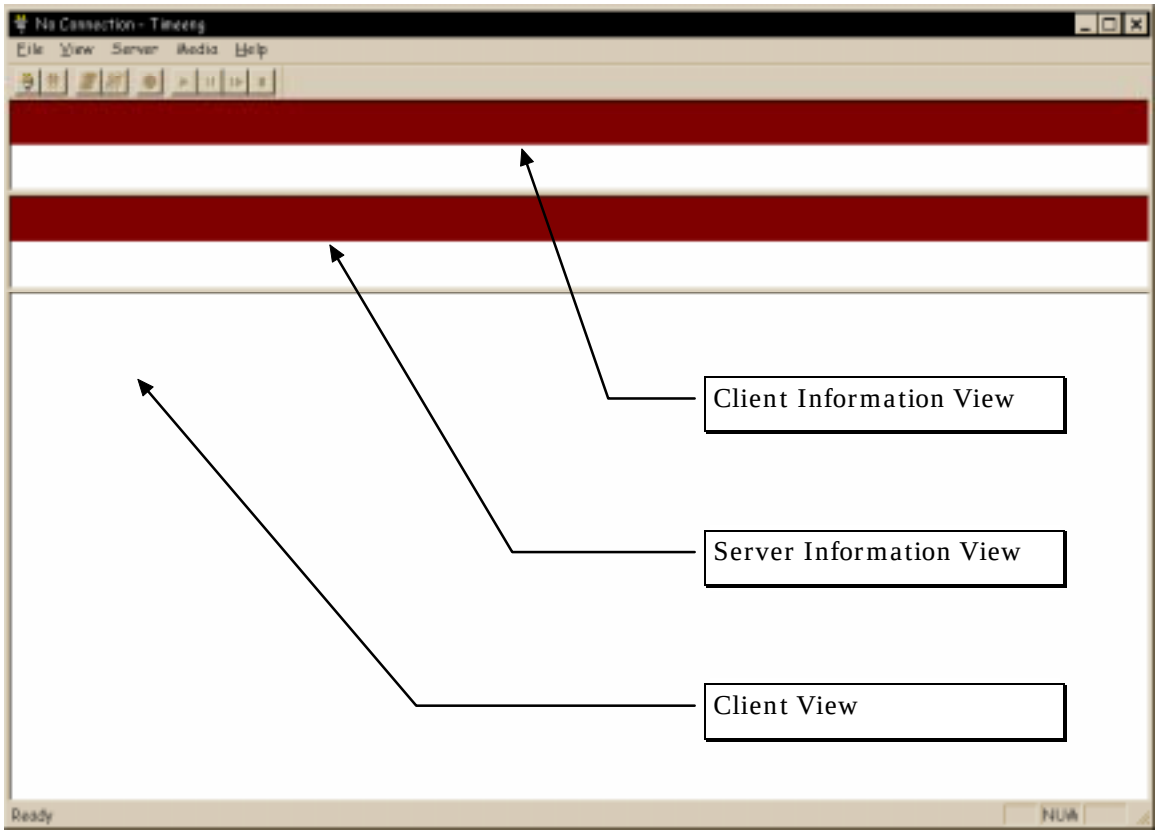

Figure 5: Application window showing client information view, server information view, and client view areas

The client view is used to display the video streams received from the media servers. Audio streams are sent to the audio device (i.e. sound card) as they are received. The client information view displays information about and current statistics for each media source (audio or video) (see Figure 6). For each media source, this information includes the filename of the media source, the name of the machine sending the data, and the name and location of the user sending the data. (The name and location are configurable on the server.) Also, the current data rate (in kilobits per second), packet rate (in packets per second), and average bytes per packet are displayed. For video streams, the number of

frames received is displayed, and for audio streams, the number of samples received is displayed.

Figure 6: Client information view showing information about current media sources

The server information view displays information about and current statistics for each media server (audio or video) (see Figure 7). For each media server, this information includes the filename of the media server. For video streams, it also includes the number of frames in the original media file, the height and width of the video frames, the original frame rate, and the name of the video compression algorithm. In addition, for audio streams, it includes the number of samples in the original media file, the number of samples per second, the number of bits per sample, the number of channels, and the name of the audio compression algorithm. For both audio and video servers, the number of data packets lost during transmission to the client, the current period, the current data rate, and the number of frames (or samples) sent is also displayed.

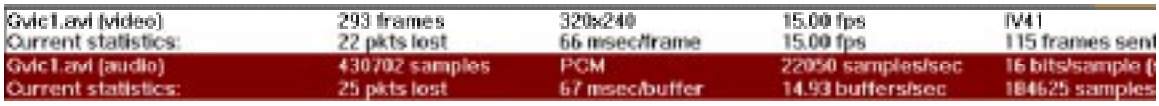

Figure 7: Server information view showing information about current media servers

In order to quickly access the functionality of the application, the most common commands are presented in the application's toolbar (see Figure 8). The toolbar buttons are grayed out when not accessible in the current application context. Their visible state

will change as various application states change. From left to right, the available toolbar commands are:

- Connect to client
- Disconnect from client
- Add media file
- Delete media file
- Change server playback rate
- Play
- Pause
- Restart
- Stop

Figure 8: Application toolbar with buttons for common commands

The complete set of available commands is available through the application's hierarchical menu structure (see Table 12).

Menu Item	Description
File	
Connect to client...	Brings up dialog to select client machine address to set up connection
Disconnect from client	Disconnects from currently connected client
Exit	Exits application
View	
Toolbar	Toggles Toolbar on/off
Status Bar	Toggles Status Bar on/off
Server	
Play	Starts playback
Pause	Pauses playback
Restart	Restarts paused playback
Stop	Stops playback
Change Server Info...	Brings up dialog to enter new server information
Change Playback Rate...	Brings up dialog to enter new output playback rate
Use Rate Control	Turns adaptive rate control algorithm on and off
Use Low Delay LTS	Toggles between using Skip/Pause LTS and Low Delay LTS
Log	Toggles media server logging on/off
Media	
Add Media File...	Brings up dialog to add media file to be played
Delete Media File	Deletes media file selected in the Server Information View
Change Media File Info...	Brings up dialog to enter new text description for media file selected in the Server Information View
Add Video Streams	Toggles adding video streams from AVI files on/off
Add Audio Streams	Toggles adding audio streams from AVI files on/off
Log	Toggles media source logging on/off
Help	
About...	Displays copyright information about application

Table 12: Application hierarchical menu with complete set of available commands

4.3.2 Networking Protocol

The RTP and RTCP networking protocols define a set of headers which make up an RTP packet. There is a standard RTP header, which is always found at the beginning of an

RTP packet, and various RTP option headers which are appended to the standard RTP header.

```
typedef struct {
      UINT rh_vers:2;
      UINT rh_chanid:6;
      UINT rh_opts:1;
      UINT rh_sync:1;
      UINT rh_content:6;
      USHORT rh_seq;
      ULONG rh_ts;
} rtphdr;
```

Figure 9: Standard RTP header

Figure 9 shows the standard RTP header. *rh_vers* is the RTP version ID (which equals 1 in this implementation). *rh_chanid* is the channel ID for the data packet which can be used to differentiate multiple conversations between client and server. *rh_opts* is a flag bit to signal whether optional headers exist after the standard RTP header. *rh_sync* is a flag bit which signals the end of a series of one or more sub-packets which are sent with the same sequence number. (In this implementation, the rh_sync bit is always set since complete video frame and audio buffers are contained in each RTP data packet.) This flag can be used when decomposing large packets into smaller sub-packets which are more likely to not be dropped during data transmission. This is a primary area for future work with this project. When high quality video codecs are used, packets containing video data can get rather large (e.g. greater than 16 kB). As a result, these packets have a higher probability of being lost in the network.

rh_content defines the type of data found in the RTP packet. In this implementation, this value will be either RTPCONT_AVI (with value of 32) for video data or RTPCONT_WAVE (with value of 33) for audio data. *rh_seq* identifies a packet within a

data stream. This number is incremented after each packet is sent. Each packet is timestamped with a monotonically increasing 32-bit value in *Network Time Protocol* (NTP) format. This value is stored in the *rh_ts* entry.

After the standard RTP header can come one or more RTP option headers. Each of these will begin with the basic RTP option header seen in Figure 10.

```
typedef struct {
      UINT roh_fin:1;
      UINT roh_type:7;
      UCHAR roh_optlen;
} rtpopthdr;
```

Figure 10: Basic RTP option header

This header contains a flag bit, *roh_fin*, to signal whether this is the last option header in the packet; an option type enumerator, *roh_type*; and the length of this option header, *roh_optlen*. Each RTP option header will have the same first three fields of the basic option header but will append extra fields for specific functionality. This allows the incoming data packet at the client to be parsed by casting an opaque byte packet pointer into an *rtpopthdr* pointer, looking at the *roh_type* and *roh_optlen* fields, figuring out the actual option header type, and then recasting the packet pointer to a specific option header structure pointer.

```
typedef struct {
      UINT rsh_fin:1;
      UINT rsh_type:7;
      UCHAR rsh_optlen;
      USHORT rsh_id;
} rtpssrchdr;
```

Figure 11: RTP synchronization source header

In a data stream being sent from a server to a client, the synchronization source option header (see Figure 11) will follow the standard RTP header in the RTP packet. This header adds the *rsh_id* field to the basic RTP option header. This ID is used to differentiate multiple streams being sent from the same server. In practice, a unique value is used for each synchronized data stream on the server. The first time a client sees a specific source ID, it will create a media source data handler and attach it to the input *logical device* (LDEV) for the appropriate content type (depending on the *rh_content* field of the RTP header).

When the server finishes sending the data stream to the client, it sends an RTP packet containing a bye header for the appropriate data stream (see Figure 12). The *rtbh_id* field contains the source ID of the stream which has been completed. Upon receipt of this header, the client removes the media source from the input LDEV for the appropriate content type.

```
typedef struct {
      UINT rtbh_fin:1;
      UINT rtbh_type:7;
      UCHAR rtbh_optlen;
      USHORT rtbh_id;
} rtcpbyehdr;
```

Figure 12: RTCP bye header

After the synchronization source header will come the source description header (see Figure 13). On a periodic basis (in this implementation, every five seconds) during data transmission, the media server will append several source description headers to the RTP packet. *rtsh_id* is the source ID of the current stream sent by the media server. The *rtsh_subtype* field will be one of the *rtp_sdesc_t* enumerators: computer address, port,

computer name, e-mail address, client user name, client user location, and a client-customizable text description (typically the filename of the media file being sent).

```
typedef enum {
      RTPSDESC_ADDR = 1,
      RTPSDESC_PORT = 2,
      RTPSDESC_CNAME = 4,
      RTPSDESC_EMAIL = 5,
      RTPSDESC_NAME = 6,
      RTPSDESC_LOC = 8,
      RTPSDESC_TXT = 16,
      RTPSDESC_PRIV = 255,
} rtp_sdesc_t;

typedef struct {
      UINT rtsh_fin:1;
      UINT rtsh_type:7;
      UCHAR rtsh_optlen;
      USHORT rtsh_id;
      UCHAR rtsh_subtype;
} rtcpsdeschdr;
```

Figure 13: RTCP source description header

Finally, any application data headers are appended to the RTP packet (see Figure 14). These option headers will contain the actual media data being sent from the server and any media-specific headers that are needed to setup the output devices. *rah_id* holds the source ID of the data stream being played. The *rah_subtype* field will contain an enumerator which defines what type of data is contained in the header. *rah_name* contains a unique 32-bit value which defines the application being used to send the media data streams. Finally, *rah_datalen* holds the length (in bytes) of the application data which is found immediately after this field in the RTP packet.

Video streams will send two application headers, one containing an AVISTREAMINFO header (RTPAPP_STREAMHEADER) and one containing a BITMAPINFOHEADER (RTPAPP_BMIHEADER), when the media server is instantiated. These are Windows-

specific multimedia headers which tell the output video *physical device* (PDEV) the original frame rate of the video file and the size, resolution, and compression format of the video frames. When playback begins on the server, an application header containing the actual bitmap data of the frame will be sent (RTPAPP_BITMAPDATA).

```
typedef enum {
      RTPAPP_BMIHEADER = 0,
      RTPAPP_BITMAPDATA,
      RTPAPP_STREAMHEADER,
      RTPAPP_AUDIOINFO,
      RTPAPP_AUDIODATA
} rtp_app_t;

typedef struct {
      UINT rah_fin:1;
      UINT rah_type:7;
      UCHAR rah_optlen;
      USHORT rah_id;
      USHORT rah_subtype;
      ULONG rah_name;
      UINT rah_datalen;
} rtpapphdr;
```

Figure 14: RTP application data header

Until now, we've been discussing RTP packets which are sent from the server to the client. One specific type of packet is sent from the client back to the server. This packet contains an RTP quality of service header (see Figure 15). On a periodic basic (in this implementation, once a second), the media source accumulates QoS statistics, fills in the QoS header and sends the RTP packet back to the client. The *rtqh_id* field contains the source ID of the data stream to which the client QoS data is referencing. The next eight fields contain the number of expected and received packets (since the last QoS packet was sent), and the minimum, average, and maximum latency of packet receipt (during the entire delivery of the data stream). The latency values are stored in two separate fields.

The final eight fields store the port and IP address of the server from which the data packets had been received.

```
typedef struct {
      UINT rtqh_fin:1;
      UINT rtqh_type:7;
      UCHAR rtqh_optlen;
      USHORT rtqh_id;
      ULONG rtqh_exppkt;
      ULONG rtqh_rcvpkt;
      USHORT rtqh_minsec;
      USHORT rtqh_minfra;
      USHORT rtqh_maxsec;
      USHORT rtqh_maxfra;
      USHORT rtqh_avgsec;
      USHORT rtqh_avgfra;
      UCHAR rtqh_subtype;
      UCHAR rtqh_sublen;
      USHORT rtqh_port;
      UCHAR rtqh_subtype2;
      UCHAR rtqh_sublen2;
      UCHAR rtqh_x3; // unused
      UCHAR rtqh_addrtype;
      ULONG rtqh_addr;
} rtcpqoshdr;
```

Figure 15: RTCP QoS header

4.3.3 Synchronization Engine

The synchronization engine consists of a set of devices - physical, logical, and compound logical - which are managed by logical time systems. The PDEVs encapsulate the actual multimedia devices on the client's machine. PDEVs can be designated as input or output devices. Input PDEVs manage the playback of received media data; output PDEVs manage the network transmission of media data. Output PDEVs can transmit media data from file storage or by direct capture.

In this implementation, there are both input and output PDEVs which abstract both the video display and audio playback devices. The output PDEVs, in this case, transmit stored media data directly from the file system. In a future implementation, output

PDEVs could be created for transmitting data captured in real-time from a microphone or video camera.

Each instance of a PDEV contains a thread which manages the timing of the playback or transmittal of the media data. PDEVs also contain a list of LTSs which are "driven" by the device. The PDEV thread sits in an event loop until it receives an "exit" event, at which time the loop is exited and the thread is destroyed. The event loop of the thread is executed at regular intervals. The *period* of the PDEV is the interval time (in milliseconds) of each pass through the loop. The *device rate* (in cycles/second) of the PDEV is calculated as the inverse of the PDEV period.

LDEVs are abstractions of physical devices. Multiple LDEVs can be bound to a single PDEV of the same type, but in this implementation, there is a one-to-one mapping between LDEVs and PDEVs. Upon creation, LDEVs are associated with a *compound logical device* (CLDEV), and a PDEV. The LDEV is added to a list of LDEVs held by both the CLDEV and the PDEV. Since there is a one-to-one mapping between LDEVs and PDEVs, the PDEV's LDEV list will always contain just one entry. A list of LDEVs is used because future implementations might make use of the extra functionality.

Input LDEVs manage a list of media sources. As new sources are received and current sources are terminated (through RTCP packets), the list will expand and shrink. Output LDEVs manage a list of media servers. The user will add and remove media servers through the user interface.

On creation of a media server, the content type that the media server will handle is fixed. After creation, the media server is instructed to load a media file. The media server will read the appropriate headers of the audio or video file and initialize itself to prepare for transmitting data packets. It is here where the original period of an output PDEV is set. (In this implementation, all media servers for a specific content type are managed by a single output PDEV and the *last* media server to be loaded defines the period of the PDEV.)

There can be input and output CLDEVs. In this application, there is one of each. Each CLDEV manages the network connection to a set of clients (output), or a set of servers (input). Socket creation and destruction is done by the CLDEV. The CLDEV is also used by the LDEVs to send packets to the socket. The CLDEV contains a network I/O thread which sits in an event loop and blocks waiting either for an "exit" event or a network packet. If the thread gets an exit event, the loop is exited and the thread is destroyed.

If a network packet is received and the CLDEV is an input CLDEV, the RTP option headers are parsed to see if a new media source has been received or an active media source is being terminated (e.g. a synchronization source header or bye header is received, respectively). Synchronization source information RTP packets that are received are parsed to extract the media server name, location, and e-mail fields. If an RTP application data header is received, the header is handed to the current media source.

If the current media source handles video data, the application data header is parsed to see if it contains an RTPAPP_BMIHEADER or RTPAPP_STREAMHEADER header. If so, the video source makes a local copy of the received header. If the application data header contains an RTPAPP_BITMAPDATA header, the bitmap data found after the header is copied into a pre-allocated buffer and pushed onto a queue for use during playback.

If the current media source handles audio data, the application data header is parsed to see if it contains an RTPAPP_AUDIOINFO header. If so, the audio source makes a local copy of the received header. If the application data header contains an RTPAPP_AUDIODATA header, the audio data found after the header is copied into an allocated buffer and pushed onto a queue for use during playback.

After each network packet is received, the current media source updates its QoS statistics and, at regular intervals (in this implementation, every second), it formats and sends QoS packets to the media server which is sending data to the current media source.

If a network packet is received and the CLDEV is an output CLDEV, then only QoS packets are parsed. The current media server (known from the QoS header) is called to respond to the QoS statistics.

The media server implements a rate control algorithm which dynamically changes the output PDEV period (for the appropriate media type). This algorithm can be disabled by unchecking the *Server/Use Rate Control* menu item (e.g. the *Use Rate Control* menu item in the *Server* menu). A penalty value is used to determine the change to the PDEV period. The rate control penalty is initialized to 0, and is incremented by 10 if one or more packets

were lost during the current QoS interval (in this implementation, every second). It is incremented by another 10 if one or more packets were lost during the previous QoS interval. If no packets were lost in both the current and previous period, the penalty is decremented by 10. The value of the penalty is thresholded to 0, on the low end, and 100, on the high end.

The new PDEV period (*m_period*) is calculated from the penalty value (*m_penalty*), the average frame rate (*avgfps*), and the optimal frame rate (*optfps*) using the calculation in Figure 16. The average frame rate is calculated from the average packet delay that is returned in the QoS header, and the optimal frame rate is calculated from the original PDEV period (set when the media server is initialized).

```
double pen_ratio = ((double)m_penalty / 100.0);
double complement_pen_ratio = 1.0 - pen_ratio;
m_fps = optfps * complement_pen_ratio + avgfps * pen_ratio;
m_period = (m_fps) ? (1.0 / m_fps) : 0.0;
```

Figure 16: Rate control calculation

Logical time systems manage the synchronization of the LDEVs held by each CLDEV. On creation, an LTS is associated with a PDEV. This PDEV is termed the *master PDEV* of the LTS. There are two types of LTS: *low-delay* and *skip-pause*. In this implementation, the input LDEVs are managed by a low-delay LTS and, depending on the state of the *Server/Use Low Delay LTS* menu item, the output LDEVs are managed by either a low-delay or skip-pause LTS. (See [Anderson91] for a complete description of low-delay and skip-pause LTSs.)

Each LTS keeps its own current logical time. For output LTSs, this time is incremented each time a media server sends a data packet, and for input LTSs, the time is incremented each time a data packet is received by a media source.

During each loop of a PDEV thread, each LTS will attempt to maintain synchronization of the LDEVs associated with the CLDEV to which the LTS is bound. At each time interval, the LTS will inform the PDEV associated with each LDEV to either play or skip the current frame. The low-delay and skip-pause LTSs each exhibit different algorithms for playing or skipping frames (See [Anderson91]).

When a PDEV is told to play or skip a frame, it tells its associated LDEV to play or skip that frame. When input LDEVs are told to play a frame, they step through their list of media sources and each video source is told to draw the next video frame. The video source will remove a bitmap data buffer from its queue (if one is available) and will use the VFW function *DrawDibDraw()* to decompress and render the bitmap to the display.

Output LDEVs step through their list of media servers and tell each one to play a frame. Both audio and video servers will call the VFW function *AVIStreamRead()* to read the next video frame or set of audio samples, respectively. Then, the media data will be packaged in an RTP packet and transmitted to the client using the socket held by the associated CLDEV. When media servers are told to skip a frame, they advance their media file pointers to the next frame (or audio sample), but do not transmit any data.

4.4 Application Walkthrough

We will now present a typical usage scenario of the application. This will involve the synchronized playback of two streams of continuous media (one audio stream and one video stream) using RTP over a UDP network connection.

The steps in the scenario:

- Start the application
- Connect to client
- Change server information (optional)
- Add audio and/or video media file(s)
- Change media file information (optional)
- Set server playback rate (optional)
- Start playback
- Suspend playback (optional)
- Resume playback (optional)
- Stop playback
- Disconnect from client
- Close the application

When starting up the application, the input devices are created (see Table 13).

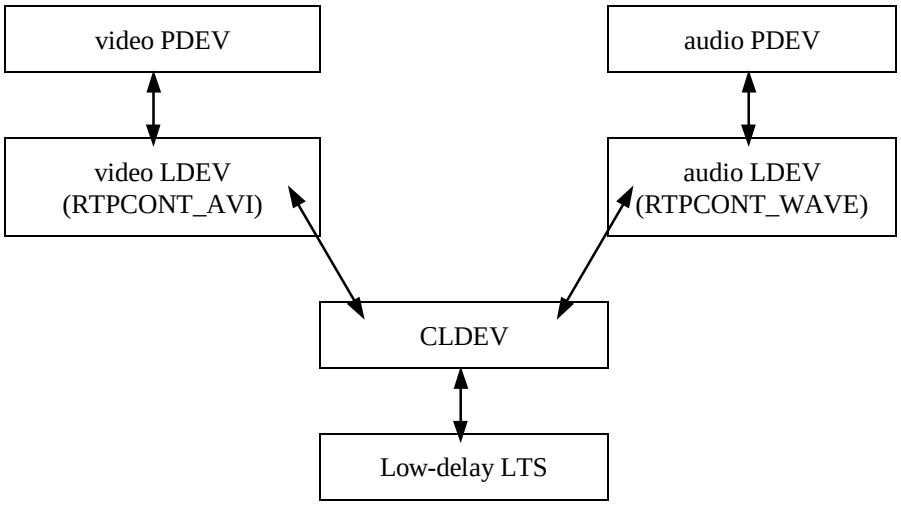

Figure 17: Data flow between physical, logical, and compound logical input devices and logical time system

Separate input PDEVs are created for audio and video stream playback. A low-delay LTS and an input CLDEV are created next. The LTS is bound to the input CLDEV. Separate LDEVs for AVI video and WAVE audio content are created and are bound to the input CLDEV. A network I/O thread for the input CLDEV is initiated and then the LTS is started. The input CLDEV thread will now block until it receives packets from the server.

Figure 18: Dialog box used for setting up connection with client

Next, the user must select a client to which the server will be connected. By pressing the *Connect to client* toolbar button or by selecting the *File/Connect to client...* menu item, the "Setup connection to client" dialog is opened (see Figure 17). The user can type in the

machine name or IP address of the desired client. A textual machine name will be converted to an IP address by calling the WinSock function *gethostbyname*. If the user selects OK on this dialog, the output devices will be created (see Table 14).

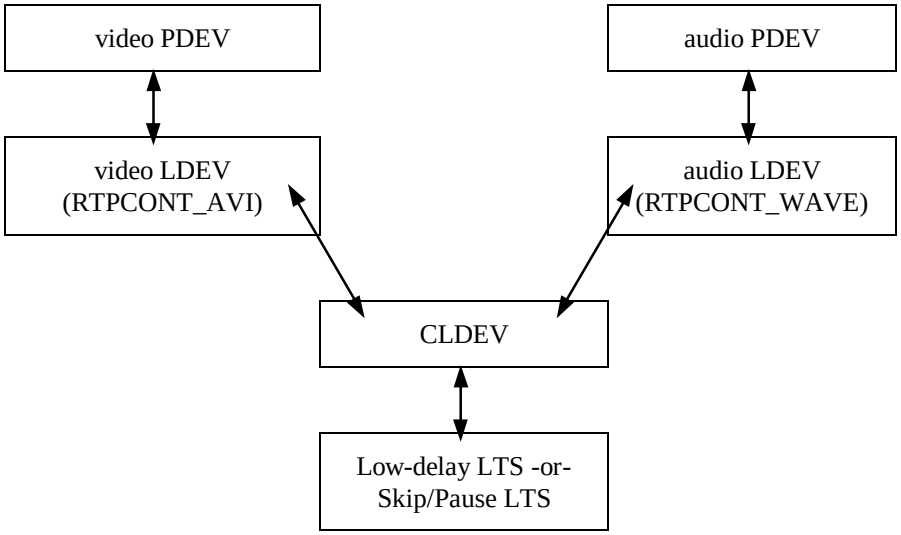

Figure 19: Data flow between physical, logical, and compound logical output devices and logical time system

Separate output PDEVs are created for audio and video stream transmission. Depending on the toggled state of the *Server/Use Low Delay LTS* menu item, either a low-delay or skip/pause LTS will be created. An output CLDEV is created and the LTS is bound to it. Separate LDEVs for AVI video and WAVE audio content are created and are bound to the output CLDEV. Next, a network I/O thread for the output CLDEV is initiated. The output CLDEV thread blocks until it receives packets from the client. (These packets will contain QoS information about the streams being transmitted.)

After the user has connected to a client, the toolbar state is updated to reflect accessible functionality (see Figure 18). Notice that the *Disconnect from client* button becomes

enabled while the *Connect to client* button becomes disabled. This provides for a context sensitive UI which allows the user to only access commands which apply to the current application state.

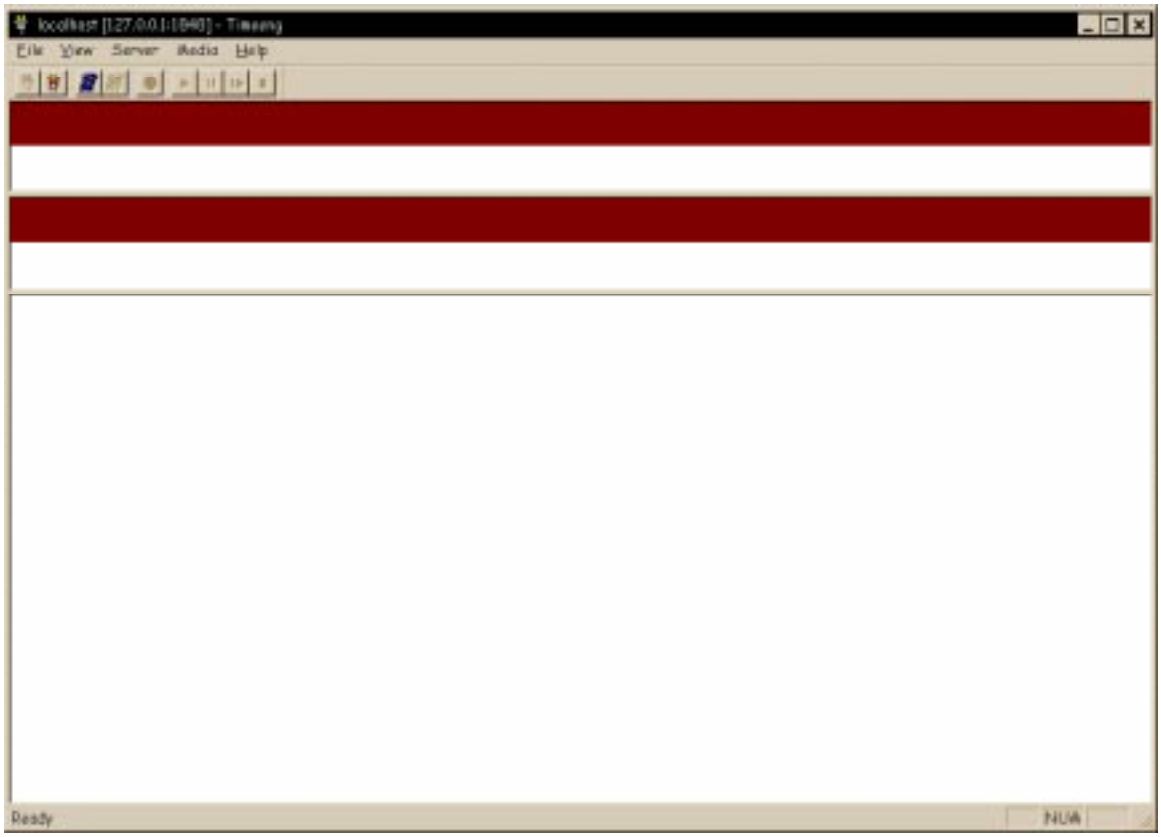

Figure 20: Application window shown after connecting to client *localhost*

When media data packets are sent to a client using RTP, information about the user is periodically sent with it. Once the user has connected to a client, this set of information can be edited. By selecting the *Server/Change Server Info...* menu item, the "Server Information" dialog is opened (see Figure 19). This dialog allows the user to change the user name, e-mail address, and location information which will be sent along with the media data packets. Changing this information is optional.

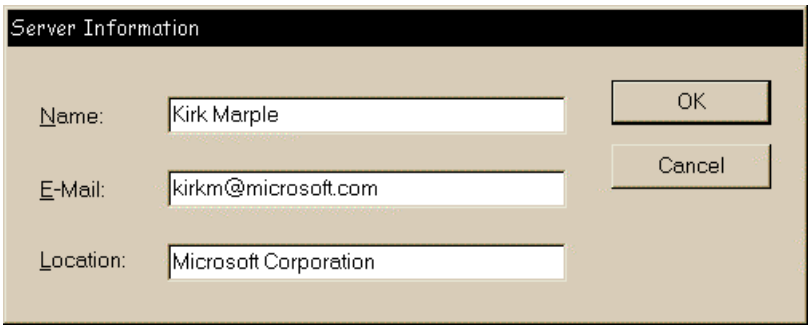

Figure 21: Dialog box used for editing server information

The next step in the application is to select one or more media files to be delivered to the client. Pressing the *Add media file* toolbar button or selecting the *Media/Add media file...* menu item will bring up the "Open multimedia file" dialog (see Figure 20). From this dialog, the user can choose either an AVI file (with .avi extension) or a Wave file (with .wav extension). AVI files normally contain a video stream and (optionally) an audio stream. Wave files contain just an audio stream.

If an AVI file is chosen, and if the *Media/Add Video Streams* menu item is toggled on, an instance of the CAVIVideoServer class is created and added to the output video LDEV. If an AVI file is chosen, and if the *Media/Add Audio Streams* menu item is toggled on, an instance of the CWaveAudioServer class is created and added to the output audio LDEV. If a Wave file is chosen, an instance of the CWaveAudioServer class is created and added to the output audio LDEV.

The filename of the media file chosen is passed as a parameter to the Load method on the CAVIVideoServer and CWaveAudioServer classes. On invocation, this method attempts

to read the headers of the desired stream in the media file. If the desired stream does not exist or some other error occurs, the server instance will not be added to the LDEV.

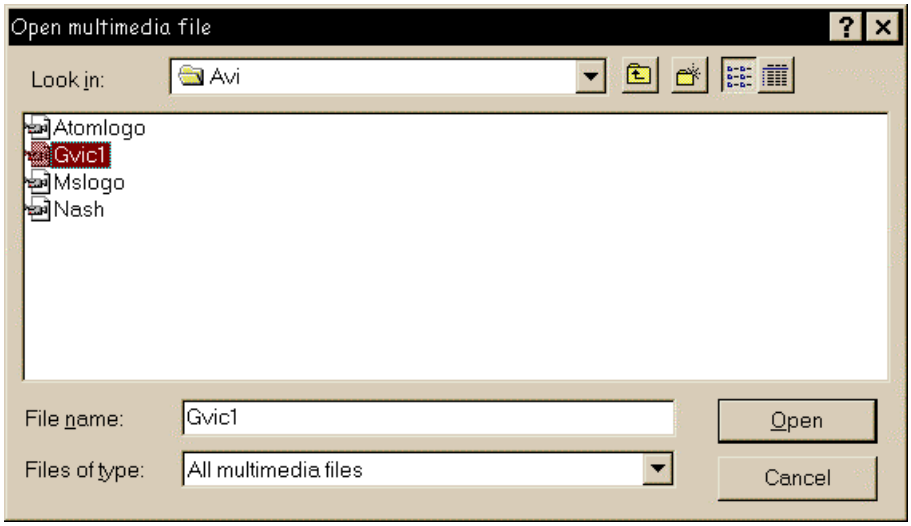

Figure 22: Dialog box used for selecting which media file to play

After the media file is added, the server information view on the server machine and the client information view on the client machine will be updated to show the new media server and media source, respectively (see Figure 21). If the client machine and the server machine are the same, both views are updated in the same application (as in Figure 21). Notice that the name and location information that can be edited in the "Server Information" dialog are present in the client information view for each media source. In the first row and second column of the client information view, the text *kirkm@kirkm-home*, which is the logged-in user and machine name of the server, is automatically generated from the network state of the server machine.

Multiple media files containing audio or video streams can be added to the server to be delivered to the client. In this scenario, we will show a single file containing two streams added to the server.

Figure 23: Application window shown after an AVI file is added to server list

Once a media file has been added, the text description for each media server can be edited by the user. This description is part of the RTP protocol (as is user name, location, etc.) and will be sent along with the media data packets. The default value for this text description (in this implementation) is the filename of the media file concatenated with either " (audio)" or " (video)" depending on if the media server delivers an audio or video stream.

The user can edit this information by clicking on a media server in the server information view and selecting the *Media/Change Media File Info...* menu item. This brings up the "Media File Information" dialog (see Figure 22). If the user edits the text and selects OK, the text description for the selected media server is updated. The next time the server sends that information (as part of the RTP protocol), the client information view on the client machine will reflect that change.

Figure 24: Dialog box used for editing media file information

After one or more video streams have been chosen for playback, the user can optionally edit the rate at which the video frames from all video servers will be delivered. *Note*: this is one major limitation of the current implementation. All video streams added to the server will be delivered at the same frame rate because they are all driven from the same PDEV. Currently, this frame rate is set to the frame rate of the *last* video stream added, unless overridden by the user.

Selecting the *Server/Change Playback Rate…* menu item brings up the "Server Rate" dialog (see Figure 23). This dialog shows the current playback rate (in frames per second) of the output video PDEV. If the user edits this value and presses OK, the frame rate and period of the output video PDEV will be updated.

Audio servers contain a fixed frame rate (15 fps). This means that each second of audio data is broken up into 15 equal-size packets and sent to the client. This frame rate is an arbitrary value and was chosen to provide an optimal packet size. The *Server/Change Playback Rate…* menu item is not accessible when an audio server is selected in the server information view.

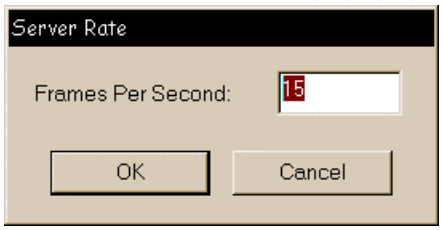

Figure 25: Dialog box used for editing server frame rate

To begin playback of all media servers, the user can press the *Play* button on the toolbar or select the *Server/Play* menu item. To pause or restart the playback, there are corresponding *Pause* and *Restart* buttons on the toolbar, in addition to *Server/Pause* and *Server/Restart* menu items.

After the Play command is invoked, and if the output devices are using a skip/pause LTS, the master PDEV of the LTS must be set. If any audio servers have been added, the master PDEV of the output LTS is set to the output audio LDEV. If no audio servers were added, the master PDEV is set to the output video LDEV. In either case, the output LTS is next started. Playback will begin on the client application as soon as data is received (see Figure 24).

If the Pause command is selected, the output LTS is suspended, and similarly, if the Restart command is selected, the output LTS is resumed.

Figure 26: Application window shown during server playback

To stop playback of the media devices, the user can press the *Stop* button on the toolbar or select the *Server/Stop* menu item. After the command is invoked, the media servers for both the output audio LDEV and the output video LDEV are stopped (see Figure 25). This involves sending the RTPOPT_BYE packet to the client for each media server. Following that, the output LTS is stopped.

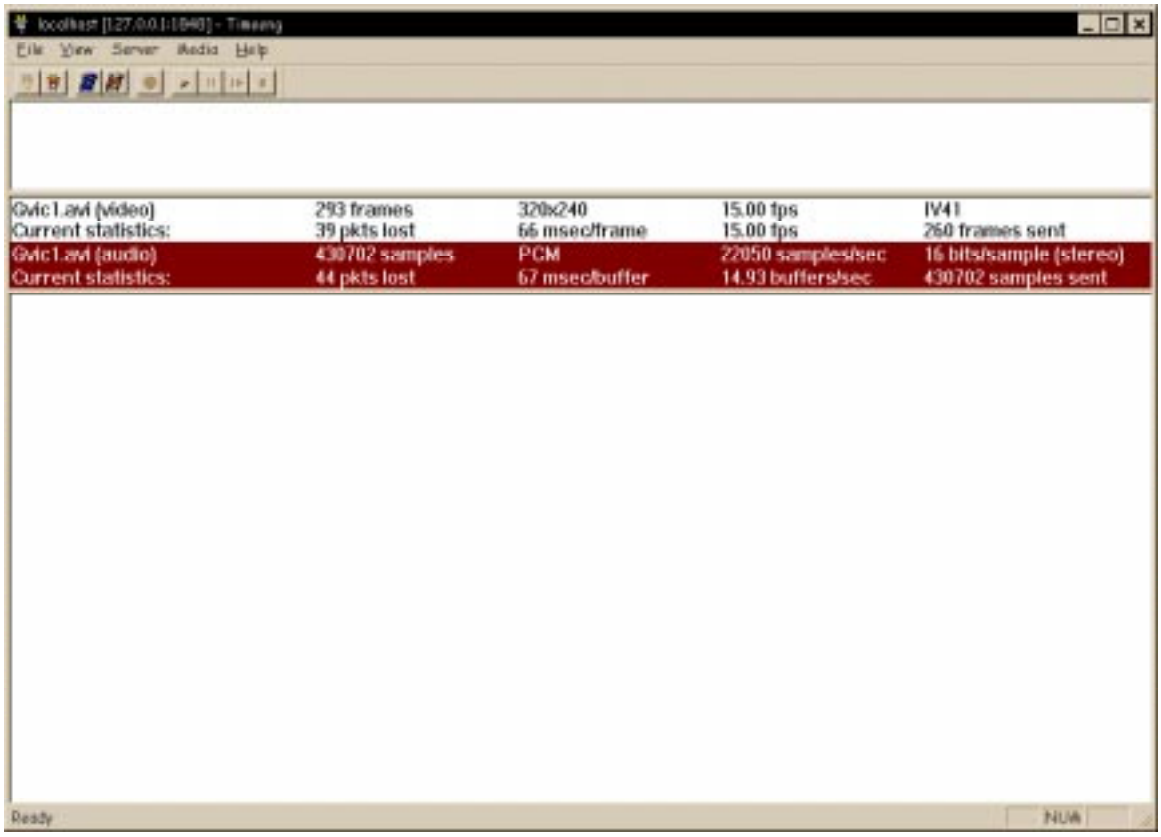

Figure 27: Application window shown after server playback has been stopped

To disconnect from the client, the user presses the *Disconnect from client* button on the toolbar or selects the *File/Disconnect from client* menu item. Invoking this command will stop the playback of the media servers (if it had not been stopped already), terminates the network I/O thread of the output CLDEV, and then deletes all output devices.

Closing the application disconnects from the client (if this had not been done already), stops the input LTS, terminates the network I/O thread of the input CLDEV, and then deletes all input devices.

4.5 Conclusion

In this implementation, a server can transmit multiple streams of video and audio data to a single client. In addition, a client can receive multiple video and audio streams from each of multiple servers. However, because of a design decision, a server cannot transmit to multiple clients.

Each of the server's video streams are transmitted at the same data rate. The client can receive video streams from multiple servers, but it will play them all back at a single frame rate. Audio streams are transmitted and played at a fixed frame rate (15 packets/second).

For future research, it would be useful to reorganize the code so that the server can transmit to multiple clients. It would also be useful to encapsulate the video source code so that each video source could have a separate data rate. Both of these tasks would require a different model for the device hierarchy.

In the current implementation, the server portion of the application has a dynamic list of media streams to be played and a single thread which drives the data transmission. Transmitting multiple video streams each to a different client would require an individual thread for each output video stream in order to provide accurate timing. Rather than having the server maintain a dynamic list of output streams, it would make sense to have a dynamic list of servers each of which would maintain a single output stream. This would give full control over the data transmission rate to each of the streams.

Similarly, in the current implementation, the client uses a single thread to decompress and render video data to the display device. To support the rendering of multiple video

streams, each at a different data rate, the application would require a different configuration of devices. One possibility would be to have multiple video devices each of which would manage the rendering to the display device of the video data at the proper time. A single network input thread would suffice to receive all video packets. These packets would then be passed off to the corresponding video source for playback.

The main problem that became apparent with this implementation was the loss of RTP data packets. The RTP protocol uses UDP which is a non-guaranteed protocol. It is well known that transmitting packets over a UDP socket will incur lost packets. In this case, packets are being lost even when they are transmitted to *localhost* (the same machine on which the server is running). Packets also are being lost when they are transmitted to another machine across an Ethernet LAN, which should be capable of data rates as high as 1.5 Mbps.

The packet loss when transmitting to *localhost* is most likely a result of either sending large UDP packets or thread scheduling. It has been noted in testing of the application that packets over 16 kB in size will be regularly dropped. Even though the buffer size on the sending and receiving sockets was set to 64 kB, some packets are sent but never received. Unfortunately, but consistent with the UDP semantics, no errors are returned to the application when these packets are dropped so we can not know for sure if this was the cause.

Another possible cause for the packet loss is thread scheduling. The application spawns seven threads during its execution. One is the user interface thread, two are for CLDEV

input and output network I/O, two are for the audio input and output PDEVs, and two are for the video input and output PDEVs. The PDEV threads are each executing a tight loop with a short period (about 66 msec). This most likely degrades the thread scheduling performance. Also, on the Windows 95 operating system, the use of too many threads is commonly known to degrade application performance. Because of differences in kernel thread support, the Windows NT operating system should probably not exhibit these limitations, but this hypothesis was not tested.

The cause for lost packets between machines on an Ethernet LAN is probably because the RTP packets are greater in size that the *Maximum Transmission Unit* (MTU) of the network. In general, UDP packets sent across the Internet must conform to a 512 byte MTU. In this application, the average packet size is normally around 8192 bytes.

The RTP protocol does contain a provision for breaking up these large packets. The *sync* bit in the standard RTP header is used to signal the last subpacket in a series of subpackets which make up a larger complete packet. In a future implementation, it would be useful to implement the decomposition of large packets into several subpackets. If this application were to be used to transmit media streams over the Internet, this feature would be a requirement.

In response to lost packets, the server data rate is altered dynamically to provide the highest data rate without sacrificing packet loss. This rate control algorithm has been shown (in the CMPlayer application [Rowe92]) to provide a higher average data rate than the “best effort” approach of trying to transmit data packets at a constant rate.

One limitation that was found with the rate control algorithm was that it did not take the quantity of lost packets into account when calculating the penalty. One alternative that was tested was to scale the penalty value by the ratio of lost packets to expected packets. This seemed to give a more valid penalty value with different levels of packet loss.

After some testing, the penalty method for the rate control algorithm still did not reflect the current packet loss very accurately. Another algorithm was tried which calculated the penalty ratio as a weighted average of the current lost packet ratio and the lost packet ratio at the last QoS notification. The current ratio was weighted 75% and the previous ratio was weighted 25%. This new penalty ratio was used in the existing equation which linearly interpolated a new frame rate between the optimal frame rate and the average frame rate. This method seemed more adaptive and more indicative of the actual packet loss. It is possible to directly calculate the new frame rate from the current packet loss ratio and the optimal frame rate, but the linear interpolation was added in order to smooth out the frame rate changes and to limit the jitter of packet transmission times.

This new algorithm still was not perfect and did not seem to lower the frame rate enough because the lower bound was always the average frame rate. Reducing the lower bounds of the interpolation (heuristically or statistically) would most likely provide a more accurate result and allow the frame rate variance to better reflect the actual data rate. In future research, statistical analysis should be done to compare the performance of different rate control algorithms.

One side effect of using UDP packets is the possible loss of RTCP packets which signal the start and end of media streams. For example, if an RTCP bye packet was lost in the network, the receiving client would never realize that the stream had ended and would think that the data transmission had just been stalled. Other applications use TCP for the session control because of the guaranteed delivery of transmitted packets. This is one valid solution to the problem. As was mentioned earlier, if we were to use TCP for all data transmission, performance would be degraded by the delay caused by the retransmission of lost packets.

Another solution to this problem would be to return an acknowledgment packet to the server when a client is creating or destroying a video source. The server would not allow data transmission to proceed until it had received the acknowledgment packet. If the server does not receive the acknowledgment packet in a prescribed amount of time, it will resend the RTCP packet. If this happens, the client may see an RTCP packet for an already opened or closed video source. The client will ignore this duplicate packet, conclude that the acknowledgment packet had been lost during transmission, and will resend the acknowledgment packet to the server.

The loss of data packets can cause a discontinuity in media stream playback which can appear as jumpiness in a video stream or as stuttering in an audio stream. For video streams, if a keyframe is lost, the stream will exhibit a greater loss in visual quality than if an intermediate frame is lost. For audio streams, the application uses a fixed frame rate of 15 fps, so each audio packet lost in transmission will cause a 67 msec gap in audio playback.

Video streams that use sub-frame encoding of video frames will exhibit less discontinuity in the playback than streams which use full frame encoding. If a data packet containing a sub-frame is lost, only a small area (i.e. 8×8 pixel) of the frame would be lost. Some parts of the video frame will be up-to-date while other sub-frames will lag behind. Typically, this is sufficient for video conferencing applications.

Some applications will typically send only sub-frames which have changed since the last full frame. These sub-frames describe the most active areas of the full frame. The areas of the frame which have not changed recently can be called static sub-frames. Static sub-frames are sent on a regular, but infrequent, basis. Because of this, there will be a larger ratio of active sub-frames to static sub-frames which are transmitted. Consider the situation where there are 9 active sub-frames for every 1 static sub-frame, and there is a constant rate of 1 out of every 10 frames lost in the network. The active areas of the frame will be up-to-date (from receiving 8 sub-frames) while the static areas may rarely, if ever, get refreshed (from receiving 0 sub-frames).

For audio streams, one possible optimization would be to interlace the samples in every two audio buffers such that, if one of the pair of packets were lost, the other packet can be de-interlaced to generate two packets with every other sample being valid. This results in shorter gaps between valid audio data and, if the compression algorithm supports it, the valid samples in the two packets can be interpolated to generate the missing samples. Simpler compression algorithms, such as PCM or A-Law, support this type of interpolation because the data must be decompressed and interpolated before playback.

Another problem that occurred with audio streams is when enough packets were lost, the number of buffered samples of audio can drop to zero. This causes stuttering of the audio stream because there is a race condition between packets of samples being received and packets of samples being played. In order to overcome this problem, if the number of audio packets lost is greater than one half of the number of packets normally sent per second (an arbitrary threshold), the audio playback is paused until enough samples are buffered. Once enough samples are buffered, the audio playback is resumed. In practice, this works well and provides smoother playback at the cost of longer periods of silence after severe packet loss, but with the benefit of fewer discontinuities in the audio playback.

Most of the related applications (including ACME and Netvideo) run on the Unix platform. By writing to the high-level Win32 API, our software application is less dependent on the underlying hardware and, therefore, can transparently make use of advances in video and audio codecs, multiprocessor support, and new display devices. Using the Win32 APIs and MFC for development greatly decreased the implementation time and made the application more extensible and portable.

This project was begun two years ago and there have been many changes in the computer industry since that time. However many, if not all, of the original design decisions are still valid. The application has been able to make the transition to the newer 32-bit operating systems, such as Microsoft Windows NT 4.0, without change. In addition as more efficient video codecs become available, the application is able to make use of content compressed with a new codec without any changes to the software. The Internet has now become commonly used for networked data delivery. With only the minor packet

fragmentation changes noted earlier, the application should be able to transmit and receive media data over the Internet.

This software application has been tested in a multi-machine environment on a LAN. Further testing with a larger number of machines over a larger LAN or a WAN is certainly required before generalizing the performance statistics. Nevertheless, the experience reported in this chapter suggests that the goals of extensibility, portability and interoperability have been achieved to a large degree by this design.

GLOSSARY

ADPCM: Adaptive Delta Pulse Code Modulation
ADU: Application Data Unit
ASP: Application Synchronization Protocol
ATM: Asynchronous Transfer Mode
CD-I: Compact Disc-Interactive
CLDEV: Compound Logical Device
codec: compressor-decompressor
DCT: Discrete Cosine Transformation
DDA: Device Dependent Audio
DIA: Device Independent Audio
DVI: Digital Video Interactive
DWT: Discrete Wavelet Transformation
EIDE: Enhanced Integrated Drive Electronics
GIF: Graphics Interchange Format
HTML: HyperText Markup Language
HTTP: HyperText Transfer Protocol
ISDN: Integrated Service Digital Network
JPEG: Joint Photographic Experts Group
LDEV: Logical Device
LTS: Logical Time System
MIDI: Musical Instrument Digital Interface
MPEG: Motion Picture Experts Group
MTP: Movie Transmission Protocol
MTU: Maximum Transmission Unit
MVC: Multimedia Virtual Circuit
NSP: Network Synchronization Protocol
NTP: Network Time Protocol
PAM: Pulse Amplitude Modulation
PDEV: Physical Device
PLV: Production Level Video
PNM: Pulse Number Modulation
PPM: Pulse Position Modulation
PWM: Pulse Width Modulation
QoS: Quality of Service
RTCP: Real-Time Control Protocol
RTP: Real-Time Protocol
RTV: Real-Time Video
SCSI: Small Computer System Interface
TIFF: Tagged Image File Format
VC: Virtual Circuit
VLC: Variable-Length Coding
VQ: Vector Quantization

BIBLIOGRAPHY

[Ahrens94] Ahrens, Richard L., "Internet Voice Chat," software application, 1994.

[Ahuja92] Ahuja, S.R. and J.R. Esnor, "Coordination and control of multimedia conferencing," *IEEE Communications*, May 1992, pp. 38-43.

[Anderson90] Anderson, David P., Ramesh Govindan, George Homsy, and Robert Wahbe, "Integrated Digital Continuous Media: A framework based on Mach, X11, and TCP/IP," *Technical Report, University of California, Berkeley*, Aug. 1990.

[Anderson91] Anderson, David P. and George Homsy, "A continuous media I/O server and its synchronization mechanism," *IEEE Computer*, vol. 24, no. 10, Oct. 1991, pp. 51-57.

[Anderson91a] Anderson, David P., Yoshitomo Osawa, and Ramesh Govindan, "Real-time disk storage and retrieval of digital audio/video data," *Technical Report, University of California, Berkeley,* Sep. 1991.

[Anderson91b] Anderson, David P., and Pamela Chan, "Toolkit support for multiuser audio/video applications," *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, Nov. 1991, pp. 230-241.

[Anderson93] Anderson, David P., "ACME Programmer's Guide," *Technical Report, University of California, Berkeley*, 1993.

[Baecker92] Baecker, R.M. (Ed., *Readings in Groupware and Computer Supported Collaborative Work*, Morgan Kaufmann Publishers, 1992.

[Blackketter92] Blackketter, Dean and Greg Gretsch, "Building network-based interactive media," *Digest of Papers - IEEE Computer Society International Conference*, Feb. 1992, pp. 58-63.

[Blair91] Blair, Gordon S. and Nigel Davies, "Incorporating multimedia in distributed object-oriented systems: The importance of flexible management," *Proceedings of the International Workshop on Object Orientation in 1991*, pp. 36-41.

[Blair92] Blair, Gordon S., Geoff Coulson, Nigel Davies, and Neil Williams, "The role of operating systems in object-oriented distributed multimedia platforms," *Proceedings of the Second ACM/IEEE Workshop on Object Orientation in Operating Systems*, Sep. 1992.

[Brown93] Brown, Eric, "Codec á Go Go: Who's squeezing who?" *New Media Magazine*, Mar. 1993, pp. 40-42.

[Christodoulakis86] Christodoulakis, Stavros, and Christos Faloutsos, "Design and performance considerations for an optical disk-based, multimedia object

server," *IEEE Computer*, vol. 19, no. 12, Dec. 1986, pp. 45-56.

[Cogger94] Cogger, Richard, and Tim Dorcey, "CU/SeeMe for Windows," Cornell University, software application, 1994.

[Davenport91] Davenport, Glorianna, Thomas Aguierre Smith, and Natalio Pincever, "Cinematic primitives for multimedia," *IEEE Computer Graphics and Applications*, vol. 11, no. 4, Jul. 1991, pp. 67-74.

[Donovan91] Donovan, J.W., "Intel/IBM's audio-video kernel," *Byte Magazine*, vol. 16, no. 13, Dec. 1991, pp. 177-186.

[Doyle93] Doyle, Bob, "Crunch time for digital video," *New Media Magazine*, Mar. 1993, pp. 47-50.

[Doyle93a] Doyle, Bob, "How codecs work," *New Media Magazine*, Mar. 1993, pp. 52-55.

[Drapeau91] Drapeau, G.D., and H. Greenfield, "MAEstro - A distributed multimedia authoring environment," *Proceedings of the Summer 1991 USENIX Conference*, 1991, pp. 315-328.

[Educational95] Educational Testing Service, "Graduate Record Exam", Princeton, NJ, 1995.

[Federighi94] Federighi, Craig, and Lawrence A. Rowe, "A distributed hierarchical storage manager for a video-on-demand system," *Storage and Retrieval for Image and Video Databases II: IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, Feb. 1994.

[Fogg96] Fogg, Chad, "MPEG-2 FAQ - Questions that should be frequently asked about MPEG," Apr. 1996.

[Foley90] Foley, James D., Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice*, Second Edition, Addison-Wesley, 1990.

[Fox91] Fox, Edward A., "Advances in interactive digital multimedia systems," *IEEE Computer*, vol. 24, no. 10, Oct. 1991, pp. 9-21.

[Frederick93] Frederick, Ronald, "Netvideo version 3.3," Xerox Corporation, software application, 1993.

[Gemmell92] Gemmell, James and Stavros Christodoulakis, "Principles of delay sensitive multimedia data storage and retrieval," *ACM Transactions on Information Systems*, vol. 10, no. 1, 1992, pp. 51-90.

[Gibbs91] Gibbs, Simon, "Composite multimedia and active objects," *OOPSLA '91, SIGPLAN Notices*, vol. 26, no. 11, 1991, pp. 97-112.

[Govindan91] Govindan, Ramesh, and David P. Anderson, "Scheduling and IPC mechanisms for continuous media," *13th ACM Symposium on Operating Systems Principles*, 1991.

[Harney91] Harney, Kevin, Mike Keith, Gary Lavelle, Lawrence D. Ryan, and Daniel J. Stark, "The i750 video processor: A total multimedia solution," *Communications of the ACM*, vol. 34, no. 4, Apr. 1991, pp. 64-78.

[Harrison89] Harrison, Steve, Scott Minneman, Bob Stults, and Karon Weber, "Video: A design medium," *SIGCHI Bulletin*, vol. 21, no. 2, Oct. 1989, pp. 62-66.

[Harrison92] Harrison, Beverly L. and Ronald Baecker, "Designing video annotation and analysis systems," *Proceedings of Graphics Interface '92*, May 1992, pp. 157-166.

[Haskins93] Haskins, Roger L., "The Shark continuous-media file server," *Proceedings of IEEE COMPCON '93*, Feb. 1993, pp. 12-15.

[Hoffert92] Hoffert, Eric, et al., "QuickTime: An extensible standard for digital multimedia," *Digest of Papers - IEEE Computer Society International Conference*, Feb. 1992, pp. 15-20.

[Hopper90] Hopper, Andy, "Pandora: An experimental system for multimedia applications," *ACM Operating Systems Review*, vol. 24, no. 2, Apr. 1990, pp. 19-34.

[Jeffay91] Jeffay, K., D.L. Stone, T. Talley, and F.D. Smith, "Adaptive, best-effort delivery of digital audio and video across packet-switched networks," *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, Nov. 1991, pp. 3-14.

[Kim91] Kim, Y., "Chips deliver multimedia," *Byte Magazine*, vol. 16, no. 13, Dec. 1991, pp. 163-173.

[Kretz92] Kretz, Francis, and Françoise Colaïtis, "Standardizing hypermedia information objects," *IEEE Communications*, pp. 60-70, May 1992.

[Lamparter91] Lamparter, Bernd, and Wolfgang Effelsberg, "X-MOVIE: Transmission and presentation of digital movies under X," *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, Nov. 1991, pp. 328-339.

[Lane95] Lane, Thomas, "JPEG Image Compression FAQ," Usenet newsgroup *comp.graphics*, Jan. 1995.

[LeGall91] Le Gall, Didier, "MPEG: A video compression standard for multimedia applications," *Communications of the ACM*, vol. 34, no. 4, Apr. 1991, pp. 46-58.

[Lehtinen94] Lehtinen, Rick, "Dawn of the video network," *AV Video Magazine*, Jan. 1994, pp. 19-23.

[Lemay95] Lemay, Laura, *Teach Yourself Web Publishing with HTML in a week*, Sams Publishing, 1995, pp. 11-15.

[Levergood93] Levergood, Thomas M., Andrew C. Payne, James Gettys, G. Winfield Treese, and Lawrence C. Stewart, "AudioFile: A network-transparent system for distributed audio applications," *Proceedings of the USENIX Summer Conference*, Jun. 1993.

[Liebhold91] Liebhold M. and E. Hoffert, "Towards an open environment for digital video," *Communications of the ACM*, vol. 34, no. 4, Apr. 1991, pp. 103-112.

[Liou91] Liou, M., "Overview of the p×64 Kbits/s video coding standard," *Communications of the ACM*, vol. 34, no. 4, Apr. 1991, pp. 59-63.

[Little90] Little, Thomas D.C. and Arif Ghafoor, "Network considerations for distributed multimedia object composition and communication.," *IEEE Network*, vol. 4, no. 6, 1990, pp. 32-49.

[Little90a] Little, Thomas D.C. and Arif Ghafoor, "Synchronization and storage models for multimedia objects," *IEEE Journal on Selected Areas in Communication*, vol. 8, no. 3, Apr. 1990, pp. 413-427.

[Little91] Little, Thomas D.C. and Arif Ghafoor, "Spatio-temporal composition of distributed multimedia objects for value-added networks," *IEEE Computer*, Oct. 1991, pp. 42-50.

[Little93] Little, Thomas D.C., "A framework for synchronous delivery of time-dependent multimedia data," *ACM Multimedia Systems*, vol. 1, no. 2, 1993, pp. 87-93.

[Little93a] Little, Thomas D.C., G. Ahanger, R.J. Folz, J.F. Gibbon, F.W. Reeve, D.H. Schelleng, and D. Ventatash, "A digital on-demand video service supporting content-based queries," *Proceedings of ACM Multimedia '93*, Aug. 1993.

[Loeb92] Loeb, Shoshana, "Delivering interactive multimedia documents over networks," *IEEE Communications*, May 1992, pp. 52-59.

[Lougher93] Lougher, P. and D. Shepherd, "The design of a storage server for continuous media," *The Computer Journal*, vol. 36, no. 1, Jan. 1993, pp. 32-42.

[Luther91] Luther, A.C., *Digital Video in the PC Environment*, Intel/McGraw-Hill, 1991.

[Mackay89] Mackay, Wendy E., "EVA: An experimental video annotator for symbolic analysis of video data.," *SIGCHI Bulletin*, vol. 21, no. 2, 1989, pp. 68-71.

[Mackay89a] Mackay, Wendy E. and Glorianna Davenport, "Virtual video editing in interactive multimedia applications," *Communications of the ACM*, vol. 32, no. 7, Jul. 1989, pp. 802-810.

[Markey92] Markey, Brian D., "HyTime and MHEG," *Digest of Papers - IEEE*

Computer Society International Conference, Feb. 1992, pp. 25-40.

[Media91] Media Cybernetics, Inc., *HALO Image File Format Library, Programming Manual*, Version 1.1, 1991.

[Microsoft95] Microsoft Corporation, "Sound Recorder," software application, Windows 95 operating system, 1995.

[Murray95] Murray, James D., "Graphics File Formats Frequently Asked Questions (FAQ)," Usenet newsgroup *comp.graphics*, Jan. 1995.

[Nichols91] Nichols, Kathleen M., "Performance studies of digital video in a client/server environment," *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, Nov. 1991, pp. 81-91.

[Nicolaou90] Nicolaou, Cosmos, "An architecture for real-time multimedia communication systems ," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, Apr. 1990, pp. 391-400.

[Perey94] Perey, Christine, "A field guide to software video codecs," *Desktop Video World Magazine*, Mar. 1994, pp. 28-34.

[Pohlmann89] Pohlmann, Ken C., *Principles of Digital Audio*, Second Edition, Sams Publishing, 1989, pp. 72-73, 146, 152-154, 364-366.

[Progressive95] Progressive Networks, "RealAudio Player," software application, 1995.

[Ramanathan92] Ramanathan, Srinivas, P. Venkat Rangan, and Harrick M. Vin, "Integrating virtual reality, tele-conferencing, and entertainment into multimedia home computers," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 2, May 1992.

[Rangan91] Rangan, P. Venkat and Harrick M. Vin, "Designing file systems for digital video and audio," *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP '91*, vol. 25, no. 5, Oct. 1991, pp. 69-79.

[Rangan92] Rangan, P. Venkat, Harrick M. Vin, and Srinivas Ramanathan, "Designing an on-demand multimedia service," *IEEE Communications*, vol. 30, no. 7, Jul. 1992, pp. 56-65.

[Rangan93] Rangan, P. Venkat and Srinivas Ramanathan, "Adaptive feedback techniques for synchronized multimedia retrieval over integrated networks," *IEEE/ACM Transactions on Networking*, vol. 1, no. 2, Apr. 1993.

[Rangan93a] Rangan, P. Venkat, Harrick M. Vin, Srinivas Ramanathan, and Thomas Kaeppner, "Techniques for multimedia synchronization in network file systems," *Computer Communications Journal*, Mar 1993.

[Reisman91] Reisman, Sorel, "Developing multimedia applications," *IEEE Computer*

Graphics and Applications, vol. 11, no. 4, Jul. 1991, pp. 52-57.

[Ripley89] Ripley, G. David, "DVI: A digital multimedia technology," *Communications of the ACM*, vol. 32, no. 7, Jul. 1989, pp. 811-822.

[Rosenberg91] Rosenberg, J., Gil Cruz, and Thomas Judd, "Presenting multimedia documents over a digital network," *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, Nov. 1991, pp. 346-357.

[Rosenberg92] Rosenberg, J., "Multimedia communications for users," *IEEE Communications*, May 1992, pp. 20-36.

[Rowe92] Rowe, Lawrence A. and Brian C. Smith, "A continuous media player," *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, Nov. 1992.

[Rowe94] Rowe, Lawrence A. John S. Boreczky, and Charles A. Eads, "Indexes for user access to large video databases," *Storage and Retrieval for Image and Video Databases II: IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, Feb. 1994.

[Rowe94a] Rowe, Lawrence A., Ketan D. Patel, Brian C. Smith, and Kim Liu, "MPEG Video in Software: Representation, transmission, and playback," *High Speed Networking and Multimedia Computing: IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, Feb. 1994.

[Rubin91] Rubin, Michael, *NonLinear: A Guide to Electronic Film and Video Editing*, Triad Publishing Company, 1991, pp. 99-117.

[Schulzrinne93] Schulzrinne, H., and S. Caster, "RTP: A transport protocol for real-time applications," *IETF working draft, Audio-Video Transport Working Group*, Oct. 1993.

[Schürmann90] Schürmann, Gerd, and Uwe Holzmann-Kaiser, "Distributed multimedia information handling and processing," *IEEE Network*, Nov. 1990, pp. 23-31.

[Scott96] Scott, Michael, "comp.sys.ibm.pc.hardware.video FAQ," Usenet newsgroup *comp.sys.ibm.pc.hardware.video*, May 1996.

[Shepherd91] Shepherd, Doug, David Hutchinson, Francisco Garcia, and Geoff Coulson, "Protocol support for distributed multimedia applications," *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, Nov. 1991.

[Sijstermans91] Sijstermans, Frans, and Jan van der Meer, "CD-I full-motion video encoding on a parallel computer," *Communications of the ACM*, vol. 34, no. 4, Apr. 1991, pp. 81-91.

[Smith92] Smith Jr., K.H., "Accessing multimedia network services," *IEEE Communications*, May 1992, pp. 72-80.

[Staehli93] Staehli, Richard, and Jonathan Walpole, "Script-based QOS specifications for multimedia presentations," *Technical Report, Oregon Graduate Institute of Science & Technology*, 1993.

[Steinmetz90] Steinmetz, Ralf, "Synchronization properties in multimedia systems," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, Apr. 1990, pp. 401-412.

[Terry88] Terry, D.B. and D.C. Swinehart, "Managing stored voice in the Etherphone system," *ACM Transactions on Computer Systems*, vol. 6, no. 1, Feb. 1988, pp. 3-27.

[Tinker89] Tinker, Michael, "DVI parallel image compression," *Communications of the ACM*, vol. 32, no. 7, Jul. 1989, pp. 844-851.

[Tobagi92] Tobagi, Fouad A. and James E. Long, "Client-server challenges for digital video," *Digest of Papers - IEEE Computer Society International Conference*, Feb. 1992, pp. 88-91.

[Uppaluru92] Uppaluru, Prem, "Networking digital video," *Digest of Papers - IEEE Computer Society International Conference*, Feb. 1992, pp. 76-83.

[van Rossum94] van Rossum, Guido, "FAQ: Audio File Formats," Usenet newsgroup *comp.multimedia*, Feb. 1994.

[Vin91] Vin, Harrick M., Polle T. Zellweger, Daniel C. Swinehart, and P. Venkat Rangan, "Multimedia conferencing in the Etherphone environment," *IEEE Computer*, Oct. 1991, pp. 69-79.

[Vin93] Vin, Harrick M. and P. Venkat Rangan, "Designing a multi-user HDTV storage server," *IEEE Journal on Selected Areas in Communications - Special Issue on High Definition Television and Digital Video Communication*, vol. 11, no. 1, Jan 1993.

[Wayner91] Wayner, P., "Inside QuickTime," *Byte Magazine*, vol. 16, no. 13, Dec. 1991, pp. 189-196.

[Williams91] Williams, Neil, Gordon S. Blair, and Nigel Davies, "Distributed multimedia computing: An assessment of the state of the art," *Information Services and Use*, vol. 11, 1991, pp. 265-281.

[Wolfinger91] Wolfinger, Bernd and Mark Moran, "A continuous media data transport service and protocol for real-time communication in high speed networks," *Technical Report, University of California, Berkeley*, 1991.

[Yager91] Yager, T., "Information's human dimension," *Byte Magazine*, vol. 16, no. 13, Dec. 1991, pp. 153-160.