An Investigation into on-Demand Multimedia Services in a Real-time, Distributed Computing Context

By

Pareen Jamnadas Daya

Submitted in partial fulfillment of the requirement for the Bachelor of Science Honours in Computer Science and Information Systems

12 November 2001

Department of Computer Science
Rhodes University
P Bag 94
Grahamstown, 6140
ABSTRACT

This aim of this project is to investigate and analyse the transmission of multimedia data using the Java Media Framework Application Programming Interface (JMF API). Classes of multimedia investigated include “live” and “stored” video and audio data using the Real-time Transport Protocol (RTP). Further investigation into other transmission protocols such as the Real-time Streaming Protocol (RTSP) is also addressed.

The results produced with respect to the performance of streaming multimedia data has enabled the deployment of a simple desktop video conferencing system and a number of other video streaming applications, thus showing the capabilities of the Real-time Transport Protocol.

The implementation of facilities enabling the efficient transfer of multimedia data has culminated in the design and deployment of a pilot Video-on-Demand application that utilises the eLinda architecture for distributed and parallel computing in a distributed multimedia context.
# TABLE OF CONTENTS

## CHAPTER 1:
**INTRODUCTION** ................................................................................................................ 1

1.1 BACKGROUND ..................................................................................................................... 2
1.2 CURRENT TECHNOLOGIES ................................................................................................. 3
   1.2.1 Media Presentation ........................................................................................................ 3
   1.2.2 Presentation Controls .................................................................................................. 3
   1.2.3 Latency ....................................................................................................................... 3
   1.2.4 Presentation Quality .................................................................................................... 4
1.3 CHALLENGES OF ON-DEMAND MEDIA PRESENTATION ............................................. 4
   1.3.1 Application Areas of on-Demand Multimedia Services ............................................ 5
   1.3.2 Requirements for a Successful Multimedia-on-Demand Implementation ................ 5
1.4 PROJECT DESCRIPTION AND GOALS ............................................................................. 6
   1.4.1 Project Statement ....................................................................................................... 6
   1.4.2 Project Deliverables ................................................................................................. 7
1.5 THESIS OVERVIEW ........................................................................................................... 7

## CHAPTER 2:
**MULTIMEDIA ENCODING** ................................................................................................. 9

2.1 COMPRESSION OF MULTIMEDIA DATA ......................................................................... 10
   2.1.1 Lossless Media Compression ..................................................................................... 10
   2.1.2 Lossy Media Compression ........................................................................................ 11
   2.1.3 Spatial Compression .................................................................................................. 12
   2.1.4 Temporal Compression ............................................................................................. 12
2.2 MULTIMEDIA CODECS ...................................................................................................... 13
   2.2.1 Hardware Codecs .................................................................................................... 13
   2.2.2 Software Codecs ..................................................................................................... 14
2.3 COMMON MEDIA FORMATS .............................................................................................. 14
2.4 COMMON VIDEO COMPRESSION FORMATS .................................................................. 16
   2.4.1 JPEG (Joint Photographic Experts Group) ................................................................ 16
   2.4.2 MPEG (Moving Pictures Expert Group) ................................................................... 18
      2.4.2.1 MPEG-1 ............................................................................................................. 18
      2.4.2.2 MPEG-2 ............................................................................................................. 18
      2.4.2.3 MPEG-3 ............................................................................................................. 19
      2.4.2.4 MPEG-4 ............................................................................................................. 19
   2.4.3 H.261 and H.263 Video Encoding Formats ................................................................. 20
2.5 COMMON AUDIO FORMATS ............................................................................................ 20
   2.5.1 G.711 and G.723 Audio Encoding Formats ............................................................... 21
2.6 MEDIA PRESENTATION ....................................................................................................... 21
   2.6.1 Multimedia Processing ............................................................................................... 22
   2.6.2 Media Capture .......................................................................................................... 22
   2.6.3 Capture Devices ........................................................................................................ 23
   2.6.4 Demultiplexers and Multiplexers ............................................................................. 23
   2.6.5 Compositing .............................................................................................................. 24
CHAPTER 3: DEVELOPMENT ENVIRONMENT ................................................................. 26

3.1 JAVA™ TECHNOLOGY ............................................................................................ 27
3.2 THE JAVA™ MEDIA FRAMEWORK API ................................................................. 28
   3.2.1 Managers ........................................................................................................... 29
   3.2.2 Data Sources ..................................................................................................... 29
   3.2.3 Players ................................................................................................................ 29
   3.2.4 Processors .......................................................................................................... 30
3.3 REAL-TIME TRANSPORT PROTOCOL ............................................................... 30
   3.3.1 Overview of the Real-time Transport Protocol ................................................. 31
   3.3.2 Features of the Real-time Transport Protocol .................................................. 31
   3.3.3 RTP Architecture ............................................................................................. 32
      3.3.3.1 RTP Data Packets ......................................................................................... 32
      3.3.3.2 RTCP (Control) Packets .............................................................................. 33
3.4 THE JAVA™ MEDIA FRAMEWORK REAL-TIME TRANSPORT PROTOCOL API 34
   3.4.1 JMF RTP Architecture ...................................................................................... 34
   3.4.2 RTP Formats ...................................................................................................... 36
3.5 THE REAL-TIME STREAMING PROTOCOL ......................................................... 37

CHAPTER 4: SYSTEM IMPLEMENTATION .................................................................... 38

4.1 SYSTEM ARCHITECTURE AND DESIGN CONSIDERATIONS ................................. 39
4.2 SYSTEM CAPABILITIES ......................................................................................... 40
   4.2.1 The Processing and Transmission of RTP Data .............................................. 41
   4.2.2 Reception and Presentation of RTP Data ......................................................... 42
   4.2.3 Multimedia Data Capture ............................................................................... 44
   4.2.4 Video Conferencing ....................................................................................... 45
4.3 POTENTIAL ENHANCEMENTS ............................................................................. 45
4.4 APPLICATION AREAS ......................................................................................... 46

CHAPTER 5: RESULTS AND SYSTEM EVALUATION .................................................. 48

5.1 EQUIPMENT AND NETWORK INFRASTRUCTURE ............................................. 49
5.2 VIDEO AND AUDIO CAPTURE - RESULTS ....................................................... 51
5.3 TRANSMISSION OF VIDEO DATA - RESULTS .................................................. 53
5.4 VIDEO CONFERENCING – PERFORMANCE RESULTS ....................................... 55
5.5 CONCLUSIONS ..................................................................................................... 56
# TABLE OF CONTENTS

## CHAPTER 6:
**USING ELINDA TO DEVELOP DISTRIBUTED MULTIMEDIA APPLICATIONS** .................................................. 57
6.1 **THE LINDA MODEL** ....................................................................................................................... 58
   6.1.1 **Basic Operations of Linda** ........................................................................................................ 58
   6.1.2 **Why Linda?** ............................................................................................................................. 59
6.2 **DEVELOPING AN ON-DEMAND MULTIMEDIA APPLICATION WITH eLINDA** .................. 59
6.3 **THE MULTIMEDIA COMPONENT OF eLINDA** .............................................................................. 62
6.3 **CONCLUSIONS** ............................................................................................................................. 64

## CHAPTER 7:
**SUMMARY AND CONCLUSIONS** ........................................................................................................ 65
7.1 **THESIS OVERVIEW** ....................................................................................................................... 66
   7.1.1 **Compression Techniques** ........................................................................................................ 66
   7.1.2 **Real-time Transport Protocol** .................................................................................................. 67
   7.1.3 **Distributed Processing using eLinda** ...................................................................................... 67
7.2 **THE FINAL PRODUCT** .................................................................................................................... 68
7.3 **FUTURE DIRECTIONS** ...................................................................................................................... 68

**REFERENCES** ........................................................................................................................................... 70

**APPENDIX A:**
**SYSTEM DOCUMENTATION** .............................................................................................................. I

**APPENDIX B:**
**GRAPHICS** ......................................................................................................................................... XX

**APPENDIX C:**
**PROJECT POSTER** .............................................................................................................................. XXII
LIST OF FIGURES

Figure 2.1: Block Diagram of JPEG Compression ............................................................. 16
Figure 2.2: Zigzag Sequence for Binary Encoding ............................................................. 17
Figure 3.1 High-level JMF Architecture ............................................................................ 28
Figure 3.2 RTP Data-packet Header Format ...................................................................... 32
Figure 3.3 High-level Architecture of the JMF RTP API .................................................. 35
Figure 3.4 Transmission of RTP Data via JMF RTP API .................................................. 35
Figure 3.5 Reception, Playback and Manipulation of RTP Data in the JMF RTP API .... 36
Figure 4.1 High Level Architecture of Multimedia Studio ............................................... 40
Figure 5.1 Network architecture, Department of Computer Science. .............................. 51
Figure 5.2 Comparison of Osprey-100 and Quickcam Transfer Performance with
    JPEG/RTP .............................................................................................................. 54
Figure 6.1 Simple Linda Operations. ................................................................................ 59
Figure 6.2 Mode of Operation of Video-on-Demand Application in eLinda. ................. 60
Figure 6.4 High Level Architecture of the Multimedia Component of the eLinda System...
    ........................................................................................................................... 62
Figure 6.5 Operation of Multimedia Support in eLinda.................................................... 63
LIST OF TABLES

Table 2.1: Characteristics of Common Video Formats ...................................................... 15
Table 2.2: Characteristics of Common Audio Formats ..................................................... 15
Table 2.3: Compression Ratios for JPEG Quality Factors ................................................. 17
Table 2.4: The H.323 Protocol Suite ................................................................................ 21
Table 3.1 Supported Formats in the JMF RTP API .......................................................... 37
Table 5.1 Attributes of Deployment Environment – Machine 1 ........................................ 50
Table 5.2 Attributes of Deployment Environment – Machine 2 ....................................... 50
Table 5.3 Video Capture Rates (measured in frames per second)................................... 52
Table 5.4 Characteristics of JMF Audio Capture ............................................................ 52
Table 5.5 Transmission Performance of Video Data Using Osprey-100 Video Capture Card at 20 fps ..................................................................................................... 53
Table 5.6 Transmission Performance of Video Data Using Logitech Quickcam at 20 fps .................................. .................................................................................................... 54
Table 5.7 Transmission Performance (fps) of Stored Multimedia Files ......................... 55
Table 5.7 Transmission Performance of Video Conferencing System Using JPEG_RTP Encoding Format .............................................................................................................. 56
ACKNOWLEDGEMENTS

This has been one of the most enjoyable and rewarding years of my university experience and there are a number of people who have contributed to the completion of this document and have made this year what it was. While it is impossible to mention each of them individually I would like to pay respect to those that have played a large part in helping me achieve success.

Firstly my supervisor, Dr. George Wells, thank you for agreeing to supervise this project and providing insight, assistance and suggestions for all facets of this reference, from its design and implementation to the deployment of the final product.

I would like to thank my “lab-mates” Kauna, Lindy, Mboneli and Melissa for making every second in the lab enjoyable and unforgettable.

I am extremely grateful to Aadil Patel, George Wells and Jill Wolvaardt for doing a comprehensive job of proofreading this thesis during this busy and stressful time of the year.

To my friends, near and far, especially Vim, Roops, Pedro, Harshal, Sanj, Krish, Nayan, Mel, Lindy and Reetal; thank you for moulding me into the person I have become. My thanks will never amount to the care and support that you have given me over the years.

I would like to pay special homage to my parents, my brother Viran, and my sister Bhavini, for giving me their ceaseless love, encouragement and motivation to strive for excellence in every task that I am faced with.

And most of all, I would like to give thanks to God for giving me the strength to pull myself up whenever life knocked me down. I am eternally grateful for everything that you have provided in my life. Jai Shri Ram.
Chapter 1:

INTRODUCTION
1.1 Background

Present day multimedia systems suffer from poor and inefficient handling of video data formats and the lack of sufficient bandwidth associated with the Internet. Another major problem often associated with current multimedia systems is the lack of available functionality for these applications.

The aim of this research project is to address these issues by investigating efficient ways of transmitting multimedia data across the World Wide Web. This project also aims to improve the functionality of current multimedia applications by using and extending Linda, a distributed and parallel programming language, for distributed multimedia applications. These improvements in functionality involve providing Linda with mechanisms for handling the various data structures that are used to process and transfer video and audio data across the Internet.

The primary development environment is the Java Media Framework (JMF) which is an Application Programming Interface (API) for incorporating time-based media into Java applications and applets. Research of Internet communication and multimedia presentation has lead to the increased functionality and efficiency of the Java Media Framework in the area of distributed multimedia applications. Specifically, this work has resulted in facilities that enable the use of video conferencing and the transfer of both “live” and stored video, with the Internet as a viable medium.
1.2 Current Technologies

1.2.1 Media Presentation

According to Sun Microsystems (1999), any data that changes meaningfully with respect to time can be characterized as *time-based media*. Audio clips, MIDI sequences, movie clips, and animations are common forms of time-based media.

Most time-based media is audio or video data that can be presented through output devices such as speakers and monitors. Such devices are the most common destination for multimedia data output. Media streams can also be sent to other destinations, for example, saved to a file or transmitted across the network. An output destination for media data is sometimes referred to as a data sink.

1.2.2 Presentation Controls

While a media stream is being presented, VCR-style presentation controls are often provided to enable the user to control playback. For example, a control panel for a movie player might offer buttons for stopping, starting, fast-forwarding, and rewinding the movie.

Many modern media players extend the media control panel by providing advanced facilities such as scrollable progress bars, visualisation options, volume control and on-line information about the current media being played.

1.2.3 Latency

In many cases when playing a media file, particularly when presenting a media stream that resides at a remote location on a network, the presentation of the media stream cannot begin immediately. The time taken before presentation can begin is referred to as the *start latency*. Users might experience this as a delay between the time the start button is clicked and the time when playback actually starts.
Multimedia presentations often combine several types of time-based media into a single, synchronized presentation. For example, background music might be played during an image slide-show, or animated text might be synchronized with an audio or video clip. When the presentation of multiple media streams is synchronized, it is essential to take into account the start latency of each stream - otherwise the playback of the different streams might actually begin at different times.

1.2.4 Presentation Quality

The quality of the presentation of a media stream depends on several factors, including:

1. The compression scheme used,
2. The processing capability of the playback system and
3. The bandwidth available (for media streams acquired over the network)

(Adapted from Sun Microsystems, 2001)

Traditionally, the higher the quality, the larger the file size and the greater the processing power and bandwidth required. Bandwidth is usually represented in terms of *bit rate* - the number of bits that are transmitted in a certain period of time.

To achieve high-quality video presentations, the number of frames displayed in each period of time, the *frame rate* (commonly referred to as the number of *frames-per-second*), should be as high as possible. Usually, video presentations at a frame rate of 30 frames-per-second are considered indistinguishable from TV broadcasts or analogue video tapes.

1.3 Challenges of on-Demand Media Presentation

*Video-on-Demand* is an umbrella term for a wide set of technologies whose goal is to enable individuals to select videos from a central server for viewing on a television or computer screen to start at any point in time specified by the user.

Current communications infrastructure and software architectures limit the functionality and efficiency of on-Demand multimedia services. This is due mainly to the fact that on-Demand
services require numerous resources in order to provide a “true” on-Demand multimedia solution.

1.3.1 Application Areas of on-Demand Multimedia Services

The scope and extent of on-Demand multimedia services is deep. Table 1.1 addresses some of the key application areas of Video-on-Demand systems and provides a short explanation of each of these areas.

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movies-on-Demand</td>
<td>Customers are able to select and play movies with full VCR capabilities.</td>
</tr>
<tr>
<td>Interactive Video Games</td>
<td>Customers are able to play downloadable computer games without having to buy a physical copy of the game.</td>
</tr>
<tr>
<td>Interactive News Television</td>
<td>Newscasts tailored to the customer’s taste enabling the customer to select and retrieve stories of personal interest.</td>
</tr>
<tr>
<td>Distance Learning</td>
<td>Customers have the ability to subscribe to courses being presented at remote locations.</td>
</tr>
<tr>
<td>Interactive Advertising</td>
<td>Customers can respond to advertiser surveys and are rewarded with free products and services.</td>
</tr>
<tr>
<td>Video Conferencing</td>
<td>Customers can communicate and interact visually from remote locations.</td>
</tr>
</tbody>
</table>

Table 1.1: Application Areas of on-Demand Video Services (Jarmo, 1995)

1.3.2 Requirements for a Successful Multimedia-on-Demand Implementation

Gee (2000) observes that in order to implement a successful Multimedia-on-Demand service certain interdependent issues need to be addressed. These include, but are not limited to,
**Transfer Media** - Inexpensive but efficient transfer media need to be implemented.

**Independence of Time** - The transfer medium must be able to customise order requests at any time to each client.

**Fast, Unimpeded Access** - The system must provide clients with fast response times for interactions.

**Quality** - The quality of transmitted video and audio must equal the quality of recorded media.

**Privacy and Security** - Various measures to ensure this must be implemented. These include:
- Service Access Control
- Network Access Security
- Protection of user data and information

Current multimedia implementations are unable to fulfill these objectives because of the lack of efficient and reliable telecommunications infrastructure and the existence of necessary bandwidth to ensure the quality of service required by such systems.

The aims of this investigation are to provide a suitable software design that will alleviate some of these issues through an innovative yet feasible architecture using current communications infrastructure.

### 1.4 Project Description and Goals

#### 1.4.1 Project Statement

This project specifically deals with the Java Media Framework implementation of the Real-time Transport Protocol. This implementation was tested for a variety of multimedia formats and application areas and can potentially result in its deployment for applications such as desktop video conferencing and video streaming.

In addition to the testing and implementation of the Java Media Framework Real-time Transport Protocol application programming interface, this paper also discusses the implementation of these facilities in various other application areas, specifically focusing on the Linda model of
distributed and parallel processing. This investigation has resulted in the implementation and deployment of a highly functional and efficient distributed multimedia application using the extended-Linda (eLinda) system developed by Wells (2001), thus enabling on-Demand multimedia services in an Internet-based environment.

1.4.2 Project Deliverables

Prior to the commencement of this investigation, the eLinda system already incorporated some simple multimedia facilities. The primary deliverable of this study is to build on and improve these facilities in terms of functionality and performance. The implementation and deployment of a pilot Video-on-Demand application, which utilises these extensions, will be presented as the final deliverable in this investigation.

Research collected about the various methods of transferring multimedia data across the Internet is also presented in this paper. A number of tests have been executed in order to perform this research and the results of these experiments are presented as a final deliverable in the form of performance measurements. In addition, a discussion about the architecture of the Java Media Framework API is provided in order to communicate insights of programming in the Java Media Framework to the reader.

1.5 Thesis Overview

Having addressed some of the background issues of developing on-demand multimedia systems in this chapter, Chapter 2 will discuss in detail the compression, transmission and presentation of multimedia data. Chapter 3 gives an overview of the technologies used to design and implement the applications developed in order to carry out this investigation. The discussion presented in this chapter includes a description of the Java Media Framework, the Real-time Transport Protocol and Java support for the Real-time Transport Protocol.

Chapter 4 of this thesis, entitled System Implementation, describes the design, implementation and functionality of the systems developed to perform the research. Results of the experimentation performed are presented in Chapter 5.
The application of the research performed in the area of distributed and parallel processing is introduced in Chapter 6 and a detailed discussion about the implementation of a typical Video-on-Demand system in this area is addressed. The architecture of the multimedia component of the eLinda system is also presented in this chapter. Finally, Chapter 7 draws this document to a close by presenting a summary and the conclusions of the research performed.
Chapter 2: Multimedia Encoding
When developing multimedia applications it is important to have a firm understanding of current technologies that enable the compression, transmission and presentation of that media.

This chapter addresses the various issues and available technologies involved with developing a multimedia application, including compression of multimedia data and common media formats. The major focus of this chapter is to address in detail the common video formats utilised by the Java Media Framework during the implementation of the software deliverables of this project.

The latter sections of this chapter present some key issues involved with the delivery and presentation of multimedia data from a development perspective.

2.1 Compression of Multimedia Data

The format in which multimedia data is stored is referred to as its content type. QuickTime, MPEG, and WAV are all examples of content types. Content type is essentially synonymous with file type. The term content type is used because multimedia data is often acquired from sources other than local files.

Most multimedia data undergoes some kind of compression in order to ensure fast and efficient retrieval and processing prior to being presented. All media content types fall under either one of two compression mechanisms: lossless media compression and lossy media compression.

2.1.1 Lossless Media Compression

Lossless compression of media ensures that all of the information of the original media clip is preserved in the encoded media file. Since the quality of the original clip is maintained, lossless compression is useful for final-cut editing or moving media clips between systems.
Preserving the original quality level of the media clip however, limits the degree to which the data transfer rate and file size can be minimised, and the resulting data rate for presentation may be too high for smooth playback on many systems (Adobe Systems, 2001). All lossless compression is based on the idea of breaking a file into a “smaller” form for transmission or storage, and then putting it back together so that it can be presented in its original form.

An example of a lossless compression media format is planar RGB.

2.1.2 Lossy Media Compression

A number of compression algorithms discard some of the original media information during compression. For example, if the pixels making up an image of the sky contain 70 shades of blue, a lossy compression algorithm, set for less-than-best quality, may record 60 shades of blue. Lossy compression algorithms usually let you specify how much of the picture quality the user may want to trade in order to lower the data processing rate and file size.

The extent of the information discarded is based on the “importance” of that region of the media stream. Wentworth (2001) observes that the importance of potentially discarded media information is based on human perception. Comparisons of lossy and lossless compression techniques are often subjective and qualitative.

Lossy compression allows much lower data rates and file sizes than lossless compression. Thus, lossy codecs are commonly used for the final production of multimedia delivered using CD-ROM or the Internet. However, lossy compression results in the permanent loss of the original quality of the media clip.

JPEG (Joint Photographic Expert Group), MPEG (Motion Picture Expert Group) and fractal-based compression methods are all examples of lossy compression techniques.
2.1.3 Spatial Compression

Spatial compression is a lossy technique that uses a different method to discard information from a media file. It works by deleting information that is common to the entire file or an entire sequence within the file. Spatial compression compacts the binary description of the visual area of a video frame by looking for patterns and repetition among pixels (Adobe Systems, 2001). For example, in a picture displaying a blue sky, spatial compression will note that many of the pixels are a similar shade of blue and instead of describing each of several thousand pixels, spatial compression can record a much shorter description, such as "All the pixels in this area are light blue."

Harris (2001) states that bit rate and file size are decreased as the level of compression is increased, but the picture loses its sharpness and definition. Run-length encoding is a version of this technique that is used by many media encoders and will be discussed further, later in this chapter.

2.1.4 Temporal Compression

Temporal compression is another lossy compression technique, but is time-based, rather than space-based. Temporal compression looks for ways to compact the description of the changes that take place during a sequence of frames. It does this by looking for patterns and repetition over time.

For example, in a video clip of a person speaking in front of a static background, temporal compression notices that the only pixels that change from frame to frame are those forming the face of the speaker. All the other pixels remain static (assuming the camera is still). Instead of describing every pixel in every frame, temporal compression describes all the pixels in the first frame, and for each frame that follows, describes only the pixels that are different from the previous frame. This technique is referred to as frame differencing. When most of the pixels in a frame are different from the previous frame, it is preferable to describe the entire frame again. Each whole frame is called a keyframe, which sets a new starting point for frame differencing,
and the consequent frame is called the \textit{delta-frame}. Many media encoders can create keyframes at an interval specified by the developer. Fewer keyframes result in a lower bit rate and file size.

\section*{2.2 Multimedia Codecs}

A codec performs the \textit{compression} and \textit{decompression} of multimedia data. Fowler (1997) observes that a codec can be either a software application or a hardware device that processes binary multimedia data through complex algorithms, which compress the file and then decompress it for playback. When a track is encoded, it is converted to a compressed format suitable for storage or transmission. When it is decoded, it is converted to a non-compressed (or \textit{raw}) format suitable for presentation.

Each codec has certain input formats that it can handle and a specified output format that it can generate. In some situations, a series of codecs might be used to convert from one format to another. Unlike other kinds of file-compression packages that require you to decompress a file before viewing, video codecs decompress the video as and when required, allowing the client to view the file from its compressed original. Codecs work in two ways: Spatial Compression and Temporal Compression and are categorised as either hardware codecs or software codecs.

\subsection*{2.2.1 Hardware Codecs}

Fowler (1997) states that hardware codecs are the most efficient way to compress and decompress video files. This is due to the fact that hardware codecs are faster and require fewer CPU resources than software codecs. Hardware codecs are expensive, but using a hardware-compression device will deliver high-quality source video footage. However, the use of hardware encoding requires viewers to have the same decompression device in order to decompress and present the selected media. Hardware codecs are used often in video conferencing, where the equipment of the audience and the broadcaster are configured in the same way and have also found application in the area of digital broadcasting where their efficiency becomes a necessity when processing multimedia data.
2.2.2 Software Codecs

Software codecs are less expensive than their hardware counterparts, and freeware versions are readily available. Most commercially available digital-video software products have built-in codecs that output files of relatively high quality. A major drawback to software codecs is that they are CPU-intensive since analysing and compressing media files is a time-consuming process. Freeware versions of commercial compression software packages are available online and provide facilities to access basic compression features, but do not, however, allow advanced control over these media clips.

2.3 Common Media Formats

Tables 2.1 and 2.2 identify some of the characteristics of common video and audio formats respectively. When selecting a format for processing, it is important to take into account the characteristics of the format, the target environment, and the expectations of the intended audience. For example, when delivering media content across the Internet, special attention needs to be paid to the bandwidth requirements of the selected media format in conformance to the resources available.

The CPU Requirements column characterises the processing power necessary for optimal presentation of the specified format. The Bandwidth Requirements column characterizes the transmission speeds necessary to send or receive data quickly enough for optimal presentation.

<table>
<thead>
<tr>
<th>Format</th>
<th>Content Type</th>
<th>Quality</th>
<th>CPU Requirements</th>
<th>Bandwidth Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinepak</td>
<td>AVI</td>
<td>Medium</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>MPEG-1</td>
<td>MPEG</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>H.261</td>
<td>AVI</td>
<td>Low</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>H.263</td>
<td>QuickTime</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>AVI</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2.1: Characteristics of Common Video Formats (Sun Microsystems, 1999)

<table>
<thead>
<tr>
<th>Format</th>
<th>Content Type</th>
<th>Quality</th>
<th>CPU Requirements</th>
<th>Bandwidth Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG</td>
<td>QuickTime AVI</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Indeo</td>
<td>QuickTime</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
</tbody>
</table>

### Table 2.2: Characteristics of Common Audio Formats (Sun Microsystems, 1999)

<table>
<thead>
<tr>
<th>Format</th>
<th>Content Type</th>
<th>Quality</th>
<th>CPU Requirements</th>
<th>Bandwidth Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCM</td>
<td>AVI</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Mu-Law</td>
<td>AVI</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>ADPCM (DVI, IMA4)</td>
<td>AVI</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>MPEG-1</td>
<td>MPEG</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>MPEG Layer3</td>
<td>MPEG</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>GSM</td>
<td>WAV</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>G.723.1</td>
<td>WAV</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
</tr>
</tbody>
</table>

Some formats are designed with particular applications and requirements in mind. High-quality, high-bandwidth formats are generally targeted toward CD-ROM or local storage applications. H.261 and H.263 are generally used for video conferencing applications and are optimized for video where there is little movement, for example in a corporate meeting. Similarly, G.723 is typically used to produce low bit-rate speech for telephony applications. The next section of this chapter will discuss the common media formats that have been used during the implementation of this project.
2.4 Common Video Compression Formats

2.4.1 JPEG (Joint Photographic Experts Group)

JPEG is a lossy compression technique for colour images. Although it can reduce file sizes to about 5% of their original size, some detail is lost in the compression process.

According to Wolfgang (1997), JPEG works by dividing up the image into 8 by 8 pixel blocks, and then calculating the discrete cosine transform (DCT)\(^1\) of each block. A quantizer rounds off the DCT coefficients according to the quantization matrix, the 8 x 8 matrix of step sizes. This step produces the "lossy" nature of JPEG, but allows for large compression ratios thereby achieving good bit rate and file size reduction. JPEG's compression technique uses a variable length code on these coefficients, and then writes the compressed data stream to an output file (*.jpg). For decompression, JPEG recovers the quantized DCT coefficients from the compressed data stream, takes the inverse transforms and displays the image.

![Block Diagram of JPEG Compression](Wolfgang, 1997)

Figure 2.1: Block Diagram of JPEG Compression (Wolfgang, 1997)

Figure 2.1 displays graphically the compression process of JPEG images. After quantization, it is not unusual for more than half of the DCT coefficients to be zero. JPEG incorporates \textit{run-length coding} to take advantage of this: for each non-zero DCT coefficient, JPEG records the number of zeros that preceded the number, the number of bits needed to represent the number's amplitude, and the amplitude itself. To consolidate the runs of zeros, JPEG processes DCT coefficients in the zigzag pattern shown in figure 2.2.

\[^1\] The Discrete Cosine Transform separates the image into sections of differing importance by transforming a signal or image from the frequency domain to the spatial domain.
After each block has been encoded, JPEG writes a unique end-of-block sequence to the output stream, and moves to the next block. When finished with all the blocks, JPEG writes the end-of-file marker.

Table 2.3: Compression Ratios for JPEG Quality Factors (Wentworth, 2001)

Table 2.3 shows the compression ratios of JPEG images of various sizes and quality. JPEG compression is normally used for still photographic images with true (24-bit) colour. JPEG is not used in situations where loss of quality is critical (for example, JPEG is not used for compressing X-rays).
2.4.2 MPEG (Moving Pictures Expert Group)

MPEG is a group of individuals that meet under the International Standards Organisation (ISO) to generate standards concerning compression techniques for digital video and audio streams for consumer distribution (Filippini, 1997). A number of different MPEG formats have arisen over time; these include MPEG-1, MPEG-2, MPEG-3 and MPEG-4. MPEG significantly reduces bit rates and file sizes by performing a combination of spatial and temporal compression.

2.4.2.1 MPEG-1

Originally, MPEG-1 was a low resolution video format that displayed video at 30 frames-per-second but presented near CD quality audio. According to Filippini (1997), the basic method used by MPEG-1 is to predict motion from frame-to-frame on the temporal axis and then applying Discrete Cosine Transforms (DCT) to organize the redundancy in the spatial plane. The motion prediction is done on 16x16 pixel blocks while the DCTs are applied to 8x8 pixel blocks, not dissimilar to that compression performed on JPEG as discussed in a previous section.

MPEG I contains three types of coded frames: Intra (I) frames, Predicted (P) frames and Bi-directional (B) frames. I-frames are coded as a still image and are not based on any prior data. P-frames are predicted from the most recently constructed I or P frame. B-frames are based on the two closest I or P frames; one in the past and one in the future.

Because of the block sizes of MPEG spatial compression, MPEG video is usually presented at resolutions of 352 x 240 pixels at 30 frames-per-second or at 352 x 288 pixels at 25 frames-per-second and the coded video rate is limited to approximately 1.8 Mega-bits per-second (Mbps). Because of the high bit rate, presentation of multimedia data is usually limited to such digital storage media as CD-ROM.

2.4.2.2 MPEG-2

MPEG-2 is a video standard based on the MPEG-1 format. MPEG-2 was specifically designed to support the features necessary to present High Definition Television (HDTV) and DVD-CD by specifying a coded bit stream for high quality digital video, thus supporting interlaced video formats and other advanced features. (Filippini, 1997)
The MPEG-2 profile was designed to support digital video transmission in the range of about 2 to 15 Mbps over cable, satellite and other broadcast channels. The original MPEG-2 profile extended the features of the original profile by defining a hierarchical or scalable profile. This profile aims to support applications such as compatible terrestrial television or HDTV, packet-network video systems, backward-compatibility with existing standards (MPEG-1 and H.261), and other applications for which multi-level coding is required. For example, such a system could give the consumer the option of using either a small portable receiver to decode standard definition TV, or a larger fixed receiver to decode HDTV from the same broadcast signal.

2.4.2.3 MPEG-3
Filippini (1997) observed that MPEG-3 no longer exists in a video-encoding context. It was initially intended to target HDTV applications with sampling dimensions up to 1920 x 1080 pixels at 30 frames-per-second with a limiting coded bit rate of between 20 and 40 Mbps. It was later discovered that by “tweaking” the MPEG-1 and MPEG-2 formats the above-mentioned targeted resolutions were easily attainable.

However, MPEG-3 audio encoding has achieved a great amount of success by achieving compression ratios of between 10:1 and 12:1 while still maintaining CD-quality sound (Fraunhoffer, 2001). The popular music format MP3 is based on the MPEG-3 specification.

2.4.2.4 MPEG-4
The next implementation of MPEG, MPEG-4, is still being developed (Filippini, 1997). MPEG-4 implementations are intended to span across a range of applications including multimedia, game audio, streaming audio and computer music. MPEG-4 appears to be the most efficient encoding standard developed to date, and will encode at incredibly high quality over the most constrained bandwidth requirements.

Streaming video today is mostly viewed on the Internet, but in future, streaming video will be far more prolific across multiple devices, appliances, platforms and computers (Marioni, 2001). Streaming Video will be seen in hotel lobbies, supermarkets, airplanes and on videophones, TVs and more. MPEG-4 appears to be the most efficient standard for future multimedia devices.
2.4.3 H.261 and H.263 Video Encoding Formats

Cherriman, Hanzo and Lucas (1996) observe that H.261 and H263 are programmable video transceivers that are commonly used for video conferencing. These encoding formats enable multimedia applications to adapt to a range of service requirements, video quality, bit rate and robustness.

Cherriman et al (1996) go on to state that H.261 and H.263 form the video encoding component of the H.323 Protocol Suite for real-time, interactive video conferencing and audio applications such as audio telephony. The H.261 codec transmits video images at 64 Kbps (VHS quality). This high bit-rate codec is appropriate for video over higher speed connections. The H.263 codec was built on the H.261 codec specification and supports the common interchange format (CIF), quarter common interchange format (QCIF), and sub-quarter common interchange format (SQCIF) picture formats. H.263 is well suited for Internet transmission over low-bit-rate connections, such as a 28.8 Kbps modem.

2.5 Common Audio Formats

This section aims to introduce the reader to some common audio formats used in the transmission of audio data across low bandwidth networks. It will mainly focus on the H.323 Protocol Suite without delving into in-depth technical information.
### 2.5.1 G.711 and G.723 Audio Encoding Formats

There are a numerous codecs available for the transmission of audio data over computer networks. The two most widely used encoding formats for this transmission are G.711 and G.723. (Penton, 2000)

G.711 and G.723 make use of a technique called Pulse Code Modulation (PCM). PCM is a process of transmitting analogue data in a digitized form. According to Harris (2001), the analogue signal amplitude is sampled (measured) at regular time intervals and converted to digital form for transmission. G.711 compressed audio requires either 56 or 64 Kbps (kilobits per second) and is thus not well suited to low bandwidth networks (Penton, 2000). G.723 compression offers a lower audio quality compared with that of the G.711 codec, but requires only 6.4 Kbps of bandwidth.

### 2.6 Media Presentation

This section of the chapter aims to introduce the reader to the pre-processing of multimedia data before presentation, transmission or manipulation of that multimedia data. The various issues and technologies used in the implementation of the project deliverables will be discussed in
greater detail in order to direct the reader to the relevant technologies associated with this project.

2.6.1 Multimedia Processing

In many instances, the data in a multimedia stream is manipulated before it is presented to the user. Generally, a series of processing operations occur before presentation:

- If the stream is multiplexed, the individual tracks are extracted.
- If the individual tracks are compressed, they are decoded.
- If necessary, the tracks are converted to a different format.
- If desired, effect filters are applied to the decoded tracks.
- The tracks are then delivered to the appropriate output device.

If the media stream is to be stored instead of rendered to an output device, the processing stages might differ slightly. For example, if you wanted to capture audio and video from a video camera, process the data, and save it to a file:

- The audio and video tracks would be captured.
- Effect filters would be applied to the raw tracks.
- The individual tracks would be encoded.
- The compressed tracks would be multiplexed into a single media stream.
- The multiplexed media stream would then be saved to a file.

(Sun Microsystems, 1999)

2.6.2 Media Capture

Time-based media can be captured from a live source for processing and playback. For example, audio can be captured from a microphone, and a video capture card can be used to obtain video from a camera. Capturing can be thought of as the input phase of the standard media processing model.
A capture device might deliver multiple media streams. For example, a video camera might deliver both audio and video. These streams might be captured and manipulated separately or combined into a single, multiplexed stream that contains both an audio track and a video track as discussed earlier.

2.6.3 Capture Devices

To capture time-based media you need specialised hardware. For example, to capture audio from a live source, you need a microphone and an appropriate audio card. Similarly, capturing a Television broadcast requires a TV tuner and an appropriate video capture card.

Capture devices can be characterized as either push or pull sources. For example, a still camera is a pull source; the user controls when to capture an image. A microphone is a push source; the live source continuously provides a stream of audio.

The format of a captured media stream depends on the processing performed by the capture device. Some devices do very little processing and deliver raw, uncompressed data. Other capture devices might deliver the data in a compressed format.

Capture controls are sometimes provided to enable the user to manage the capture process. For example, a capture control panel might enable the user to specify the data rate and encoding type for the captured stream and start and stop the capture process.

2.6.4 Demultiplexers and Multiplexers

A multiplexer is a device for combining several digital signals into an aggregate bit stream. Digital multiplexing may be implemented by interleaving bits, in rotation, from several digital bit streams either with or without the addition of extra framing, control, or error detection bits. Multiplexers enable audio and video data to be integrated into a single multimedia stream, presenting the illusion of a single source stream. (Streaming Media Magazine, 2001)
A demultiplexer extracts individual tracks of media data from a multiplexed media stream. A demultiplexer thus performs the opposite function of a multiplexer.

2.6.5 Compositing

Certain specialized devices support compositing. Compositing time-based media is the process of combining multiple tracks of data onto a single presentation medium. For example, overlaying text on a video presentation is one common form of compositing. Compositing can be done in either hardware or software. A device that performs compositing can be abstracted as a renderer that can receive multiple tracks of input data.

2.6.6 Streaming Media

A key characteristic of time-based media is that it requires timely delivery and processing. Once the flow of media data begins, there are strict timing deadlines that must be met, both in terms of receiving and presenting the data. For this reason, time-based media is often referred to as streaming media; it is delivered in a steady stream that must be received and processed within a particular timeframe to produce acceptable results. Streaming media technology enables the real-time distribution of audio, video and multimedia data on the Internet.

For example, when a movie is played, if the media data cannot be delivered quickly enough, there might be odd pauses and delays in playback. On the other hand, if the data cannot be received and processed quickly enough, the movie might appear jumpy as data is lost or frames are intentionally dropped in an attempt to maintain the proper playback rate.

Marioni, 2001 observes that streaming media technology enables the real-time distribution of audio, video and multimedia data on the Internet. Streamed data is transmitted by a server application and received and displayed in real-time by client applications. The server houses the media to be streamed (the content) and the technology to stream those files. The client can start displaying video or playing back audio as soon as enough data has been received and stored in the receiving station’s buffer. A streamed file is simultaneously downloaded and viewed on the
client machine with client player software such as Real Player® or Sonique®, but leaves behind no physical file on the viewer's machine.

An in-depth discussion of streaming technology specifics will be presented in following chapters.

2.6.8 Effect Filters

An effect filter modifies an audio or video track of a multimedia stream in some way, often to create special effects such as blur or echo (Sun Microsystems, 1999). Effect filters are classified as either pre-processing effects or post-processing effects, depending on whether they are applied before or after the codec processes the track. Typically, effect filters are applied to uncompressed (raw) data.
Chapter 3:

DEVELOPMENT ENVIRONMENT
When implementing a distributed multimedia application it is necessary for the developer to utilise a number of technologies and resources in order to successfully deploy the software solution so that it produces reliable results, in an efficient, easily accessible manner.

The aim of this chapter is to introduce the reader to some of the key technologies used in the implementation of this research project and to provide a valid justification of the choice of these technologies. Specifically, this chapter mentions the implementation language of choice (Java™) and goes on to discuss in detail the development environment and communication protocols used to perform the investigation.

### 3.1 Java™ Technology

According to Sun Microsystems, (2000), Java technology is:

- A programming language
- A development environment
- An application environment and,
- A deployment environment

Java technology provides the programmer with a wide range of development tools including a compiler, an interpreter, a document generator and a class packaging tool. The Java Runtime Environment (JRE) is a virtual machine environment that allows stand-alone applications to execute; most web browsers supply a Java interpreter and runtime environment.

The power of Java lies in its platform independence characteristics. Java applications can be executed on any Java compliant platform from PCs and Macintoshes to Personal Digital Assistants (PDAs) and Internet compatible cell phones. It is for these reasons and the distributed

---

1 Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.
computing nature of this research, that Java technology is the programming tool of choice. The use of Java technology ensures that future work resulting from the findings of this implementation will enable the commercial deployment and continued research of distributed multimedia applications across all previously mentioned platforms.

3.2 The Java™ Media Framework API

The Java Media Framework (JMF) is an Application Programming Interface (API) for incorporating time-based media into Java applications and applets. Audio clips, MIDI sequences, movie clips, and animations are common forms of time-based media.

The JMF 2.0 API extends the original media framework to provide support for capturing and storing media data, controlling the type of processing that is performed during playback, and performing custom processing on media data streams. In addition, JMF 2.0 and later versions, define a plug-in API that enables advanced developers and technology providers to more easily customize and extend JMF functionality.

A number of key classes encapsulated within the JMF package\(^2\) facilitate the above-mentioned functionality. The following sections discuss some of these classes and provide the reader with an insight into programming in the JMF.

\(^2\) The term ‘package’ refers to the JMF.jar file that aggregates the compiled JMF class libraries into a single file.
3.2.1 Managers

The JMF API contains numerous interfaces that define the behavior and interaction of objects used to capture, process, and present time-based media. Implementations of these interfaces operate within the structure of the framework. By using intermediary objects called managers, JMF makes it easy to integrate new implementations of key interfaces that can be used seamlessly with existing classes.

The JMF uses four types of managers:

- **Manager** - handles the construction of Players, Processors, DataSources, and DataSinks.
- **PackageManager** - maintains a registry of packages that contain JMF classes.
- **CaptureDeviceManager** - maintains a registry of available capture devices.
- **PlugInManager** - maintains a registry of available JMF plug-in processing components, such as Multiplexers, Demultiplexers, Codecs, Effects, and Renderers.

In order to write programs based on JMF, the developer needs to use the Manager’s create method to construct the players, processors, data sources, and data sinks for the application.

3.2.2 Data Sources

The `javax.media.DataSource` class in the JMF is able to encapsulate the information about the location of the media and the protocol and software used to deliver the given multimedia data. A data source is identified by either a JMF `MediaLocator` or a URL (universal resource locator). A media locator is similar to a URL and can be constructed from a URL, but can be constructed even if the corresponding protocol handler is not installed on the system.

3.2.3 Players

A `Player` processes an input stream of media data and renders it to the destination at a precise time. A `DataSource` is used to deliver the input media-stream to the player. The rendering
destination depends on the type of media being presented but a player cannot control the processing of the media stream being presented. During normal operation a JMF player steps through each of five states (Unrealized, Realizing, Realized, Prefetching, and Prefetched) until it reaches the Started state. The current state of the player defines what functionality the player may present at any point in time.

3.2.4 Processors

Processors, like players, can also be used to present multimedia data. A Processor is just a specialized type of Player that provides control over what processing is performed on the input media stream before presentation of that data (Sun Microsystems, 1999). The Processor class extends the state transition cycle of a Player by adding the configuring and configured states. The purpose of these additional states is to further refine the realising process. The state transition cycle of the Processor class follows the sequence: Unrealized, Configuring, Configured, Realizing, Realized, Prefetching, Prefetched, and Started.

In addition to rendering media data to presentation devices, a Processor can output media data through a data source so that it can be presented by another player or processor, further manipulated by another processor, or delivered to some other destination, such as a file.

The control classes provided by the JMF enable the processing and manipulation of multimedia data. This functionality has been used to facilitate the collection of data for various video and audio encoding formats and the presentation of multimedia data at various resolutions, frame rates and bit rates. The flexibility of changing multimedia data in this way has facilitated data collection and examination in an efficient and reliable manner. Details concerning these and other issues will be discussed further in Chapter 4, and the results arising from these experiments will be presented in Chapter 5.

3.3 Real-time Transport Protocol

The Real-time Transport Protocol (RTP) was the Internet transport protocol selected to facilitate the transmission of real-time multimedia streams of various formats across computer networks
(Shulzrinne, 1997). The following section discusses some of the important features of RTP and how the JMF RTP API facilitates these features.

3.3.1 Overview of the Real-time Transport Protocol

The Real-time Transport Protocol is the Internet-standard protocol for the transport of real-time data, including audio and video (Shulzrinne, 1997). RTP provides the end-to-end, protocol and network-independent delivery of real-time data and comprises two major components – a data component and a control component, known as the Real-time Control Protocol (RTCP). RTP has a variety of application areas including video and audio streaming and multimedia-on-demand as well as interactive services such as Internet telephony and video conferencing.

3.3.2 Features of the Real-time Transport Protocol

The RTP specification stipulates a number of features that characterise RTP as a protocol suitable for the unicast or multicast delivery of real-time data across a network. In addition, RTP enables developers to:

- Identify the type of data being transmitted,
- Determine what order the packets of data should be presented in, and
- Synchronize media streams from different sources.

(Sun Microsystems, 1999)

RTP data packets are not guaranteed to arrive in the order that they were sent. RTP is described as an unreliable protocol as it does not guarantee the arrival of RTP packets transmitted from the transmission source. However, the receiver of the RTP packets is able to reconstruct the packet sequence and detect lost packets from information embedded within the header of every RTP packet transmitted.

While the RTP specification does not provide any mechanism to ensure the timely delivery of RTP packets or provide other quality of service guarantees, the Real-time Control Protocol

---

3 Unreliable protocols require less processing overhead than reliable protocols as data packets do not need to be re-sent. Re-transmitted packets are useless in real-time systems. (Penton, 2000)
(RTCP) augments RTP and enables developers to monitor the quality of the data delivery. If quality of service is essential for a particular application, RTP can be implemented over a resource reservation protocol that provides connection-oriented services.

3.3.3 RTP Architecture

According to Sun Microsystems, Inc. (2001), an RTP session is an association among a set of applications communicating via RTP. A session is characterised by a network address and a pair of ports. One of these ports is used for the delivery of the multimedia data and the other is used for control (RTCP) data. A participant is a single machine, host, or user participating in a RTP session. Participation in a session may involve passive reception of data, active transmission of data or both.

3.3.3.1 RTP Data Packets

The multimedia data for a RTP session is transmitted as a series of packets and is sometimes referred to as an RTP stream. Every RTP data packet in a stream contains two parts, a structured header and the actual RTP (real-time) data, also known as the packet's payload.

![RTP Data-packet Header Format](image)

Figure 3.2 RTP Data-packet Header Format (Sun Microsystems, 2001)

The RTP packet header contains information including:

- The RTP Version Number (V)
- Padding (P) to supplement the RTP Packet
- Extension (X), if the X bit is set then the RTP header is followed by a user-defined extension to the current RTP Packet header
- CSRC Count (CC), this number defines the number of CSRC identifiers that follow the fixed header
Marker (M), a marker bit defined by the particular media profile

Payload Type (PT), this bit sequence specifies the index into a media profile table that describes the payload format. The payload mappings for audio and video are specified in RFC 1890.

Sequence Number defines a unique packet number that identifies the current packet's position in the sequence of packets. The packet number is incremented by one for each packet sent.

Timestamp reflects the sampling instant of the first byte in the payload.

SSRC (Synchronisation Source) identifies the synchronisation source.

CSRC (Content Source [32 bits each]) identifies the contributing sources for the payload.

(Sun Microsystems, 1999)

3.3.3.2 RTCP (Control) Packets

In addition to the media data for a particular RTP session, control data (RTCP) packets are sent periodically to all the participants in that session. RTCP packets can contain information about the quality of service (QOS) for the session participants, information about the source of the media being transmitted on the data port, and statistics pertaining to the data that has been transmitted so far.

There are several types of RTCP packets. These include:

- Sender Reports
- Receiver Reports
- Source Description
- Bye, and
- Application-specific Packets.

RTCP packets can be sent as compound packets containing at least two packets, a report packet and a source description packet. All compound RTCP packets must include a source description (SDES) element that contains the canonical name (CNAME) that identifies the source of the RTP data. Additional information might be included in the source description, such as the source's name, email address, phone number, geographic location, application name, or a message describing the current state of the source.
A participant that has recently sent data packets issues a *sender report*. The sender report (SR) contains the total number of packets and bytes sent as well as information which can be used to synchronise media streams from different sessions. Session participants periodically issue *receiver reports* for all of the sources from which they are receiving data packets. A receiver report (RR) contains information about the number of packets lost, the highest sequence number received, and a timestamp that can be used to estimate the round-trip delay between a sender and the receiver. When a source is no longer active, it sends an RTCP BYE packet. (Sun Microsystems, 1999)

RTCP APP (Application) packets enable developers to define and send custom information through the RTP control port.

### 3.4 The Java™ Media Framework Real-time Transport Protocol API

The classes encapsulated within the `javax.media.rtp`, `javax.media.rtp.event`, and `javax.media.rtp.rtcp` packages provide support for the Real-Time Transport Protocol. The first version of the JMF RTP API enabled developers to receive RTP streams and play them using the Java Media Framework. In JMF 2.0, the RTP API also supports the *transmission* of RTP streams.

#### 3.4.1 JMF RTP Architecture

The JMF RTP API is designed to work seamlessly with the capture, presentation, and processing capabilities of the JMF. Players and processors are used to present and manipulate RTP media streams, as they are used in the manipulation of any other media content in the JMF. Figure 3.2 displays the integrated JMF and JMF RTP architectures.
The JMF RTP API allows developers to transmit media streams that have been captured from a local capture device or media that has been stored to a file. Figure 3.3 shows the transmission and storage to file of multimedia data via RTP in the Java Media Framework.

Figure 3.4 displays the reception of RTP data. This diagram shows not only how data is received by the RTP API, but also how processing and media manipulation may be performed on the received multimedia data.
3.4.2 RTP Formats

Table 3.2 displays the multimedia encoding formats supported by the JMF RTP application programming interface. R indicates that codecs exist to receive, decode and present the given format. T indicates that the media stream can be encoded and transmitted via the Java RTP implementation.

<table>
<thead>
<tr>
<th>Media Type</th>
<th>RTP Payload</th>
<th>JMF 2.1.1 Windows Performance Pack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio: G.711 (U-law) 8 kHz</td>
<td>0</td>
<td>R, T</td>
</tr>
<tr>
<td>Audio: GSM mono</td>
<td>3</td>
<td>R, T</td>
</tr>
<tr>
<td>Audio: G.723 mono</td>
<td>4</td>
<td>R, T</td>
</tr>
<tr>
<td>Audio: 4-bit mono DVI 8 kHz</td>
<td>5</td>
<td>R, T</td>
</tr>
<tr>
<td>Audio: 4-bit mono DVI 11.025 kHz</td>
<td>16</td>
<td>R, T</td>
</tr>
<tr>
<td>Audio: 4-bit mono DVI 22.05 kHz</td>
<td>17</td>
<td>R, T</td>
</tr>
<tr>
<td>Audio: MPEG Layer I, II, III</td>
<td>14</td>
<td>R, T</td>
</tr>
<tr>
<td>Video: JPEG (411, 422, 111) (^4)</td>
<td>26</td>
<td>R, T</td>
</tr>
<tr>
<td>Video: H.261</td>
<td>31</td>
<td>R</td>
</tr>
</tbody>
</table>

---

\(^4\) JPEG/RTP can only be transmitted in video dimensions that are in multiples of 8 pixels.
3.5 The Real-time Streaming Protocol

As is the Real-time Transport Protocol (RTP), the Real Time Streaming Protocol, or RTSP, is an application-level protocol for control over the delivery of data with real-time properties (Shulzinne, Rao and Lanphier, 1998). RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. Sources of data can include both live data feeds and stored media clips.

RTSP is a protocol intended to control multiple data delivery sessions, provide a means for choosing delivery channels such as UDP, multicast UDP and TCP, and provide a means for choosing delivery mechanisms based upon RTP. (Shulzinne et al, 1998)

The Real-Time Streaming Protocol establishes and controls either a single stream or several time-synchronized streams of continuous media such as audio and video. RTSP differs from RTP in that RTSP does not deliver the continuous streams, even though interleaving the continuous media stream with the control stream is possible. As a simple analogy, RTSP acts as a "network remote control" for multimedia servers.

Although beyond the scope of this project, RTSP could be used in future work to enhance the functionality of multimedia-on-demand systems by providing resources with which to remotely control a media stream. This will allow clients to ‘fast-forward’, ‘rewind’, ‘pause’ or ‘slow-motion’ the streaming media remotely, thus overcoming one of the key functional areas of concern when implementing a multimedia-on-demand system.

Table 3.1 Supported Formats in the JMF RTP API. (Sun Microsystems, 1999)

<table>
<thead>
<tr>
<th>Format</th>
<th>Value</th>
<th>Codec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video: H.263</td>
<td>34</td>
<td>R, T</td>
</tr>
<tr>
<td>Video: MPEG-1</td>
<td>32</td>
<td>R, T</td>
</tr>
</tbody>
</table>

5 H.263/RTP can only be transmitted in 3 different video dimensions: SQCIF (128x96), QCIF (176x144) and CIF (352x288). Both RPC 2190 (PT 34) and 2429 (dynamic payload) payload formats are supported.

6 MPEG/RTP video can only be transmitted from pre-encoded MPEG content, i.e. from an MPEG-encoded file or MPEG enabled capture source. Real-time software MPEG encoding is not feasible for RTP transmission.
Chapter 4: System Implementation
The tools and technologies discussed in the previous chapters have facilitated the implementation of a number of functional and efficient distributed multimedia applications. These applications have been integrated to produce a multimedia system capable of performing a variety of media processing and presentation functions.

The aim of this chapter is to enlighten the reader about the capabilities, functionality and application of tools such as the Real-time Transport Protocol (RTP), the Java Media Framework (JMF) and other facilities discussed in the previous chapter, in implementing a distributed multimedia application.

The early sections of this chapter address the design and architecture of the system implemented, aptly named *Multimedia Studio*, and relates these design considerations to issues previously discussed. The functionality and the techniques used to achieve this functionality will be discussed in some detail before addressing some of the commercial and academic application areas of systems such as these. An analysis of results achieved and data collected will subsequently be presented in the following chapter.

### 4.1 System Architecture and Design Considerations

Multimedia Studio was designed with the intention of exploiting all the facilities available to the Java Media Framework API with respect to the manipulation and transportation of multimedia data. With these issues in mind, Multimedia Studio follows a component based architecture where the major components reflect the functionality that they address.

Therefore, modules concerning the reception, transmission and capture of multimedia data form the base of the entire system while method calls from higher up in the system hierarchy invoke the functionality of these components.
Figure 4.1 displays the high-level system architecture of Multimedia Studio.

![Multimedia Studio Diagram](image)

The video conferencing system embedded within the overall Multimedia Studio system comprises “server” and “client” objects that, after negotiating communication channels, make audio and video conferencing at the desktop level possible. Reception and playback of RTP data is facilitated by a separate thread of execution for receiving RTP data from a user defined location and port. The transmission of multimedia data involves the user selecting a media file for transmission and specifying a remote location for streaming. A detailed discussion concerning the implementation of these features will be provided in the next section.

4.2 System Capabilities

By using the facilities provided by the Java Media Framework it is possible to implement the features previously discussed. This section addresses in detail the implementation of these features and aims to introduce the reader to issues involved when programming a distributed multimedia system.

Initially, the processes of preparing a multimedia data source for transmission and the transmission of that multimedia stream is addressed. The steps involved with receiving and presenting this streamed data is then discussed, followed by a section describing the capture of “live” multimedia data within the JMF API. Finally, these processes are combined to produce an explanation of the video conferencing component of Multimedia Studio.
4.2.1 The Processing and Transmission of RTP Data

The transmission of RTP data from a multimedia source comprises several distinct stages that involve the retrieval, processing and transmission of the multimedia data. Creating a `DataSource` object from a given multimedia file simply entails invoking the `createDataSource` method from the `javax.media.Manager` class with a given path or `MediaLocator` object:

```java
DataSource myDS;
myDS = Manager.createDataSource(new MediaLocator("file://video.mov"));
```

In order to manipulate the media data and make it suitable for transmission a processor needs to be created from the current data source. “Suitable for transmission” implies that the multimedia intended for transmission needs to be encoded in a format conducive to its efficient streaming across the network.

```java
Processor myProcessor = Manager.createProcessor(myDS);
```

Before calling on the functionality of the processor class, it is necessary to wait for the processor to reach the `configured` state. Once configured, the processor is able to retrieve and manipulate the `Controls` of the current multimedia data source. This process simply involves a call to retrieve the desired multimedia control object and setting it to a user defined value. For example: setting the frame rate to a user defined value:

```java
FrameRateControl frc;
frc = myProcessor.getControl("javax.media.control.FrameRateControl");
frc.setFrameRate(newFrameRate);
```

A similar process is followed to manipulate the various other controls including the bit rate control, the media track controls, the quality control, the packet size control, and so on. The output of the processor can be fed to a newly created data source by capturing the output of the processor:

```java
DataSource dataOutput = myProcessor.getDataOutput();
```

---

8 See Section 3.2 of this document for an explanation all the Java Media Classes discussed in this chapter.
9 The state transition cycle of the Processor class is discussed in Section 3.2.4 of this document.
After processing the selected media file and retrieving the data output of the processor, the next step to transmitting the processed data is to create and set-up the network connections. The transmission of multimedia data via RTP involves the creation of an RTP Manager object for each transmitted track of the multimedia data source to control the transmission stream.

```java
PushBufferDataSource pbds = (PushBufferDataSource)dataOutput;
PushBufferStream [] pbss = pbds.getStreams();
RTPManager [] rtpmgrs = new RTPManager[pbss.length];

The client machine is added as a target to each RTP manager, and SourceStream objects are then used to connect to the local data source. Opening the streaming media channel for data transmission is consequently started by a call to the SourceStream.start() method.

```r
rtpmgrs[i].initialize(localAddr);
rtpmgrs[i].addTarget(destAddr);
```

```java
SourceStream sendStream = rtpmgrs[i].createSendStream(dataOutput, i);
sendStream.start();
```

Supplying the source stream object with data to transmit merely involves the request:

```java
myProcessor.start();
```

This sequence of steps facilitates the efficient transmission of stored multimedia data to a remote location via the Real-time Transport Protocol. By implementing minor adjustments to the media locator object, developers can easily transport captured video and audio data across the computer network.

4.2.2 Reception and Presentation of RTP Data

As with the transmission of RTP data, receiving RTP data involves creating an RTP Manager object to handle each incoming media stream from the source. The ReceiveStreamListener and SessionListener interfaces are implemented by the receiving object in order to detect and control incoming RTP media streams. The process of establishing “listeners” and waiting for RTP data to arrive follows the coding process below.

Create the RTP managers:

```java
Manager [] manager = new RTPManager[numStreams]
```
Add the session listener and receive stream listeners to the current object:

```java
manager[i].addSessionListener(this);
manager[i].addReceiveStreamListener(this);
```

Initialise the manager and listen for RTP data from a particular server:

```java
manager[i].initialise(localAddress);
manager[i].addTarget(destAddress);
```

Once these connections have been established the client program waits until a `ReceiveStreamEvent` is generated by the RTP Manager listening to a particular server on a particular port. If the stream event is an instance of a `NewReceiveStreamEvent`, it is possible to retrieve the media stream from the stream event and subsequently retrieve the data source from the media stream.

```java
ReceiveStream stream;
DataSource ds = stream.getDataSource();
```

Now that the data source has been extracted from the incoming media stream, it is necessary to present the media stream to the user of the system. This is achieved by creating a multimedia player object from the transmitted data source which is able to decode and visually display the streaming media.

```java
Player mmPlayer = javax.media.Manager.createPlayer(ds);
```

Once realised, the visual component of the player can be added to a Java frame object and is subsequently played.

```java
mmPlayer.realize();
myFrame.add(mmPlayer.getVisualComponent());
mmPlayer.start();
```

The above sequence of steps represents a basic outline of the steps required in simply receiving and playing back a streamed multimedia object. There is greater functionality and flexibility in the complete Multimedia Studio package than is shown here. Of course, in systems such as these, there exists a high potential for enhancements to the usability, efficiency and flexibility in the playback and retrieval of multimedia streams. Complete listings of these classes are available in Appendix B for a comprehensive review of system integrity and full functionality.
4.2.3 Multimedia Data Capture

Multimedia Studio facilitates the capture, presentation and transmission of multimedia data. Capture devices, such as cameras and microphones, are referenced through the `CaptureDeviceManager` which acts as a central registry for all the capture devices installed on the host machine and are available to the Java Media Framework. Each connected capture device is represented by an instance of the `CaptureDeviceInfo` class. This object encapsulates all the information associated with a particular capture device, including all the available video or audio formats associated with that device.

Several steps are involved when capturing and displaying a “live” multimedia stream. The first step involves querying the capture device registry for the desired capture device. Once the device has been found the media locator for the device can be used to create a data source which is subsequently used to create a player or processor as discussed in the previous section.

```java
CaptureDeviceInfo videoCDI;
videoCDI = CaptureDeviceManager.getDevice("deviceName");
MediaLocator mediaLocator = videoCDI.getLocator();
DataSource captureDS = Manager.createDataSource(mediaLocator);
Player livePlayer = Manager.createPlayer(captureDS);
```

When capturing audio and video data simultaneously, the developer is required to create a `MergingDataSource` from a list of 2 or more data sources.

```java
DataSource captureDS;
DataSource [] captureSources = {videoDS, audioDS};
captureDS = Manager.createMergingDataSource(captureSources);
```

Creating a player with the captured data source only facilitates rendering of the captured media to the output devices. In order to manipulate the captured media it is necessary to create an instance of the `Processor` class from the data source as discussed in Section 4.2.1.
4.2.4 Video Conferencing

The video conferencing system developed as a component of Multimedia Studio comprises a series of calls to the previously mentioned receive, transmission and capture components of the system. Of the two participants in a video conferencing session one performs the duties of a server by providing the “client” with instructions about which communication channels to utilise. To facilitate this, an initial communication between the client and the server takes place in which the server waits for a request from the client. Once the request is received by the server, an initial negotiation takes place to determine the communication channels required to facilitate the session. The server then provides the client with all the information required to establish and maintain the conferencing session. Details concerning this transaction and the operations of the entire Multimedia Studio system are presented in Appendix A of this document.

Once these preliminary interactions have been completed and the channels have been established, both the client and the server begin reception and transmission of the captured data.

4.3 Potential Enhancements

Currently Multimedia Studio presents functionality capable of creating and maintaining a single desktop video conference and all associated capabilities such as transmitting and receiving RTP streams, playing media files and capturing “live” multimedia data. At the development level, the system is capable of manipulating multimedia data in order to adapt it to a format suitable for what it is intended: either playback or transmission.

An enhancement to the system would be to bring this functionality to the user interface so that the user may specify a personal preference for the playback or transmission format of the multimedia data. This would entail presenting the user with a selection of bit rates, frame rates, required packet sizes and, of course, presentation format (JPEG, MPEG, G.723, and so on).10

Some current, commercially available video conferencing systems allow users to record a video conference for review at some later point in time. This option may be facilitated in the JMF by allowing the user to write the incoming media stream from a conference to a random access file.

10 Tables 2.1 and 2.2 present all JMF-compatible video and audio encoding formats.
on the host machine. By creating a processor from the inward bound RTP stream and recording its output to a data sink, the functionality of recording a conference session would easily be achieved.

```java
DataSink sink;
MediaLocator destination = new MediaLocator("file://myMeeting.mov");

sink = Manager.createDataSink(processor.getDataOutput(), destination);
sink.open();
sink.start();
```

A similar sequence of steps is performed when capturing “live” multimedia data from a camera or a microphone to a file.

The `RemotePayloadExchangeEvent` class, an extension of the `RemoteStreamEvent` class, identifies format changes at the transmission end of an RTP session and is able to handle the change. This implies that a video conference participant is able to change the transmission format, for example a change from JPEG to MPEG, and the associated conference participant is able to detect and accommodate those changes without interrupting the conference session. By implementing this enhancement users would be able to change transmission and presentation formats according to the performance of the session and the resources of the participants.

### 4.4 Application Areas

The technologies and facilities utilised by Multimedia Studio address a number of application areas and capabilities. The streaming of multimedia data has application in Multimedia-on-Demand services, Internet telephony, video over IP and interactive conferencing services.

Applications addressing these areas can be combined with the power of Java technology in implementing distributed applications. By employing tools such as Java applets, developers are able to providing interactive, Internet based multimedia solutions, and are able to enhance the functionality and efficiency of these systems by employing transportation protocols such as RTP and RTSP that are specifically designed for such systems.
The power of streaming multimedia lies in its application in areas such as on-line education, video-on-demand and video conferencing, all using the restrictive bandwidth of the Internet as a viable medium. The next generation of these applications can thus be applied to current wireless networks, and interactive video communication via hand-held devices becomes a not too distant dream of the future.
Chapter 5:

RESULTS AND SYSTEM EVALUATION
Chapter 4 of this document described the features, facilities and application areas of the distributed multimedia systems that have been developed for this research exercise. However, the performance and efficiency of systems of this nature need to be determined in order to evaluate the feasibility of the given system, and its ability to be implemented in a desired application area.

The results of the testing performed on applications developed are presented in this chapter. A qualitative discussion accompanies the measurements presented in order to portray to the reader the feasibility of implementing any given attribute of the system.

5.1 Equipment and Network Infrastructure

The performance of distributed multimedia applications is severely dependant on processing power, memory, and the bandwidth of the network. This section presents the equipment used to develop and test the software implemented.

<table>
<thead>
<tr>
<th><strong>Machine 1</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Screen</strong></td>
<td>15&quot; Proline CrystalView SVGA</td>
</tr>
<tr>
<td><strong>Processor</strong></td>
<td>Intel Pentium II 500MHz</td>
</tr>
<tr>
<td><strong>Random Access Memory</strong></td>
<td>192 MB</td>
</tr>
<tr>
<td><strong>Capture Device (Visual)</strong></td>
<td>Osprey-100 Video Capture Card with Standard web cam.</td>
</tr>
<tr>
<td><strong>Video Memory</strong></td>
<td>86 Kb on board (Osprey-100) Video RAM</td>
</tr>
<tr>
<td><strong>Sound Card (with speakers and microphone)</strong></td>
<td>Creative SoundBlaster SBS20</td>
</tr>
<tr>
<td><strong>Java SDK</strong></td>
<td>Version 1.3</td>
</tr>
<tr>
<td><strong>JMF API</strong></td>
<td>Version 2.1.1</td>
</tr>
</tbody>
</table>
Table 5.1  Attributes of Deployment Environment – Machine 1

Machine 1 was the primary computer used to compile and deploy the application. It is a standard, Rhodes University, Department of Computer Science, honours machine with 128 megabytes of extra RAM, used with the intention of increasing system performance when handling the processing of multimedia data.

Table 5.2  Attributes of Deployment Environment – Machine 2

Machine 2 was used when testing the transmission and reception of multimedia data across the network.

The performance of these machines with respect to the capture, transmission and presentation of multimedia data was severely restricted by the insufficient amount of installed RAM. Testing of JMF-based multimedia applications is normally performed with a minimum of 256 MB RAM (Sun Microsystems, 2001).

The network architecture of the Rhodes University, Department of Computer Science is composed of 2 major segments; the Calnet network and the Struben network. Figure 5.2 portrays this architecture visually. Machines 1 and 2 are located in the Calnet Segment of the
overall network. This means that transmission of multimedia data was executed on an available bandwidth of about 10 Megabits per second (Mbps). However, Penton (2000) records that the amount of available bandwidth on this segment is about 4Mbps due to the restrictions of shared Ethernet.

![Network Architecture](image)

Figure 5.1 Network architecture, Department of Computer Science. Adapted from Penton (2000)

### 5.2 Video and Audio Capture - Results

Two different video capture devices were used while conducting this experiment. The Logitech Quickcam Express was attached to Machine 2 via the USB port. The Osprey-100 video capture card was installed directly on the motherboard of Machine 1. Table 5.3 shows the maximum video capture rates\(^\text{11}\) at each of the resolutions tested.

---

\(^{11}\) Video capture rates were measured by the number of frames per second (fps) rendered to the screen.
CHAPTER 5 - Results and System Evaluation

### Video Capture Rates (Frames per second)

<table>
<thead>
<tr>
<th>Resolution:</th>
<th>Osprey-100</th>
<th>Logitech Quickcam</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 x 60</td>
<td>30.0</td>
<td>Not Supported</td>
</tr>
<tr>
<td>160 x 120</td>
<td>30.0</td>
<td>29.4</td>
</tr>
<tr>
<td>176 x 144</td>
<td>30.0</td>
<td>28.4</td>
</tr>
<tr>
<td>320 x 240</td>
<td>29.9</td>
<td>14.9</td>
</tr>
<tr>
<td>352 x 288</td>
<td>Not Supported</td>
<td>12.9</td>
</tr>
</tbody>
</table>

Table 5.3 Video Capture Rates (measured in frames per second).

The video data was captured and displayed in RGB (Red, Green, Blue) format using 16 bits per pixel colour. The resolutions chosen for display are those resolutions that are supported by the JMF for encoding and transmission. The Osprey-100 video capture card displayed video at a superior rate to that of the Logitech, producing a smooth sharp image. However, the Logitech QuickCam presented an image with deeper, truer colours. Due to the size of the image, higher resolutions for the Logitech QuickCam proved to be unfeasible for transmission across the network.

When using the Java Media Framework for performing audio capture, two sources are available to the developer; the JavaSound audio capture renderer and direct sound capture by the sound card. A comprehensive investigation concerning audio processing is beyond the scope of this paper. Table 5.4, however, introduces the reader to the capture capabilities of audio in the JMF.

### Characteristics of JMF Audio Capture

<table>
<thead>
<tr>
<th></th>
<th>JavaSound Audio Capture</th>
<th>Direct Sound Capture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding</td>
<td>Linear</td>
<td>Linear</td>
</tr>
<tr>
<td>Sample Rate</td>
<td>44100.0Hz</td>
<td>48000.0Hz</td>
</tr>
<tr>
<td>Bits per sample</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Channels</td>
<td>Stereo</td>
<td>Stereo</td>
</tr>
</tbody>
</table>

Table 5.4 Characteristics of JMF Audio Capture.
5.3 Transmission of Video Data - Results

Multimedia Studio has the ability to transmit stored video files and live video data via the Real-time Transport Protocol so that it is received and presented to the “client” at the other side of the network. The JMF RTP API supports the streaming of video data of various formats in various resolutions. Before transmission of this data, video sizes are checked for conformance to the JMF standard.

For JPEG, make sure width and height are divisible by 8.

\[
\begin{align*}
\text{width} &= (\text{size.width} \mod 8 == 0 \ ? \ \text{size.width} : \ \lfloor \text{size.width} / 8 \rfloor \times 8); \\
\text{height} &= (\text{size.height} \mod 8 == 0 \ ? \ \text{size.height} : \ \lfloor \text{size.height} / 8 \rfloor \times 8);
\end{align*}
\]

Only three sizes, measured in pixels, are supported by the JMF RTP H.263 format; these are: 128 x 96, 176 x 144 and 352 x 288. The transmission of video data was tested for the above-mentioned formats and resolutions.

<table>
<thead>
<tr>
<th>Resolution (Pixels):</th>
<th>Compression Format</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JPEG/RTP</td>
</tr>
<tr>
<td>128 x 96</td>
<td>19.9</td>
</tr>
<tr>
<td>176 x 144</td>
<td>19.9</td>
</tr>
<tr>
<td>320 x 240</td>
<td>14.9</td>
</tr>
<tr>
<td>352 x 288</td>
<td>8.4</td>
</tr>
</tbody>
</table>

Table 5.5 Transmission Performance of Video Data Using Osprey-100 Video Capture Card at 20 fps
CHAPTER 5 - Results and System Evaluation

<table>
<thead>
<tr>
<th>Resolution (Pixels):</th>
<th>Compression Format</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JPEG/RTP</td>
<td>H.263/RTP</td>
</tr>
<tr>
<td>128 x 96</td>
<td>16.9</td>
<td>15.4</td>
</tr>
<tr>
<td>176 x 144</td>
<td>11.9</td>
<td>9.9</td>
</tr>
<tr>
<td>320 x 240</td>
<td>6.4</td>
<td>Not Supported</td>
</tr>
<tr>
<td>352 x 288</td>
<td>3.9</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 5.6 Transmission Performance of Video Data Using Logitech Quickcam at 20 fps

Tables 5.3, 5.5 and 5.6 show that the Osprey-100 video capture card is far superior to the Logitech QuickCam when capturing video data and converting that data to a format suitable for transmission. A visual comparison of the data transfer rates accomplished by the Osprey-100 and Logitech Quickcam is provided in Figure 5.2.

![Comparison of Osprey-100 and Quickcam Transfer Performance](image)

Video data received at a rate greater that 15.0 fps provides a smooth, detailed image on the clients’ screen. The above performance was achieved by manipulating the captured data by reducing packet sizes, lowering the capture frame rate, reducing the bit rate and reducing the quality to achieve faster data transfer rates as previously discussed in Chapter 4.
The streaming of “stored” multimedia files in this environment is performed relatively efficiently and provides a smooth and close-to-original quality of the multimedia data source. Testing of video streaming was performed on numerous sample multimedia files of differing formats and resolutions. Some of the results achieved are presented in Table 5.7.

<table>
<thead>
<tr>
<th>Compression Format</th>
<th>Transmission Format</th>
<th>Resolution (Pixels)</th>
<th>Frame Rate (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVI</td>
<td>JPEG_RTP</td>
<td>320 x 240</td>
<td>22.0</td>
</tr>
<tr>
<td>MOV</td>
<td>H263_RTP</td>
<td>176 x 144</td>
<td>15.6</td>
</tr>
<tr>
<td>MPEG</td>
<td>MPEG_RTP</td>
<td>320 x 240</td>
<td>23.4</td>
</tr>
<tr>
<td>MPEG</td>
<td>MPEG_RTP</td>
<td>352 x 288</td>
<td>16.5</td>
</tr>
<tr>
<td>MP3</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5.7 Transmission Performance (fps) of Stored Multimedia Files

In most cases, the streaming of stored video files resulted in smooth and high quality reception of the transmitted video data. The JMF RTP managers handle the synchronization of audio and video data as long as the streams are tagged with the same source name. MP3 audio data was streamed smoothly and completely.

The JMF utilises codecs developed by IBM Inc. At the time of implementation of this project, the release of the MPEG-4 codec developed by IBM was delayed due to licensing issues. For this reason, the testing of “live” video streams with MPEG encoding was not possible.

5.4 Video Conferencing - Performance Results

Tables 5.5 and 5.6 record the performance of one-way streaming of multimedia data across a network. Video conferencing requires twice the bandwidth of a single stream of multimedia data as it requires a two-way, interactive communication channel. Table 5.8 shows the performance of the video conferencing system embedded within Multimedia Studio. The data was collected.
using the JPEG_RTP video encoding format at the various resolutions previously used. Using the same set of resolutions facilitates a comparison with results previously achieved.

<table>
<thead>
<tr>
<th>Resolution (Pixels):</th>
<th>Source Camera</th>
<th>Logitech Quickcam</th>
<th>Osprey-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>128 x 96</td>
<td>12.9</td>
<td>14.9</td>
<td></td>
</tr>
<tr>
<td>176 x 144</td>
<td>9.4</td>
<td>14.4</td>
<td></td>
</tr>
<tr>
<td>320 x 240</td>
<td>7.4</td>
<td>11.4</td>
<td></td>
</tr>
<tr>
<td>352 x 288</td>
<td>4.4</td>
<td>8.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.7 Transmission Performance of Video Conferencing System Using JPEG_RTP Encoding Format

Similar testing using the H.263 video encoding format gave poorer video transfer rates than those presented in Table 5.8. It is for this reason that these results have been excluded, as implementing H.263 video transmission would be an unfeasible course of action when implementing a distributed multimedia application. The production or availability of a more efficient codec may sway this decision at some later time.

The implementation of higher resolutions of this application presents a performance restriction as the video transfer rate is, to an extent, dependent on the prevailing network traffic. However, the realisation of a desktop conferencing system is feasible as frame rates of approximately 15 frames per second provide a relatively smooth, average quality video display.

5.5 Conclusions

The results achieved in streaming multimedia data, represent a good base for other, more efficient distributed multimedia applications. Given the limited resources, these applications perform well with respect to the frame rates and quality achieved. The ability to stream “live” video data and the performance of streaming “stored” multimedia data permits the implementation of a feasible multimedia-on-demand system in an Internet-based environment.
Chapter 6:

**Using extended-LINDA to Develop Distributed Multimedia Applications**
Morris (1999) defines Multimedia-on-Demand as,

*a term for a number of technologies whose goal is to enable individuals to select multimedia data objects from a central server for viewing on a television or computer screen to start at any point in time specified by the user.*

Linda, a distributed and parallel processing language, has found many applications for its associative object memory and has allowed programmers to implement parallelism with a small number of operations to create and coordinate parallel processes. (Gelernter, 1988)

The aim of this chapter is to introduce the reader to the implementation of on-Demand multimedia applications using a distributed memory model. The Linda implementation utilised for the development of a pilot video-on-demand system is the eLinda (extended-Linda) API developed by Dr George Wells (2001) at Rhodes University.

### 6.1 The Linda Model

Linda is a distributed and parallel programming language based on associative object memory called tuple space (TS) (Turner, 1997). Tuples are the fundamental data structure that exist within tuple space and are composed of a number of user defined data fields.

#### 6.1.1 Basic Operations of Linda

Linda comprises four basic operations that may be performed on tuple space: *out*, *eval*, *in* and *rd*. These operations are divided into two overall categories; out and eval comprise *Tuple Generation*; in and rd comprise *Tuple Extraction*. 
The `out(t)` operation causes the tuple, `t`, to be added to tuple space. The tuple added to tuple space is a data, or passive, tuple whose fields are evaluated before being added to tuple space. The `in(s)` operation calls upon a tuple, `t`, that matches a given template, `s`, to be withdrawn from tuple space. In other words, the actual values of `t` are assigned to the formal values of the template, `s`. If no matching `t` is available when `in(s)` executes, the executing process suspends until one is available. If many matching `t`'s are available, one is chosen arbitrarily. (Gelernter, 1988)

The third basic operation of Linda, `rd(s)`, is the same as `in(s)`, except that the matched tuple remains in tuple space. Lastly, `eval(t)`, is the same as `out(t)`, except that `t` is evaluated after, rather that before it enters tuple space. Figure 6.1 visually describes the above procedures.

![Figure 6.1 Simple Linda Operations. (Adapted from Gelernter, 1988)](image)

### 6.1.2 Why Linda?

The central issues in parallel programming are the communication and coordination of the parallel processes. Linda makes communication and coordination easy because all processes are controlled by the simple concept that tuple space is based on – a bag of objects that you can “put things in and take things out”. Gelernter (1988) states that Linda is an *uncoupled* programming model; no worker task deals directly with any other, all transactions go through tuple space. Also, developers can create tuples for any kind of data structure.

### 6.2 Developing an on-Demand Multimedia Application with eLinda

As Linda is a distributed system, an on-Demand multimedia application developed in eLinda potentially involves the distribution and retrieval of multimedia files for playback over an entire
computer network. The running of a program called the **TSMaster** (Tuple Space Master) acts as a “server” to monitor all the activities and operations that take place in the eLinda system. This comprises managing the communication handlers and takes care of opening the tuple spaces. (Wells, 2001)

The Video-on-Demand application developed for the purpose of this investigation is based on the simple concept of a video store; involving implementations of the actual “video store” and “clients”. The architecture of the application developed involves opening three tuple spaces: available videos, requested videos and supplied videos. The video store is able to add tuples comprising the name and location of the video file; the client is able to search for a particular video tuple by making a **rd** request to tuple space.

The video store acts as a kind of video server by waiting to receive requests in tuple space and will then add the actual multimedia data source to the “supplied” category of tuple space from where the client is able to extract and playback the video. The purpose of this application is merely to present the power of the eLinda system in facilitating distributed multimedia systems. Utilities such as security and payment have thus been excluded from this implementation. Figure 6.2 displays the operation of the eLinda Video-on-Demand application.

![Figure 6.2 Mode of Operation of Video-on-Demand Application in eLinda. Adapted from Wells (2001)](image-url)
Creating the various tuple spaces involves requesting the Tuple Space Manager to open a particular tuple space with the format of the tuples that exist in the tuple space (represented by the characters, \texttt{s}: string, \texttt{l}: long, \texttt{o}: object and \texttt{m}: multimedia resource object).

\begin{verbatim}
TupleSpace videos = TSManager.open("Videos", "sslf");
TupleSpace requests = TSManager.open("Requests", "sslo");
TupleSpace supplied = TSManager.open("Supplied", "sslm");
\end{verbatim}

Creating and adding a tuple to a particular tuple space is a matter of building the tuple from information to be stored in that tuple and adding the tuple to tuple space by invoking the \texttt{out} procedure of that tuple space. The video tuple below stores information about the video store offering the video; which has a video name, an identifying key and an associated cost.

\begin{verbatim}
videos.out(new Tuple (storeName, videoName, key, cost));
\end{verbatim}

Now that the tuple is in tuple space, a client application is able to search for that tuple by invoking the \texttt{rd} procedure with a matching tuple in order to locate and withdraw the information of the tuple, without extracting the actual tuple from tuple space. The user requests a video with a particular video name, but is not concerned with the shop from where it originates, its key or the cost of that video. Hence wildcards are inserted in place of these matching fields and are denoted here as “?<fieldname>”.

\begin{verbatim}
Tuple videoTuple;
videoTuple = videos.rd(new AntiTuple(?shop, videoName, ?key, ?cost));
\end{verbatim}

Assuming a matching tuple is found, the client is able to make a request to tuple space requesting the given video. The server, waiting for such requests, will extract the appeal from tuple space and locate the desired multimedia file, adding this to the “supplied” tuple space when found. These transactions follow a similar process to the above examples of reading and writing to tuple space.

What is left now is for the client to extract the multimedia tuple from tuple space and present it to the user.

\begin{verbatim}
Tuple video;
video = supplied.in(new AntiTuple(videoName, key, ?mmResource));
mmResource.play();
\end{verbatim}
As previously mentioned, clients may reside at any location on a computer network without being discriminated against. Thus, the effective and efficient implementation of a video-on-demand system is achieved through the use of a shared memory system: Tuple Space.

6.3 The Multimedia Component of eLinda

Although the Linda model takes care of all transactions surrounding tuple space, the transmission and playback of multimedia data is handled by the multimedia component of the eLinda system. The eLinda.mm library is independent of the eLinda communication systems and simply represents an extension of the coordination language in the area of distributed multimedia processing and support.

The core multimedia resource available to the eLinda system is the MultiMediaResource class. Every multimedia tuple added to tuple space encapsulates the multimedia object as a MultiMediaResource object. Figure 6.3 outlines the architecture of the multimedia component of the eLinda system.

```
<table>
<thead>
<tr>
<th>eLinda.mm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiMediaResource</td>
</tr>
<tr>
<td>MMServer</td>
</tr>
<tr>
<td>MMServerThread</td>
</tr>
<tr>
<td>TransmitMM</td>
</tr>
<tr>
<td>ClientStream</td>
</tr>
<tr>
<td>ReceiveRTP</td>
</tr>
</tbody>
</table>
```

Figure 6.4 High Level Architecture of the Multimedia Component of the eLinda System.

Every MultiMediaResource object, when created, launches a server process on the host machine, waiting for a request to play back the associated multimedia data source. The multimedia resource object is now available to be added to tuple space and be extracted or read by a client process running at some remote location on the network. Once extracted from tuple space, client programs are capable of invoking the multimedia object by calling the play
method of that instance of the MultiMediaResource object. Consequently, the multimedia resource object has the ability to determine whether the actual multimedia file is cached locally on the client machine or lies at a remote location, in which case it (the client’s copy of the multimedia resource) transparently connects to the actual resource via the ClientStream class. Thus, depending on its location, every multimedia resource object has the ability to behave as both the server and the “client”. Figure 6.5 graphically displays the operation of creating, submitting to tuple space and presenting a multimedia resource object.

Figure 6.5 Operation of Multimedia Support in eLinda. Adapted from Wells (2001)

The multimedia server (MMServer) associated with a given MultiMediaResource object accepts the connection requested by the client, or remote multimedia resource object, and creates a multimedia server thread (MMServerThread) which, after an initial negotiation, begins the transmission of the multimedia data via the Real-time Transport Protocol (RTP). In turn, the ClientStream object, running at the client side of the transaction, launches a ReceiveRTP thread which waits, on the previously negotiated communication channel, for the streamed multimedia data to arrive. Once received, the multimedia data is presented to the user. The transmission, reception and presentation of multimedia data via RTP make use of facilities provided by the Java Media Framework which are discussed in detail in Section 4.2.

The details concerning the transmission and playback of multimedia data are completely hidden from the application programmer. Thus, the multimedia component of eLinda provides a number of functional yet transparent multimedia facilities enabling developers to implement distributed multimedia applications in an efficient manner.
6.3 Conclusions

While performing research on the Linda model of distributed and parallel computing it was realised that the simplest on-demand multimedia applications are facilitated by the use of a shared memory system. The application developer is oblivious to the location, the communication channels and processing of multimedia data, and is simply concerned with making a request and waiting for that request to be realised, thus resulting in a truly “on-demand” application.

Consequently, it is possible, by using the tools and techniques developed to transmit and receive streaming multimedia data in conjunction with a distributed processing model such as Linda, to develop on-demand applications that exhibit the efficiency and functionality of the Real-time Transport Protocol and the power of the eLinda system.
Chapter 7:

SUMMARY AND CONCLUSIONS
Transmission of multimedia data over the Internet has traditionally been achieved by direct download thus limiting the performance, functionality and real-time characteristics of multimedia data.

The initial objective of this thesis was to investigate ways of efficiently transporting multimedia data across the internet and, using the eLinda system, develop a pilot Video-on-Demand application to demonstrate the power of the eLinda system when implementing distributed multimedia applications.

Section 7.2 of this chapter will address the extent to which these objectives were met and what extensions to these objectives were accomplished. A discussion exploring future work in the areas researched will follow.

### 7.1 Thesis Overview

The main goal of this thesis was to present to the user tools and techniques used in facilitating the efficient transport of multimedia data across the Internet. This involved identifying and describing the various compression techniques used when encoding and transmitting multimedia data across computer networks. Later chapters addressed the implementation and results of research carried out in these areas.

#### 7.1.1 Compression Techniques

Issues involved with the development of a distributed multimedia application, including video and audio encoding formats were addressed in Chapter 2. As the efficient transmission of multimedia data relies on its ability to be encoded into a format suitable for transmission and decoded into a format suitable for presentation, special attention was paid to the various video
encoding formats such as JPEG, MPEG and H.263. The mode of operation of codecs and the various types of codecs were also discussed. This discussion paved the way for deeper insight into understanding how to approach the development of distributed multimedia applications and formats that enabled the efficient transmission of multimedia data.

7.1.2 Real-time Transport Protocol

Distributed systems require some control protocol to define and maintain the session between the participants of a distributed transaction. In a multimedia context the transport protocol of choice should address issues such as the real-time streaming of “live” and “stored” video efficiently and easily. For these reasons, and a host of other features, the Real-time Transport Protocol was selected as the protocol used in maintaining a multimedia session between participants.

Due to its flexibility and extensibility, the Java Media Framework API was selected as the development environment of choice. The libraries within the JMF RTP API have provided sufficient support in accomplishing the objectives of implementing a distributed multimedia system. Chapter 3 of this thesis addressed in detail the development environment and the key features utilised in developing the final product.

It was found that using the Java Media Framework in conjunction with the Real-time Transport Protocol resulted in the efficient streaming and playback of multimedia data. The quantitative and qualitative results of this investigation were presented in Chapter 5 where it was noted that moderate resolutions of JPEG transmission resulted in the most feasible implementation of the desktop video conferencing system using the resources available.

7.1.3 Distributed Processing using eLinda

The eLinda system provided an application area that successfully enabled the development of a pilot video-on-demand application. The purpose of this exercise was to instill an appreciation of using a distributed, shared memory system when implementing on-demand multimedia applications. The process of developing an on-demand application and the operation and architecture of the multimedia component of the eLinda system were addressed in Chapter 6.
7.2 The Final Product

The system developed for this project encompasses a variety of tools available and demonstrates the power and functionality of developing a distributed multimedia system using the Java Media Framework API in conjunction with the Real-time Transport Protocol. It provides the functionality embedded in most commercial multimedia playback systems with its ability to present multimedia files stored on the local machine.

The functionality of the system also includes the ability to display live audio and video data from compatible multimedia capture devices. However, the power of the system lies in its ability to stream multimedia data using the Real-time Transport Protocol, giving it the functionality and efficiency of high-end commercial streaming applications such as Windows Media Player and Real Player.

Research and experience gained in accomplishing the functionality and efficiency of this distributed multimedia program paved the way to integrating high-level multimedia facilities and support into the eLinda system. The completion of this exercise made possible the development of an on-demand multimedia service using the eLinda system.

The results of experiments performed (presented in Chapter 5) shows that the original expectations of the project were met and exceeded as the systems developed provided the functionality of efficient streaming of multimedia data and resulted in a fully functional desktop video conferencing system. The final stages of implementation culminated in the deployment of a pilot video-on-demand application supporting all the facilities implemented in the original system.

7.3 Future Directions

Due to the platform independent characteristics of Java, multimedia applications that support the streaming of data via the Real-time Transport Protocol may be extended to run on hand-held computing and mobile devices, thus extending video conferencing to the mobile employee.
Java technology enables the deployment of applications in the form of active HTML embedded content such as applets and Java Server Pages (JSP). This implies that the applications developed may be extended to execute in the context of a web browser thus bringing the ability to stream video or perform a live conference over an Internet-based environment.

Running and deploying the Real-time Streaming Protocol (RTSP), based on the Real-time Transmission Protocol (RTP), on a video server leads the way to large scale, commercial streaming applications that facilitate on-demand video services to a large number of clients in an efficient manner. The implementation of such a system can easily be achieved by employing the tools and techniques addressed when deploying the applications used to perform this research.
Adobe Systems, Incorporated, 2001  
**Adobe Premier Technical Guides – Video Codec Compression Techniques.** [On-line]  

**ARQ-assisted H261 and H263-based Programmable Video Transceivers.** [On-line]  

Filippini, L., 1997  
**The MPEG Standard – Information, Questions and Answers.** [On-line]  

Fowler, T.J., 1997  
**Video Compression.** [On-line]  
Available:  
[http://hotwired.lycos.com/webmonkey/97/34/index1a_page2.html?tw=multimedia](http://hotwired.lycos.com/webmonkey/97/34/index1a_page2.html?tw=multimedia)

Fraunhofer-Gesellschaft, 2001  
**MPEG Audio, Layer – 3.** [On-line]  
Available: [http://www.iis.fhg.de/amm/techinf/layer3](http://www.iis.fhg.de/amm/techinf/layer3)

Gee, C., 1999  
**Video – on – Demand.** [On-line]  
Available:  

Gelernter, D., 1988  

Harris, T., 2001  
**How File Compression Works.** [On-line]  

Jarmo, H., 1995  
**An Overview of Video – on – Demand.** [On-line]  
Available:  
<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
<th>Availability</th>
</tr>
</thead>
</table>
This appendix presents the documentation for the major classes and resources that made up the Multimedia Studio application that was used to perform the experiments carried out for this research project. The development of this system is discussed in Chapter 4 of the document to which these appendices are attached. Results of the experiments performed are presented in Chapter 6: Results and System Evaluation.

The source code for the Multimedia Studio system and the pilot Video-one-Demand system developed in conjunction with George Wells is available on the compact disc submitted with this thesis. Full Java documentation for the entire system is available on-line at:

http://www.cs.ru.ac.za/research/g98D3523

Questions concerning the operation and functionality of this system may be forwarded by e-mail to:

pareen@rucus.ru.ac.za
Class Hierarchies:

**MMStudio**

### Class Hierarchy

- class java.lang.Object
  - class java.awt.Component
    - class java.awt.Container
      - class java.awt.Window
        - class java.awt.Dialog
          - class MMStudio.GetVideoDialog (implements java.awt.event.ActionListener)
          - class MMStudioInputDialog (implements java.awt.event.ActionListener)
          - class MMStudio.ServerAddressDialog (implements java.awt.event.ActionListener)
        - class java.awt.Frame
          - class MMStudio.ConferenceClient (implements java.lang.Runnable)
          - class MMStudio.ConferenceServer (implements java.lang.Runnable)
        - class MMStudio.NetInfo (implements java.io.Serializable)
      - class MMStudio.Transmit (implements java.lang.Runnable)
      - class MMStudio.TransmitVideo
        - class MMStudio.TransmitVideo.StateListener (implements javax.media.ControllerListener)
## APPLICATION: MMStudio

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Description</th>
</tr>
</thead>
</table>
| **ConferenceClient** | File: ConferenceClient.java  
This component creates a video conference "client" by establishing a connection with a remote server thus creating a desktop Video Conference via the Real-time Transport Protocol. |
| **ConferenceServer** | File: ConferenceServer.java  
This component facilitates a Video Conference session by accepting a connection from a ConferenceClient and negotiates a communication channel on which to perform the Video Conference. |
| **GetVideoDialog** | File: InputDialog.java  
This class creates a dialog box that allows the user to specify the IP Address, port number and Multimedia file to transmit to a remote machine. |
| **InputDialog** | File: InputDialog.java  
This class creates a dialog box that allows the user to specify the IP Address and transmitting port of the multimedia server when opening an RTP Session, i.e. When waiting for multimedia data to arrive via RTP. |
| **MMStudio** | File: MMStudio.java  
This class creates the Multimedia Studio application that is able to create and maintain a video conferencing session along with a host of other facilities and functionality |
| **NetInfo** | File: NetInfo.java  
This class stores information about the conference session. It encapsulates the data that is transferred to a ConferenceClient when negotiating the communication channel on which to perform the Video Conference. |
<table>
<thead>
<tr>
<th>Component</th>
<th>File:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Receive</strong></td>
<td>Receive.java</td>
<td>The Receive class opens an RTP channel on a specified port and listens for RTP data from a specified &quot;server&quot;. If RTP data is received, a window is created to display the incoming multimedia stream.</td>
</tr>
<tr>
<td><strong>ServerAddressDialog</strong></td>
<td>ServerAddressDialog.java</td>
<td>Create a dialog box requesting the IP Address of the &quot;server&quot; when setting up a video conferencing session. This class is invoked by the ConferenceClient.</td>
</tr>
<tr>
<td><strong>Transmit</strong></td>
<td>Transmit.java</td>
<td>Creates a TransmitVideo object with the specified parameters and begin transmission of multimedia data on a separate thread of execution.</td>
</tr>
<tr>
<td><strong>TransmitVideo</strong></td>
<td>TransmitVideo.java</td>
<td>The TransmitVideo class streams a specified JMF MediaLocator object to a remote location via the Real-time Transport Protocol (RTP). The transmission of the given data source takes place after manipulating the multimedia data in order to prepare it for transmission.</td>
</tr>
</tbody>
</table>
Class MMStudio

java.awt.Frame
    +-MMStudio.MMStudio

public class MMStudio
extends java.awt.Frame
implements java.awt.event.ActionListener, java.awt.event.WindowListener,
javax.media.ControllerListener

This class creates the Multimedia Studio application that is able to create and maintain a video
conferencing session. It can also load and play local and remote multimedia files and can capture
live video and audio data with any registered capture devices.

Inner classes inherited from class java.awt.Component
java.awt.Component.AWTTreeLock

Field Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>private java.lang.String</td>
<td>address</td>
<td>Networking variable.</td>
</tr>
<tr>
<td>private java.lang.Thread</td>
<td>client</td>
<td>Control threads to begin and end execution of video conferencing variables.</td>
</tr>
<tr>
<td>private ConferenceClient</td>
<td>confClient</td>
<td>ConferenceClient object if the current session is run as a video conferencing client.</td>
</tr>
<tr>
<td>private ConferenceServer</td>
<td>confServer</td>
<td>ConferenceServer object if the current session is run as a video conferencing client.</td>
</tr>
<tr>
<td>private boolean</td>
<td>dataReceived</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>private</td>
<td>destAddr</td>
<td></td>
</tr>
<tr>
<td>javax.media.rtp.SessionAddress[]</td>
<td>Networking variable.</td>
<td></td>
</tr>
<tr>
<td>private</td>
<td>devices</td>
<td></td>
</tr>
<tr>
<td>java.util.Vector</td>
<td>Store list of capture devices available on the host machine.</td>
<td></td>
</tr>
<tr>
<td>private</td>
<td>manager</td>
<td></td>
</tr>
<tr>
<td>javax.media.rtp.RTPManager []</td>
<td>Store managers used to facilitate an RTP Session.</td>
<td></td>
</tr>
<tr>
<td>private</td>
<td>player</td>
<td></td>
</tr>
<tr>
<td>javax.media.Player</td>
<td>Maintain information about the current playing multimedia resource.</td>
<td></td>
</tr>
<tr>
<td>private int []</td>
<td>port</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Networking variable.</td>
<td></td>
</tr>
<tr>
<td>private int</td>
<td>portNumber</td>
<td></td>
</tr>
<tr>
<td>private Receive</td>
<td>rtpSession</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Receive RTP object for creating RTP Session on local machine.</td>
<td></td>
</tr>
<tr>
<td>private</td>
<td>server</td>
<td></td>
</tr>
<tr>
<td>java.lang.Thread</td>
<td></td>
<td></td>
</tr>
<tr>
<td>private int</td>
<td>ttl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specifies a given inputs Time-To-Live.</td>
<td></td>
</tr>
<tr>
<td>private</td>
<td>videoCDI</td>
<td></td>
</tr>
<tr>
<td>javax.media.CaptureDeviceInfo</td>
<td>Store information about selected video capture device.</td>
<td></td>
</tr>
<tr>
<td>private</td>
<td>videoDevices</td>
<td></td>
</tr>
<tr>
<td>java.util.Vector</td>
<td>List of available video capture devices on host machine.</td>
<td></td>
</tr>
</tbody>
</table>
private java.util.Vector videoFormats

Store information about the available capture formats of a selected video capture device.

**Constructor Summary**

**MMStudio()**
Constructor. Create and display Graphical User Interface.

**Method Summary**

<table>
<thead>
<tr>
<th>void</th>
<th>actionPerformed(java.awt.event.ActionEvent click)</th>
</tr>
</thead>
<tbody>
<tr>
<td>private void</td>
<td>captureVideoFeed()</td>
</tr>
<tr>
<td></td>
<td>Present the capture of the default capture devices to the user.</td>
</tr>
<tr>
<td>private void</td>
<td>closeAllSessions()</td>
</tr>
<tr>
<td></td>
<td>Close all currently running RTP Sessions.</td>
</tr>
<tr>
<td>private void</td>
<td>closeFile()</td>
</tr>
<tr>
<td></td>
<td>Closes the currently playing media file.</td>
</tr>
<tr>
<td>private void</td>
<td>closeVConfSession()</td>
</tr>
<tr>
<td></td>
<td>Method to terminate Conferencing session.</td>
</tr>
<tr>
<td>void</td>
<td>controllerUpdate(javax.media.ControllerEvent event)</td>
</tr>
<tr>
<td></td>
<td>Implementation of ControllerListener.</td>
</tr>
<tr>
<td>private void</td>
<td>exit()</td>
</tr>
<tr>
<td></td>
<td>Close the application, all RTP Sessions and all currently playing multimedia.</td>
</tr>
<tr>
<td>private void</td>
<td>getAudioFormats()</td>
</tr>
<tr>
<td></td>
<td>Retrieves the supported audio formats for each of the registered audio capture devices.</td>
</tr>
<tr>
<td>private void</td>
<td>getDevices()</td>
</tr>
<tr>
<td></td>
<td>Gets the registered capture devices of the local machine.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>getVideoFormats()</td>
<td>Retrieves the supported video formats for each of the registered video capture devices.</td>
</tr>
<tr>
<td>itemStateChanged(java.awt.event.WindowEvent e)</td>
<td></td>
</tr>
<tr>
<td>main(java.lang.String[] args)</td>
<td>Create the Multimedia Studio Application</td>
</tr>
<tr>
<td>openMediaFile()</td>
<td>Open file dialog and play selected file.</td>
</tr>
<tr>
<td>openSession()</td>
<td>Open RTP (receiving) session.</td>
</tr>
<tr>
<td>openVConfSessionAsClient()</td>
<td>Method to open Video Conferencing session as client.</td>
</tr>
<tr>
<td>openVConfSessionAsServer()</td>
<td>Method to open Video Conferencing session as server.</td>
</tr>
<tr>
<td>setFrameRate(float newRate)</td>
<td>Set the frame rate of the (live) video data to a specified value when capturing video data.</td>
</tr>
<tr>
<td>transmitVideo()</td>
<td>Open RTP transmission session.</td>
</tr>
<tr>
<td>viewReports()</td>
<td>Generate and display the RTCP reports for a given session.</td>
</tr>
<tr>
<td>windowActivated(java.awt.event.WindowEvent e)</td>
<td></td>
</tr>
<tr>
<td>windowClosed(java.awt.event.WindowEvent e)</td>
<td></td>
</tr>
<tr>
<td>windowClosing(java.awt.event.WindowEvent e)</td>
<td></td>
</tr>
<tr>
<td>windowDeactivated(java.awt.event.WindowEvent e)</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>windowDeiconified</td>
<td>java.awt.event.WindowEvent e</td>
</tr>
<tr>
<td>windowIconified</td>
<td>java.awt.event.WindowEvent e</td>
</tr>
<tr>
<td>windowOpened</td>
<td>java.awt.event.WindowEvent e</td>
</tr>
</tbody>
</table>
Class TransmitVideo

public class TransmitVideo extends java.lang.Object

TransmitVideo.java

The TransmitVideo class sends a specified JMF MediaLocator object to a remote location via the Real-time Transport Protocol (RTP).

Inner Class Summary

| (package private) class | TransmitVideo.StateListener |

Field Summary

<table>
<thead>
<tr>
<th>private</th>
<th>javax.media.protocol.DataSource</th>
<th>dataOutput</th>
<th>Store the output of the processor in the output data source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>boolean</td>
<td>failed</td>
<td></td>
</tr>
<tr>
<td>private</td>
<td>java.lang.String</td>
<td>ipAddress</td>
<td>Stores the IP address of the target client.</td>
</tr>
<tr>
<td>private</td>
<td>boolean</td>
<td>isLive</td>
<td>Check if the transmitting video data source is the video feed from a capture device.</td>
</tr>
<tr>
<td>private</td>
<td>javax.media.MediaLocator</td>
<td>locator</td>
<td>Store the location of the transmitting multimedia object.</td>
</tr>
<tr>
<td>private</td>
<td>javax.media.protocol.DataSource</td>
<td>myDS</td>
<td>Temporary store for the processed data source.</td>
</tr>
</tbody>
</table>
private java.lang.String port

private javax.media.Processor processor
    Controlling processor for transforming the multimedia data source to a format suitable for transmission.

private javax.media.rtp.RTPManager[] rtpmgrs
    Array of RTP Managers to control each transmitting multimedia stream.

private java.lang.Integer stateLock
    Convenience object to handle processor's state changes

### Constructor Summary

(package private) TransmitVideo(javax.media.MediaLocator locator, java.lang.String ipAddress, java.lang.String port)
    Constructor.

### Method Summary

private javax.media.Format checkForVideoSizes(javax.media.Format original, javax.media.Format supported)
    Ensure that the transmission sizes of the frames are supported.

private boolean createProcessor()
    Create the processor for the media locator and program it to encode RTP compatible packets for transmission.

private void createTransmitter()
    Transmit video and audio streams by capturing the output video and audio streams from the processor's output and creating an RTP manager to control each transmitting stream.

(package private) java.lang.Integer getStateLock()
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setBitRate(int rate)</td>
<td>Set the bit rate to the specified value.</td>
</tr>
<tr>
<td>setFailed()</td>
<td></td>
</tr>
<tr>
<td>setFrameRate(float newRate)</td>
<td>Set the frame rate of the (live) video data to a specified value.</td>
</tr>
<tr>
<td>setPacketSize(int size)</td>
<td>Set the packet size to the specified value.</td>
</tr>
<tr>
<td>setQuality(javax.media.Player p, float val)</td>
<td>Set the encoding quality to the specified value on the JPEG encoder.</td>
</tr>
<tr>
<td>start()</td>
<td>Starts the transmission process by creating the processor.</td>
</tr>
<tr>
<td>stop()</td>
<td>Stops the transmission process by closing the processor and disposing the RTP managers.</td>
</tr>
<tr>
<td>waitForState(javax.media.Processor p, int state)</td>
<td></td>
</tr>
</tbody>
</table>
Class Receive

java.awt.Frame
  +-MMStudio.Receive

public class Receive
extends java.awt.Frame
implements javax.media.rtp.SessionListener, javax.media.rtp.ReceiveStreamListener,

The Receive class opens an RTP channel on a specified port and listens for RTP data from a
specified "server". It then presents the received multimedia data to the clients screen.

Inner classes inherited from class java.awt.Component

java.awt.Component.AWTTreeLock

Field Summary

<table>
<thead>
<tr>
<th>private java.lang.String</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>private boolean</td>
<td>dataReceived</td>
</tr>
<tr>
<td>private java.lang.Object</td>
<td>dataSync</td>
</tr>
<tr>
<td>private javax.media.rtp.SessionAddress[]</td>
<td>destAddr</td>
</tr>
<tr>
<td>private java.net.InetAddress</td>
<td>ipAddr</td>
</tr>
<tr>
<td></td>
<td>Store the IP Address of the multimedia server.</td>
</tr>
<tr>
<td>private javax.media.rtp.SessionAddress</td>
<td>localAddr</td>
</tr>
<tr>
<td></td>
<td>Maintain the local information of the RTP Session.</td>
</tr>
<tr>
<td>private</td>
<td>javax.media.rtp.RTPManager[]</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td></td>
<td>Array of RTP Managers to control each received multimedia stream.</td>
</tr>
<tr>
<td>private</td>
<td>int[]</td>
</tr>
<tr>
<td>private</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>Store the receiving port.</td>
</tr>
<tr>
<td>private</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>Record the time to live of the received multimedia stream.</td>
</tr>
</tbody>
</table>

**Constructor Summary**

**Receive**(java.lang.String addr, int portNumber)

Constructor.

**Method Summary**

<table>
<thead>
<tr>
<th>void</th>
<th>controllerUpdate(javax.media.ControllerEvent event)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Implementation of Interface: ControllerListener</td>
</tr>
<tr>
<td>void</td>
<td>itemStateChanged(java.awt.event.WindowEvent e)</td>
</tr>
<tr>
<td>private boolean</td>
<td>openRTPSession()</td>
</tr>
<tr>
<td></td>
<td>This method opens RTPSessions on two RTPManagers (Single audio and video streams).</td>
</tr>
<tr>
<td>void</td>
<td>run()</td>
</tr>
<tr>
<td>void</td>
<td>stop()</td>
</tr>
<tr>
<td></td>
<td>Stops the reception of RTP data by shutting down each of the RTP Managers.</td>
</tr>
<tr>
<td>void</td>
<td>update(javax.media.rtp.event.ReceiveStreamEvent streamEvent)</td>
</tr>
<tr>
<td></td>
<td>Implementation of Interface: RemoteStreamListener</td>
</tr>
<tr>
<td>Method</td>
<td>Signature</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>update</td>
<td><code>void update(javax.media.rtp.event.SessionEvent sessionEvent)</code></td>
</tr>
<tr>
<td>waitforRTPData</td>
<td><code>void waitforRTPData()</code></td>
</tr>
<tr>
<td>windowActivated</td>
<td><code>void windowActivated(java.awt.event.WindowEvent e)</code></td>
</tr>
<tr>
<td>windowClosed</td>
<td><code>void windowClosed(java.awt.event.WindowEvent e)</code></td>
</tr>
<tr>
<td>windowClosing</td>
<td><code>void windowClosing(java.awt.event.WindowEvent e)</code></td>
</tr>
<tr>
<td>windowDeactivated</td>
<td><code>void windowDeactivated(java.awt.event.WindowEvent e)</code></td>
</tr>
<tr>
<td>windowDeiconified</td>
<td><code>void windowDeiconified(java.awt.event.WindowEvent e)</code></td>
</tr>
<tr>
<td>windowIconified</td>
<td><code>void windowIconified(java.awt.event.WindowEvent e)</code></td>
</tr>
<tr>
<td>windowOpened</td>
<td><code>void windowOpened(java.awt.event.WindowEvent e)</code></td>
</tr>
</tbody>
</table>
Class ConferenceServer

public class ConferenceServer
extends java.lang.Object
implements java.lang.Runnable

This class creates a video conference session that facilitates the communication between two remote "clients".

Field Summary

<table>
<thead>
<tr>
<th>private boolean</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>private Receive</td>
<td>receive</td>
</tr>
<tr>
<td>private InetAddress</td>
<td>remoteAddr</td>
</tr>
<tr>
<td>private ServerSocket</td>
<td>serverSock</td>
</tr>
<tr>
<td>private boolean</td>
<td>timeToEnd</td>
</tr>
<tr>
<td>private Transmit</td>
<td>transmit</td>
</tr>
</tbody>
</table>

Constructor Summary

ConferenceServer()

Method Summary

void closeSession()

Close the current conference session.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>private void createSession()</td>
<td>Create RTP Conferencing session with single client.</td>
</tr>
<tr>
<td>void run()</td>
<td></td>
</tr>
<tr>
<td>private void waitForConnection()</td>
<td>Create server socket and wait for client to connect.</td>
</tr>
</tbody>
</table>
**Class ConferenceClient**

```java
public class ConferenceClient
extends java.lang.Object
implements java.lang.Runnable
```

This class creates a video conference "client" that communicates with a remote partner.

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>inetAddr</td>
<td>java.net.InetAddress</td>
</tr>
<tr>
<td>info</td>
<td>NetInfo</td>
</tr>
<tr>
<td>listenPort</td>
<td>int</td>
</tr>
<tr>
<td>receive</td>
<td>Receive</td>
</tr>
<tr>
<td>SERVER_PORT</td>
<td>final int</td>
</tr>
<tr>
<td>serverInetAddr</td>
<td>java.net.InetAddress</td>
</tr>
<tr>
<td>transmit</td>
<td>Transmit</td>
</tr>
<tr>
<td>writePort</td>
<td>int</td>
</tr>
</tbody>
</table>
### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ConferenceClient</strong></td>
<td><code>java.lang.String serverInetAddr</code></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>closeSession()</strong></td>
<td>Close the current conference session.</td>
</tr>
<tr>
<td><strong>createSession()</strong></td>
<td>Create and start RTP Conference session with the provided information.</td>
</tr>
<tr>
<td><strong>getNetInfo()</strong></td>
<td>Retrieve the information provided by the server.</td>
</tr>
<tr>
<td><strong>negotiate()</strong></td>
<td>Try to create communication channel with &quot;server&quot; and initiate conference.</td>
</tr>
<tr>
<td><strong>run()</strong></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX B: GRAPHICS

Figure B.1 Appearance of Multimedia Studio

Figure B.2 Comparative Images Showing:
   a.) Video Capture with Logitech Quickcam
   b.) Video Capture with Viewcast Osprey-100
Figure B.3  Video playback (Quicktime Movie [.mov])

Figure B.4  Streamed Video (H.263_RTP)