VIDEO CONFERENCING WITH MULTICAST SUPPORT

A

Project Report

Submitted in partial fulfillment of the

requirements of the award of the degree of

Bachelor of Technology

of

National Institute of Technology Calicut

By

Abhilash Chacko M. V Y2.031

Ashok Mazumder Y2.274

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT

KOZHIKODE - 673601

Acknowledgement

We thank Mr.K.A.Abdul Nazeer, Senior Lecturer, Department of Computer Science and Engineering NIT Calicut for his guidance and co-operation. We also acknowledge the advice and help given to us by our friends. We would like to extend our gratitude to the entire faculty and staff of the CSED NITC, who stood by us in all pits and falls we had to face during the development phase of this project.

Abhilash Chacko M. V Y2.031

Ashok Mazumder Y2.274

CERTIFICATE

National Institute of Technology, Calicut.
Department of Computer Engineering

Certified that this PROJECT entitled

Video Conferencing With Multicast Support

is a bonafide work carried out by

Abhilash Chacko M. V Y2.031

Ashok Mazumder Y2.274

in partial fulfillment of his
BACHELOR OF TECHNOLOGY
under our guidance

Mr. K. A. Abdul Nazeer
Senior Lecturer
Dept. of Computer Engineering

Dr. M. P. Sebastian
Professor and Head
Dept. of Computer Engineering

Table of Content

1 INTRODUCTION 7

1.1 Project Overview 7

1.2 Problem Overview 7

1.2.1 Streaming Bandwidth and Storage Issue 8

1.2.2 Protocol Issue 8

1.2.3 Codec Issue 9

1.3 Motivation for Project 9

1.4 Project Objectives 10

1.5 Additional Considerations 10

1.5.1 Social and Political Issue 10

1.6 Report Overview 10

1.7 Chapter Summary 11

2 BACKGROUND 12

2.1 Background Research 12

2.1.1 Real Time Transport Protocol Overview 13

Figure 2.1: RTP Architecture 14

2.1.2 Java Media Framework API Overview 14

Figure 2.2: JMF Architecture 14

2.2 Related Work 15

2.3 Platform and Software Choice for the project. 16

2.3.1 Hardware Choice 17

2.4 Chapter Summary 17

3 DESIGN 18

3.1 Design for the Sender Side 19

Figure 3.1: Processor to Process Raw Media Data 20

Figure 3.2: RTP Sender 22

3.2 Design for the Receiver Side 22

Figure 3.3: RTP Reception Data Flow 22

4 IMPLEMENTATION 24

4.1 Key Concepts 24

4.2 Data Format. 24
Figure 4.1: JMF Media Formats 25
4.3 Manager 26
4.4 Capture the Media Data. 26
4.4.1 Data Model 27
Figure 4.2: JMF Data Model 28
4.5 Process Media Data 28
Figure 4.3: JMF Processor 28
4.5.1 Configuring a Processor 29
4.5.2 Selecting Track Processing Options and Changing Format 30
4.5.3 Specifying the Output Data Format 30
4.6 Transmitting RTP Data with RTP Manager. 30
4.6.1 Creating Send Stream and Streaming 30
4.7 Receiving and Presenting RTP Media Streams. 31
4.7.1 Creating a Player for an RTP Session 31
4.7.2 Creating RTP Player for Each New Receive Stream and Playing 32
4.8 Unicast Session 33
4.9 Multi Unicast Session. 34
4.10 Multicast Session. 34
5 TESTING 36
5.1 Testing Criteria/plan 36
5.2 Test Results 36
6 Conclusion 37
6.2 Summery of Work 37
6.2 Future Work 38
7 REFERENCES 39

ABSTRACT

This project aims to implement a video conferencing with multicast support software. This software is preferred to use this over Local Area Network (LAN) than internet due to the need for high bandwidth .It can be used to unicast, multicast or broadcast live captured media streams according to user wish. The same software can be used to receive the media streams and playback to the user. Unlike other server based systems this is a peer-to-peer software. This is based on progressive downloading. By using a message passing system every user are able to get a list of online users.

Chapter 1

INTRODUCTION

This Chapter outlines the aims of the project and motivation behind its implementation.

1.1 Project Overview

This project aims to implement a video conferencing with multicast support software. This software is preferred to use this over Local Area Network (LAN) than internet due to the need for high bandwidth .It can be used to unicast, multicast or broadcast live captured media streams according to user wish. The same software can be used to receive the media streams and playback to the user. This is based on progressive downloading.

1.2 Problem Overview

This software is coded in Java. It is mainly a LAN video conferencing software. We can capture live videos by using some capture devices like webcam. User can select destination points which can be a single destination (Unicast) or more (Multicast).The live captured video data will be presented at the destination points with a very small delay. We are using Real-Time Transport Protocol (RTP) for the live conferencing purpose.For monitoring purpose RTP is augmented with RTCP (Real-Time Tranport Control Procol).Here Underlying transport layer protocol is UDP. We will capture the video stream using the webcam and the audio stream using the microphone. We have to encode the tracks of the streams to an RTP compatible format. For video track we are using JPEG,H.263 or MPEG compression algorithms and we will get a RTP compatible form. For audio track we will use PCM, MPEG_AUDIO or GSM .We are using an API, JMF (Java Media Framework) released by Sun for multimedia programming in Java.

Input Video Format

-YUV format

-15 frame/s

Input Audio Format

-linear

-32100 or 44100 sample rate

-16 bits/sample

-stereo

RTP Session

-RTCP Bandwidth fraction 5% . ie for sending RTCP Reports like Sender Report,Receiver Report etc we are using 5% of the total bandwidth.

-RTCP Sender Bandwidth fraction 25% of RTCP Band width fraction .ie for sending Sender Report we are using 25% of the RTCP Bandwidth fraction.

1.2 Issues to be Addressed

1.2.1 Streaming Bandwidth and Storage Issue

Bandwidth is the limiting factor associated with communication. The limiting factor in a system is referred to as the bottleneck. Sending video through a communications channel requires a lot of bandwidth. So this software is preferred to use this over Local Area Network (LAN) than internet as bandwidth is high in the case of Local Area Network.

Streaming media storage size (in the common file system measurements mebibytes, gibibytes, tebibytes, and so on) is calculated from streaming bandwidth and length of the media with the following formula (for a single user and file):

storage size (in mebibytes) = length (in seconds) · bit rate (in kbit/s) / 8,388.608

(since 1 mebibyte = 8 * 1,048,576 bits = 8,388.608 kilobits)

Real world example:

One hour of video encoded at 300 kbit/s (this is a typical broadband video for 2005 and its usually encoded in a 320×240 pixels window size) will be:

(3,600 s · 300 kbit/s) / 8,388.608 = 128.7 MiB of storage

if the file is stored on a server for on-demand streaming. If this stream is viewed by 1,000 people, you would need

300 kbit/s · 1,000 = 300,000 kbit/s = 300 Mbit/s of bandwidth

1.2.2 Protocol Issue

Designing a network protocol to support streaming media raises many issues.

Datagram protocols, such as the User Datagram Protocol (UDP), send the media stream as a series of small packets, called datagrams. This is simple and efficient; however, packets are liable to be lost or corrupted in transit. Depending on the protocol and the extent of the loss, the client may be able to recover the data with error correction techniques, may interpolate over the missing data, or may suffer a dropout.

The Real-time Transport Protocol (RTP), the Real Time Streaming Protocol (RTSP) and the Real Time Control Protocol (RTCP) were specifically designed to stream media over the network. They are all built on top of UDP.

Reliable protocols, such as the Transmission Control Protocol (TCP), guarantee correct delivery of each bit in the media stream. However, they accomplish this with a system of timeouts and retries, which makes them more complex to implement. It also means that when there is data loss on the network, the media stream stalls while the protocol handlers detect the loss and retransmit the missing data. Clients can minimize the effect of this by buffering data for display.

1.2.3 Codec Issue

A common problem when an end user wants to watch a video stream encoded with a specific codec is that if the exact codec is not present and properly installed on the user's machine, the video won't play (or won't play optimally). It is possible for multiple codecs to be available in the same product, avoiding the need to choose a single dominant codec for compatibility reasons. In the end it seems unlikely that one codec will replace them all.

1.3 Motivation for Project

The importance of communication over LAN and internet is increasing day by day. Video conferencing is the ultimate form of communication. Network facilities are getting cheaper and communication channel bandwidth is increasing, in this scenario video chatting virtually decreases the distance between humans.

1.4 Project Objectives

This project aims to implement video conferencing with multicast support software.

1.5 Additional Considerations

1.5.1 Social and legal issues

Some streaming broadcasters use streaming systems that interfere with the ability to record streams for later playback, either inadvertently through poor choice of streaming protocol or deliberately because they believe it is to their advantage to do so. Broadcasters may be concerned that copies will result in lost sales or that consumers may skip commercials. Whether users have the ability and the right to record streams has become a significant issue in the application of law to cyberspace.

In principle, there is no way to prevent a user from recording a media stream that has been delivered to their computer. Thus, the efforts of broadcasters to prevent this consist of making it inconvenient, or illegal, or both.

1.6 Report Overview

The remaining sections of this report focus on the various stages of research, design, implementation, and testing and finally the analysis of the project discussed above.

Chapter 2 BACKGROUND

Summarises the research taken into the disciplines related to this project. Additionally,

related systems and tools are also looked at.

Chapter 3 DESIGN

This chapter contains the details of the approaches taken to design the systems and components. Details regarding the important choices made in context to implementation are also discussed.

Chapter 4 FURTHER DESIGN AND IMPLEMENTATION

In this chapter, the implementation of the software is discussed in line with any other design issues that have not been looked at in the previous chapter or issues that have arose. Choices made and problems encountered also discussed.

Chapter 5 – TESTING

This chapter will document the testing performed on the entire Framework. In addition to this, there is also an evaluation of all the work carried out and also documentation of further work that can be carried out is also in this chapter.

Chapter 7– CONCLUSION

Final words on the successful completion of the project

1.7 Chapter Summary

This chapter has given an introduction to the final year project problem, as well as the motivation behind it with aims and expectations of the final solution. The contents of the whole documents have also been introduced. The next chapter includes research undertake and problems encountered during the early stages of the development.

Chapter 2
BACKGROUND

The background section of this project looks at areas considered during the development phase of the framework. This includes research conducted, assessing similar domains and past related solutions. This section will also consider tools and development environments needed for the implementation of the final solution.
In addition, the Background section will also assess the benefits and downfalls of high a number of high-level languages to be used for the implementation

2.1 Background Research

Attempts to display media on computers date back to the earliest days of computing, in the mid-20th century. However, little progress was made for several decades, due primarily to the high cost and limited capabilities of computer hardware.

Academic experiments in the 1970s proved out the basic concepts and feasibility of streaming media on computers.

During the late 1980s, consumer-grade computers became powerful enough to display various media. The primary technical issues with streaming were:

- Having enough CPU power and bus bandwidth to support the required data rates
- Creating low-latency interrupt paths in the OS to prevent buffer under run.

However, computer networks were still limited, and media were usually delivered over non-streaming channels. In the 1990s, CD-ROMs became the most prevalent method of media distribution to computers.

The late 1990s saw:

- Greater network bandwidth, especially in the last mile
- Increased access to networks, especially the Internet

- Use of standard protocols and formats, such as TCP/IP, HTTP, and HTML
- Commercialization of the Internet

These advances in computer networking combined with powerful home computers and modern operating systems to make streaming media practical and affordable for ordinary consumers.

In general, multimedia content is large, such that media storage and transmission costs are still significant; to offset this somewhat, media are generally compressed for both storage and streaming.

A media stream can be *on demand or live*. On demand streams are stored on a server for a long period of time, and are available to be transmitted at a user's request. Live streams are only available at one particular time, as in a video stream of a live sporting event.

2.1.1 Real Time Transport Protocol (RTP) Overview

RTP is the Internet-standard protocol (RFC 1889, 1890) for the transport of real-time data, including audio and video. RTP consists of a data and a control part called RTCP. The data part of RTP is a thin protocol providing support for applications with real-time properties such as continuous media (e.g., audio and video), including timing reconstruction, loss detection, security and content identification. RTCP provides support for real-time conferencing of groups of any size within an internet. This support includes source identification and support for gateways like audio and video bridges as well as multicast-to-unicast translators. It offers quality-of-service feedback from receivers to the multicast group as well as support for the synchronization of different media streams. None of the commercial streaming products uses RTP (Real-time Transport Protocol), a relatively new standard designed to run over UDP. Initially designed for video at T1 or higher bandwidths, it promises more efficient multimedia streaming than UDP.

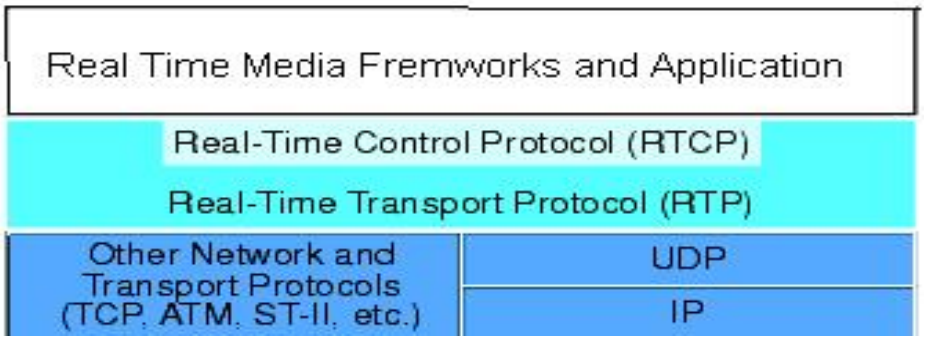

Figure 2.1: RTP Architecture

2.1.2 Java Media Framework API (JMF)

The JMF 1.0 API (the Java Media Player API) enabled programmers to develop Java programs that presented time-based media. The JMF 2.0 API extends the 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 defines a plug-in API that enables advanced developers and technology providers to more easily customize and extend JMF functionality.

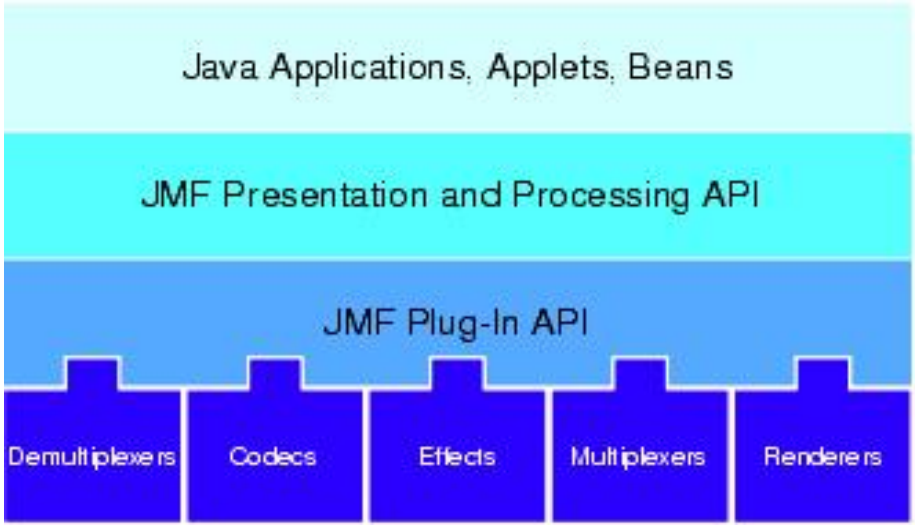

Figure 2.2: JMF Architecture

Design Goals for the JMF API

JMF 2.0 supports media capture and addresses the needs of application developers who want additional control over media processing and rendering. It also provides a plug-in architecture that provides direct access to media data and enables JMF to be more easily customized and extended. JMF 2.0 is designed to:

- Be easy to program
- Support capturing media data
- Enable the development of media streaming and conferencing applications in Java
- Enable advanced developers and technology providers to implement custom solutions based on the existing API and easily integrate new features with the existing framework
- Provide access to raw media data
- Enable the development of custom, downloadable demultiplexers, codecs, effects processors, multiplexers, and renderers (JMF *plug-ins*)
- Maintain compatibility with JMF 1.0

2.2 Related Work

There have been many papers describing the setup and control of multimedia streams over a variety of networks. See for example [IEEE90] for a good collection of papers. Some systems implement real-time voice over synchronous channels [Leung90], while others run receiving video codecs at a higher rate than the transmitting codecs [Casner90] to ensure minimum delay. Systems with adaptive buffering for jitter and clock drift compensation have been proposed, based upon an analysis of recent packet delay times, but these systems often require synchronized clocks, carefully selected parameters or end-to-end cooperation to work successfully. See [Naylor82] for a good analysis of such systems. In contrast, clawback buffers only require the setting of one parameter (related to the rate at which jitter conditions change), operate purely at the destination with no knowledge of the sources, and do not require timestamps.

2.3 Platform and Software Choice for the project.

A major factor to be taken into consideration is that an object-oriented language approach would be more suited for this project, as this will allow different components to be represented independently. Additionally, as this project's graphical user interface, Java provides an extensive API for graphical development. Furthermore Borland Java have a number of developing environments that allow users to exploit graphical development.
Also with the adaptation of a waterfall model software engineering approach, an object-oriented approach will allow the author to refer back to any stage of devolvement and amend or alter framework design and also the implementation. An object-oriented approach will make it easier for the author to add delete and modify components at any stage.

The two contenders for the choice of programming languages to be used are C# and Java:

C# is a fairly new object-oriented language from Microsoft that is based around the .NET Framework. As an object-oriented language, C# supports inheritance, polymorphism, class and custom attributes. In addition C# prevents programs from accessing objects inappropriately and supports garbage collection and memory management. Visual studio.NET is a complete set of development tools available for application building that is available to me if I was to choose C#.

Java is a portable and a high level language which compiles to byte code. Java also supports inheritance, polymorphism, class and custom attributes as well as preventing programs from accessing objects inappropriately along with its own garbage collection. Java is also a platform providing a virtual machine upon which programs can run with the API provided. To develop applications in Java, there are a number of options available ranging from Borlands JBuilder (high end of the market) to Helios Text pad (a text editor with code highlighting and auto syntaxting). JDK (Java Development Kit) can be easily accessible from java.sun.com. More over Java supports media application.

We choose to use the java programming language to implement the project because past

experiences using the language lead to the belief that is was adequate for the job. In addition, with a major aim of the Framework , being platform independent and portable, java provides an ideal solution to meet this aim.

2.3.1 Hardware Choice

A PC with Windows based operating system would be adequate for hardware required to carry out the implementation. As all the components are represented in high level languages rather than in kernel space, no expensive equipment is required. The LANs (Local Area Networks) allows sharing of files, software and other resources which makes it easier to do the work at home and in the labs.

2.4 Chapter Summary

This chapter has discussed in some detail the routing protocol and API we will be using, architecture of them are discussed. In addition, software and hardware choices were discussed and a decision on them where made to support the implementation phase of the project. The next chapter discusses the design process involved before the implementation of the project solution.

Chapter 3

DESIGN

This chapter describes the design decisions made that aim to meet the aims that were specified in the introductory chapter. This chapter will also bring in some of the ideas and research outlined in the previous chapter.

In the design of the real-time software, we found that the most severe constraints were imposed by the audio streams. The ambience of a two-way live conversation is very sensitive to audio delays in the 10 to 20ms range, whereas video has more of a supporting role and can stand delays in the 40 to 80ms range before conversation becomes impaired. Furthermore, errors in the continuity of an audio stream are much more noticeable than in a video stream.

The software is divided in to major parts Sender Side and Receiver Side Other than this, the design for the software is.

- Whenever a new user will come he/she will broadcast a message with his/her IP address and his/her status. The status of a user can be either Open or Close. If a user already joined in any conference his/her status will be Closed else Open. Just after making a connection with any other the user, both user will change their status from Open to Close. By default whenever a new user joins, his status will be Open.
- Each user will maintain an online list of other users present in the network based on the message passing. The list will contain the user name and the status of the user.
- The user with status Open will send broadcast message periodically ,with a period 5s.
- If a user wishes to leave the conferencing then he will broadcast open message so that all users can update their list.
- For the multicast session the server (the peer which hosts the mulicast session)

will unicast mulicastopensage to the specified clients.The server will create a list of users participating in that multicast session according to the reply coming from the users

3.1 Design for the Sender Side

Following stages should be followed in the Sender side.

1. We have to specify the exact format of the Video and Audio. The format of the Audio data will be"linear", 44100 samples/second, 16 bits/sample, and stereo where linear means linear audio encoding technique used by this format where other encoding technique can be PCM_SIGNED, PCM_UNSIGNED, Alaw. 44100 is the number of samples played or recorded per second, for sounds that have this format. 16 bits/sample means the number of bits in each sample of a sound that has this format is 16. Stereo is the type where other type is mono. Video format will be :"YUV",15frames/s.
2. After deciding the audio and video format we have to check for the supported capture devices. If supported device is found then we will have to find out the media locator for them. We will create the DataSource object for that media locator
3. The media processing should be done before transmitting. So we will create a Processor object for the media locator. We will prepare the Processor for transmission which consists of various stages. Along with that we will program the Processor to get suitable out put format (RTP compatible).If necessary we have to resize the video for the application of compression algorithms like JPEG or H.263. So the input of the processor is data source and output is also a data source.

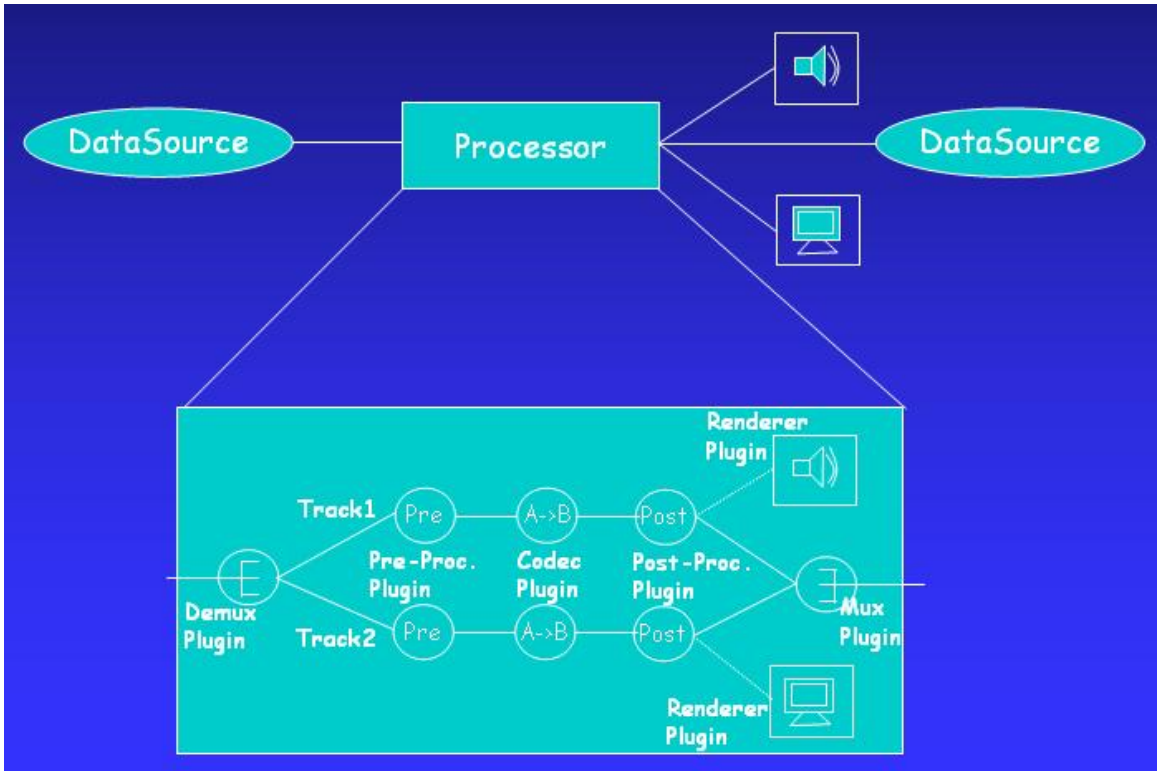

Figure 3.1: Processor to Process the Raw Media Data.

There are eight stages for the Processor. They are given below.

- **Configuring:** A `Processor` enters the configuring state from the unrealized state when the `configure()` method is called. A `Processor` exists in the configuring state when it connects to the `DataSource`, demultiplexes the input stream, and accesses information about the format of the input data.
- **Configured:** From the configuring state, a `Processor` moves into the configured state when it is connected to the `DataSource` and the data format has been determined.
- **Unrealized:** In this state, the `Player` object has been instantiated. Like a newborn baby who does not yet recognize its environment, a newly instantiated `Player` does not yet know anything about its media.
- **Realizing:** A `Player` moves from the unrealized state to the realizing state when you call the `Player`'s `realize()` method. In the realizing state, the `Player` is in the process of determining its resource requirements. A

realizing `Player` often downloads assets over the network.

- **Realized:** Transitioning from the realizing state, the `Player` comes into the realized state. In this state the `Player` knows what resources it needs and has information about the type of media it is to present. It can also provide visual components and controls, and its connections to other objects in the system are in place
- **Prefetching:** When the `prefetch()` method is called, a `Player` moves from the realized state into the prefetching state. A prefetching `Player` is preparing to present its media. During this phase, the `Player` preloads its media data, obtains exclusive-use resources, and does whatever else is needed to play the media data.
- **Prefetched:** The state where the `Player` has finished prefetching media data -- it's ready to start.
- **Started:** This state is entered when you call the `start()` method. The `Player` is now ready to present the media data
-

To capture audio and video from a video camera, process the data, and broadcast it then the steps to be followed by the processor is.

- The audio and video tracks would be captured.
- Effect filters would be applied to the raw tracks (if desired).
- The individual tracks would be encoded.
- The compressed tracks would be passed to RTP Manager to broadcast..

4. Initialize the RTP session with the local session address.For creating a Session Address we have to provide the IP Address and the port number. We are using separate sessions for video stream and audio stream
5. Create media streams for transmission through each session.
6. Open connection with the required destinations.
7. Synchronize both the stream and start streaming

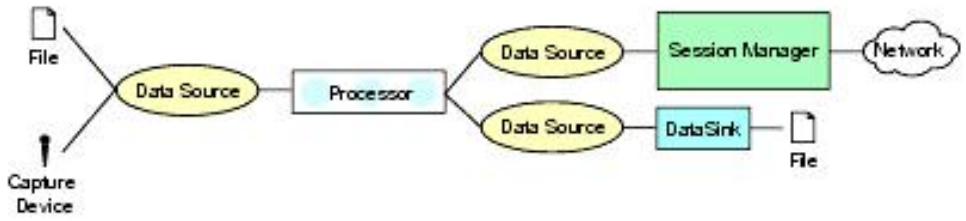

Figure 3.2: RTP Sender

3.2 Design for the Receiver Side

Following stages should be followed in the Receiver side

2 A separate thread will be listening to a special port for receiving purpose

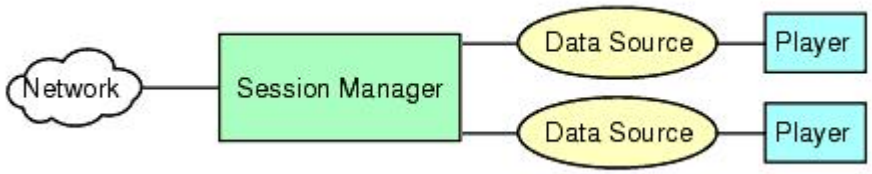

Figure 3.3: RTP Reception Data Flow

3 If it detects any new receive streams then it will create a player which consists of several stages. JMF defines six stages for the player

- **Unrealized:** In this state, the `Player` object has been instantiated. Like a newborn baby who does not yet recognize its environment, a newly instantiated `Player` does not yet know anything about its media.
- **Realizing:** A `Player` moves from the unrealized state to the realizing state when you call the `Player`'s `realize()` method. In the realizing state, the `Player` is in the process of determining its resource requirements. A realizing `Player` often downloads assets over the network.
- **Realized:** Transitioning from the realizing state, the `Player` comes into the realized state. In this state the `Player` knows what resources it needs and has information about the type of media it is to present. It can also provide visual components and controls, and its connections to other objects in the

system are in place

- **Prefetching:** When the `prefetch()` method is called, a `Player` moves from the realized state into the prefetching state. A prefetching `Player` is preparing to present its media. During this phase, the `Player` preloads its media data, obtains exclusive-use resources, and does whatever else is needed to play the media data.
- **Prefetched:** The state where the `Player` has finished prefetching media data -- it's ready to start.
- **Started:** This state is entered when you call the `start()` method. The `Player` is now ready to present the media data.

4 A separate players will be needed for each incoming stream. If both are ready double check for the synchronization and start playback.

.

Chapter 4

IMPLEMENTATION

This chapter describes the development of the final project solution based upon the decision in the Design chapter and ideas formed during this phase. This chapter also goes onto link the previous design Chapter into the Implementation phase with advanced design decisions take further. This includes the class definitions, models and pseudo code representing noteworthy system functionality.

4.1 Key Concepts

This section will break down the key aspects of the final Solution and discuss their importance, functionality and other implementation related issues. This is more an overview with majority of the technical-oriented discussion and decagons making process being carried outer later in the chapter.

The key aspects can be broken down into the following bullet points.

- Data Format
- Manager
- Capture Media Data.
- Process Media Data
- Transmitting RTP Data with Session Manager
- Receiving and Presenting RTP Media Streams
- Unicast Session
- Multi Unicast Session
- Multicast Session

4.2 Data Format

The exact media format of an object is represented by a `Format` object. The format itself carries no encoding-specific parameters or global timing information, it describes the format's encoding name and the type of data the format requires.

JMF extends `Format` to define audio- and video-specific formats.

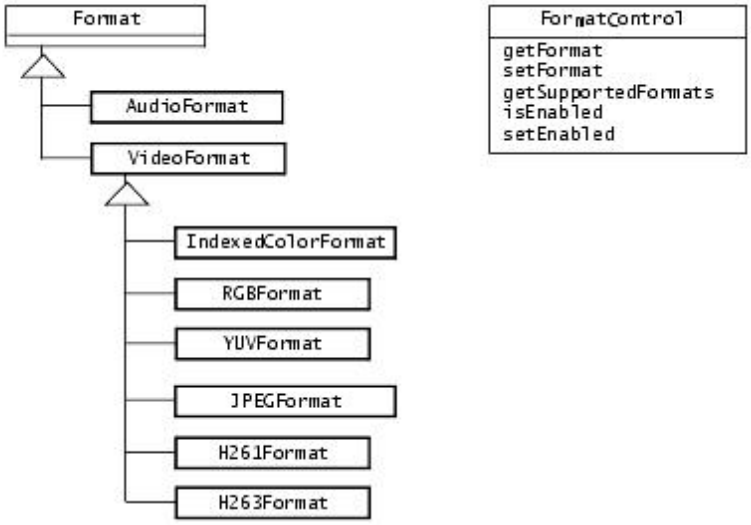

Figure 4.1: JMF Media Formats

An `AudioFormat` describes the attributes specific to an audio format, such as sample rate, bits per sample, and number of channels. A `VideoFormat` encapsulates information relevant to video data. Several formats are derived from `VideoFormat` to describe the attributes of common video formats, including:

- `IndexedColorFormat`
- `RGBFormat`
- `YUVFormat`
- `JPEGFormat`

- `H261Format`
- `H263Format`

To receive notification of format changes from a `Controller`, you implement the `ControllerListener` interface and listen for `FormatChangeEvents`.

4.3 Manager

A manager, an intermediary object, integrates implementations of key interfaces that can be used seamlessly with existing classes For example, with `Manager` you can create a `Player` from a `DataSource`. JMF offers four managers:

- **`Manager:`** Use `Manager` to create `Players`, `Processors`, `DataSources`, and `DataSinks`. For example, if you want to render a `DataSource`, you can use `Manager` to create a `Player` for it.
- **`PackageManager:`** This manager maintains a registry of packages that contain JMF classes, such as custom `Players`, `Processors`, `DataSources`, and `DataSinks`.
- **`CaptureDeviceManager:`** This manager maintains a registry of available capture devices.
- **`PlugInManager:`** This manager maintains a registry of available JMF plug-in processing components.

4.4 Capture the Media Data.

Media capture is an important task in JMF programming. One can capture media data using a capture device such as a microphone or a video camera. It can then be processed and rendered, or stored in a media format. To capture media data, we need to:

- Locate the capture device you want to use by querying the `CaptureDeviceManager`
- Obtain a `CaptureDeviceInfo` object for the device
- Get a `MediaLocator` from the `CaptureDeviceInfo` object and use it to create a `DataSource`
- Create either a `Player` or a `Processor` using the `DataSource`
- Start the `Player` or `Processor` to begin the capture process

We used the `CaptureDeviceManager` to access capture devices available on the system. This manager acts as the central registry for all capture devices available to JMF. we can obtain an available device list by calling the `getDeviceList()` method. A capture device is represented by a `CaptureDeviceInfo` object. we used the `CaptureDeviceManager`'s `getDevice()` method to get the `CaptureDeviceInfo` for a particular capture device.

To use the capture device to capture media data, we then need to get the device's `MediaLocator` from its `CaptureDeviceInfo` object. You can either use this `MediaLocator` to construct a `Player` or a `Processor` directly, or use the `MediaLocator` to construct a `DataSource` that you can use as the input to a `Player` or `Processor`. Use the `Player`'s or `Processor`'s `start()` method to initiate the capture process.

4.4.1 Data Model

JMF media players usually use `DataSources` to manage the transfer of media-content. A `DataSource` encapsulates both the location of media and the protocol and software used to deliver the media. Once obtained, the source cannot be reused to deliver other media.

A `DataSource` is identified by either a JMF `MediaLocator` or a `URL` (universal resource locator). A `MediaLocator` 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.

(In Java, a `URL` can only be constructed if the corresponding protocol handler is installed on the system.)

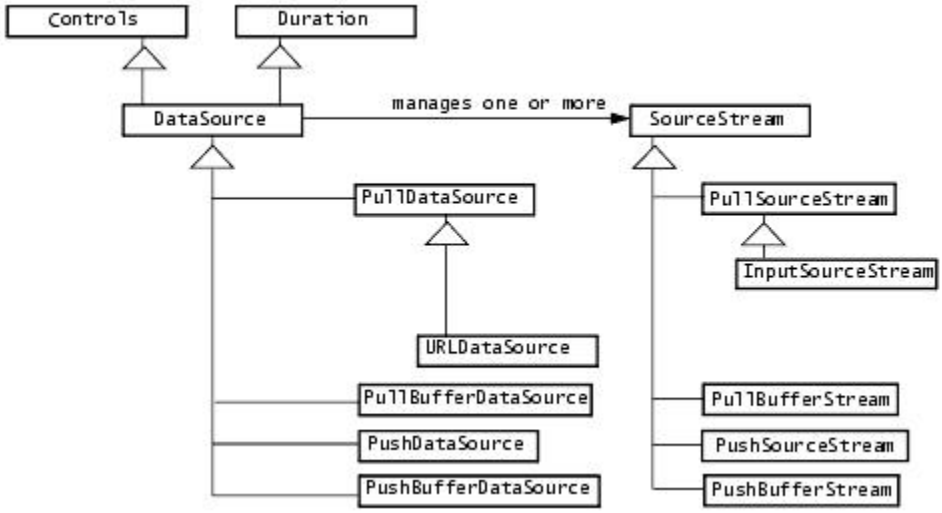

Figure 4.2: JMF Data Molel

4.5 Process Media Data

A `Processor` can be used as a programmable `Player` that enables to control the decoding and rendering process. A `Processor` can also be used as a capture processor that enables you to control the encoding and multiplexing of the captured media data.

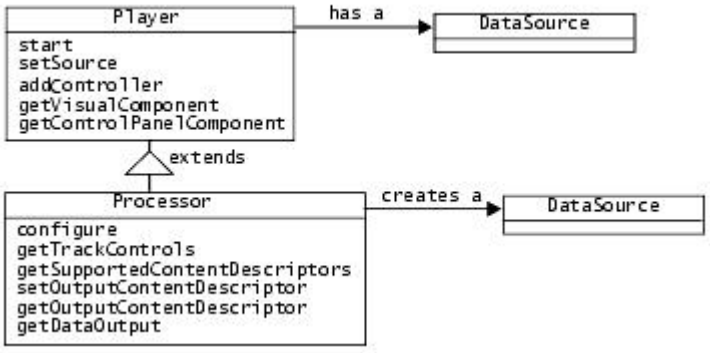

Figure 4.3: JMF Processor

We can control what processing is performed by a `Processor` several different ways:

- Use a `ProcessorModel` to construct a `Processor` that has certain input and output characteristics.
- Use the `TrackControl setFormat` method to specify what format conversions are performed on individual tracks.
- Use the `Processor setOutputContentDescriptor` method to specify the multiplexed data format of the `Processor` object's output.
- Use the `TrackControl setCodecChain` method to select the `Effect` or `Codec` plug-ins that are used by the `Processor`.
- Use the `TrackControl setRenderer` method to select the `Renderer` plug-in used by the `Processor`.

4.5.1 Configuring a Processor

In addition to the *Realizing* and *Prefetching* phases that any `Player` moves through as it prepares to start, a `Processor` also goes through a *Configuring* phase. You call `configure` to move an *Unrealized* `Processor` into the *Configuring* state.

While in the *Configuring* state, a `Processor` gathers the information it needs to construct `TrackControl` objects for each track. When it's finished, it moves into the *Configured* state and posts a `ConfigureCompleteEvent`. Once a `Processor` is *Configured*, you can set its output format and `TrackControl` options. When we're finished specifying the processing options, we call `realize` to move the `Processor` into the *Realizing* state and begin the realization process.

Once a `Processor` is *Realized*, further attempts to modify its processing options are not guaranteed to work. In most cases, a `FormatChangeException` will be thrown.

4.5.2 Selecting Track Processing Options and Changing Format

- Called `getTrackControls` on the `Processor` to get a `TrackControl` for each track in the media stream. The `Processor` must in the *Configured* state before calling `getTrackControls`.
- Used the `TrackControl` `setFormat` method to specify the format to which we want edto convert the selected track.

4.5.3 Specifying the Output Data Format

We used the `Processor` `setContentDescriptor` method to specify the format of the data output by the `Processor`. We got a list of supported data formats by calling `getSupportedContentDescriptors`.

4.6 Transmitting RTP Data with RTPManager

The basic process for transmitting RTP data with the session manager is:

- Retrieve the output `DataSource` from the `Processor`.
- Call `createSendStream` on a previously created and initialized RTP`Manager`, passing in the `DataSource` and a stream index. The session manager creates a `SendStream` for the specified `SourceStream`.
- Now RTPManager has to identify its destinations.Then call the addTarget() method for the destination. Now connection is okay.The transmission can be started.
- Control the transmission through the `SendStream` methods. A `SendStreamListener` can be registered to listen to events on the `SendStream`.

4.6.1 Creating Send Stream and Streaming

Before the RTPmanager can transmit data, it needs to know where to get the data to transmit. When we construct a new `SendStream`, we hand the `SessionManager` the `DataSource` from which it will acquire the data. Since a `DataSource` can contain multiple streams, we also need to specify the index of the stream to be sent in this session. We can create multiple send streams by passing different `DataSources` to `createSendStream` or by specifying different stream indexes.

The session manager queries the format of the `SourceStream` to determine if it has a registered payload type for this format. If the format of the data is not an RTP format or a payload type cannot be located for the RTP format, an `UnSupportedFormatException` is thrown with the appropriate message. Dynamic payloads can be associated with an RTP format using the `RTPManager addFormat` method.

4.7 Receiving and Presenting RTP Media Streams

JMF `Players` and `Processors` provide the presentation, capture, and data conversion mechanisms for RTP streams. A separate player is used for each stream received by the session manager. You construct a `Player` for an RTP stream through the standard `Manager createPlayer` mechanism. You can either:

- Use a `MediaLocator` that has the parameters of the RTP session and construct a `Player` by calling `Manager.createPlayer(MediaLocator)`
- Construct a `Player` for a particular `ReceiveStream` by retrieving the `DataSource` from the stream and passing it to `Manager.createPlayer(DataSource)`.

If you use a `MediaLocator` to construct a `Player`, you can only present the first RTP stream that's detected in the session. If you want to play back multiple RTP streams in a session, you need to use the `SessionManager` directly and construct a `Player` for each `ReceiveStream`.

4.7.1 Creating a Player for an RTP Session

When you use a `MediaLocator` to construct a `Player` for an RTP session, the `Manager` creates a `Player` for the first stream detected in the session. This `Player` posts a `RealizeCompleteEvent` once data has been detected in the session.

By listening for the `RealizeCompleteEvent`, you can determine whether or not any data has arrived and if the `Player` is capable of presenting any data. Once the `Player` posts this event, you can retrieve its visual and control components.

Note: Because a `Player` for an RTP media stream doesn't finish realizing until data is detected in the session, you shouldn't try to use `Manager.createRealizedPlayer` to construct a `Player` for an RTP media stream. No `Player` would be returned until data arrives and if no data is detected, attempting to create a *Realized* `Player` would block indefinitely.

A `Player` can export one RTP-specific control, `RTPControl`, which provides overall session statistics and can be used for registering dynamic payloads with the `RTPManager`.

4.7.2 Creating an RTP Player for Each New Receive Stream and Playing

To play all of the `ReceiveStreams` in a session, you need to create a separate `Player` for each stream. When a new stream is created, the session manager posts a `NewReceiveStreamEvent`. Generally, you register as a `ReceiveStreamListener` and construct a `Player` for each new `ReceiveStream`. To construct the `Player`, you retrieve the `DataSource` from the `ReceiveStream` and pass it to `Manager.createPlayer`.

To create a `Player` for each new receive stream in a session:

1. Set up the RTP session:
    a. Create a `SessionManager`. For example, construct an instance of `com.sun.media.rtp.RTPSessionMgr`. (`RTPSessionMgr` is an implementation of `SessionManager` provided with the JMF reference implementation.)
    b. Call `RTPSessionMgr addReceiveStreamListener` to register as a listener.
    c. Initialize the RTP session by calling `RTPSessionMgr initSession`.
    d. Start the RTP session by calling `RTPSessionMgr startSession`.
2. In your `ReceiveStreamListener update` method, watch for `NewReceiveStreamEvent`, which indicates that a new data stream has been detected.
3. When a `NewReceiveStreamEvent` is detected, retrieve the `ReceiveStream` from the `NewReceiveStreamEvent` by calling `getReceiveStream`.

4. Retrieve the RTP `DataSource` from the `ReceiveStream` by calling `getDataSource`. This is a `PushBufferDataSource` with an RTP-specific `Format`. For example, the encoding for a DVI audio player will be DVI_RTP.
5. Pass the `DataSource` to `Manager.createPlayer` to construct a `Player`. For the `Player` to be successfully constructed, the necessary plug-ins for decoding and depacketizing the RTP-formatted data must be available

4.8 Unicast Session

The following code fragment illustrates how we havw created a unicast session:

```
import java.net.*;
import javax.media.rtp.*;

// create the RTP Manager
RTPManager rtpManager = RTPManager.newInstance();

// create the local endpoint for the local interface on
// any local port
SessionAddress localAddress = new SessionAddress();

// initialize the RTPManager
rtpManager.initialize( localAddress);

// add the ReceiveStreamListener if you need to receive data
// and do other application specific stuff
// ...

// specify the remote endpoint of this unicast session
// the address string and port numbers in the following lines
// need to be replaced with your values.
InetAddress ipAddress = InetAddress.getByName( "168.1.2.3");

SessionAddress remoteAddress = new SessionAddress( ipAddress, 3000);

// open the connection
rtpManager.addTarget( remoteAddress);
```

```
// create a send stream for the output data source of a processor
// and start it
DataSource dataOutput = createDataSource();

SendStream sendStream = rtpSession.createSendStream( dataOutput, 1);
sendStream.start();

// send data and do other application specific stuff,
// ...

// close the connection if no longer needed.
rtpManager.removeTarget( remoteAddress, "client disconnected.");

// call dispose at the end of the life-cycle of this RTPManager so
// it is prepared to be garbage-collected.
rtpManager.dispose();
```

4.9 Multi Unicast Session

Creating multi-unicast sessions is similar to unicast session above. After creating and starting the SendStream new remote endpoints may be added by subsequent addTarget calls:

```
addTarget( remoteAddress2);
addTarget( remoteAddress3);
```

4.10 Multicast Session

Creating and participating in multicast sessions also works similar to the unicast. Instead of specifying local and remote endpoints a multicast session address needs to be created and passed into the initialize and addTarget calls. Everything else follows the unicast

```
//...

// create a multicast address for 224.1.1.0 and ports 3000/3001
```

```
IPAddress ipAddress = InetAddress.getByName( "224.1.1.0");

SessionAddress multiAddress = new SessionAddress( ipAddress, 3000);

// initialize the RTPManager
rtpManager.initialize( multiAddress);

// add the target
rtpManager.addTarget( multiAddress);

// ...
```

Chapter 5

TESTING

This section will begin by stating the testing criteria for the framework and then show the results of the tests that are aimed to establish whether the system functions correctly.

5.1 Testing Criteria/plan

We have tested our software in the each phase of its development. In the first step after configuring the processor we have tested the processor whether the processor is correctly working or not. Then after interfacing the capture device with the program we checked whether it is supporting the given format or not. After coming out the each stage of the processor we checked the output whether its working according to the theory or not. This testing method carried out until we reached to our goal.

Finally the software has been successfully tested on the small network or more specifically on the Local Area Network (LAN). The software is working properly in all respect for video conferencing as well as for audio chatting.

5.2 Test Results

As we tested our software in its each phase of its development and successfully over come each stage. So finally we reached to our goal of our software development. This project has been wholly successful in designing and implementing a Video Conferencing with Multicast Support Software

Chapter 6

Conclusion

The software is complete as per the SRS specified and the Design provide but is still under development to add more things like we are planning to implement our own compression and decompression technique. Also we are planning to modify our code so that compression algorithm can be applied dynamically according to the available bandwidth.

The software has been successfully tested on the small network or more specifically on the Local Area Network (LAN).

This project has helped me to understand the concepts and visualize the importance of the Software Engineering and how Java Language can be used for multimedia programming. Through this project we learned about the Java Media Framework API and how it can be used for conferencing.

The project helped us to learn the basic design principles that go in the background in the development of any Conferencing Software.

6.1 Summary of work

In conclusion, this project has been wholly successful in designing and implementing a Video Conferencing with Multicast Support Software. The software has audio chatting facility also and it supports the video on demand facility also. Although there is a lot of future work to be carried out.

6.2 Future Work

The software is complete as per the SRS specified and the Design provide but is still under development to add more things like we are planning to implement our own compression and decompression technique. We are using only JPEG compression algorithm for the compression of the video, so we are planning to modify our code so that compression algorithm can be applied dynamically according to the available bandwidth.

.

.

REFERENCES

1. Peterson L.L. & Davie B.S., *Computer Networks, A systems approach*, 3/E, Harcourt Asia, 2003.
2. Details about the Java Media framework is given in the following link: http://www.java.sun.com/products/java-media/jmf/
3. Huadong Ma and Kang G. Shin.Multicast, *"Multicast Video-on-Demand Services"*. Journal of ACM SIGCOMM Computer communication Review, ACM Press, January 2002.
4. Steven McCanne and Van Jacobson *"A Flexible Framework for Packet Video"* Journal of The third ACM International Conferencing on Multimedia. ACM Press, January 1995.
5. Ian Sommerville, *Software Engineering, 6/e*, Pearson Education Asia, 2001
6. Introduction about the streaming is available in the following link http://www.angelfire.com/ar2/videomanual1/streaming.html
7. Thesis papers about the video conferencing is available here in the link http://www.teamsolutions.co.uk/tspapers.html
8. Details about the video conferencing constrains are available here in the link http://www.mediacollege.com/video/streamingz.