**Audio-Video Conferencing**

Project for Advanced Internet Services

By Aayush Beri & Di Wen

1. **ABSTRACT**

In this project, we achieved Audio/Video communication on Internet, including the collection, playback, codec, and transmission issues about Audio/Video signals. We used Java Media Framework (JMF) for the implementation, which is an API providing a framework for Audio/Video media manipulation in Java. JMF utilizes RTP protocol for multi-media data transmissions, and RTCP to provide control and ensure reliability. Based on UDP, RTP works either in one-to-one or one-to-may. The one-to-many transmission in RTP is implemented using multicast.

Key words for our project include: Audio/Video conferencing, JMF, RTP, RTCP, UDP, multicast.

1. **JMF BASICS**

JMF extends Java’s functionalities including collection, compression, transmission, playback of media content and the support for multiple media formats and codec, including M-JPEG, H.263, MP3 and RTP/RTSP. As a well-developed API, JMF also allows programmers to utilize its functions to implement a variety of applications.

The model of JMF is very simple. Firstly, we get signals from Capture Device. Then we store the signal into DataSource, and send the signal to Player. Player will determine where to send the signals to depending on signal formats.



Figure 1: JMF Model

The common classes in JMF include:

1. DataSource:

In DataSource we store the data of the media. DataSource means “the data from source”. Here source can be everything: Camera, Speaker, Media File, or stream downloaded from Internet. Once we create a DataSource object, we get the information of the location of multi-media data and how to play the data. We can then directly send DataSource objects to the Player.

1. MediaLocator:

DataSource is defined either using MediaLocator or Universal Resource Locator (URL). MediaLocator is similar to URL and can be constructed from URL. It is used to locate the source of data, including capture devices, media files, and RTP data flows from the Internet.

1. Player:

Player reads the DataSource, and outputs it via screen or sound devices. Player has 6 different states, and needs to go through all these states to become fully prepared to start playing media content: Unrealized, Realizing, Realized, Prefetching, Prefetched, and Started. The transition between states are shown below:



Figure 2: State Transitions on Player

When a Player’s state changes, TransitionEvent will be generated, and we can check the current state and other information via ControllerListener.

1. Processor:

Processor is a special kind of Player. It extends Player and has all functionalities of a Player. The special characteristic of Processor is that it can output to a DataSouce besides media devices. Besides, Processor has 2 more states than Player, which are Configuring and Configured between Unrealized and Realizing. The state transfer for a Processor is shown below:



Figure 3: State Transitions on Processor

1. DataSink:

DataSink reads from DataSouce and outputs the data to places such as a file, Internet transmission or RTP broadcasting instead of screen and sound devices.

1. Format:

Format keeps the information of media format, such as the coding and data types. Format includes AudioFormat and VideoFormat, and VideoFormat itself includes H261Format, H263ZFormat, IndexedColorFormat, JPEGFormat, RGBFormat and YUVFormat.

1. Manager:

Manager works as the interface to combine 2 different objects. For example, when we want to play a DataSource, we can call Manager.createPlayer() to construct a Player for the DataSource object. Manager creates objects of Player, Processor, DataSource and DataSink.

JMF uses events to notify the system status to programs. A MediaEvent is initialized when JMF needs to report its current status. MediaEvent has subclasses ControllerEvent, DataSinkEvent, GainChangeEvent, and RTPEvent. JMF defines a corresponding listener for the an event. We rewrite the methods in listeners when handling events, and call addListener method to get access to the event. Following is the figure for ControllerEvent, which is a good example of the MediaEvent architecture:



Figure 4: Controller Events

1. **RTP & RTCP**
2. RTP (Real-Time Transport Protocol) is designed to provide the real-time data flow transport based on UDP. (This is because UDP has much lower delay than TCP, although unreliable) RTP only ensures the data transportation. RTCP provides the services such as reliable transfer, flow control and congestion control for RTP.

RTP solves the unpredictability of the data arrival time, which is the most important factor in multi-media transport. RTP protocol has timestamp, serial number and some other structures for “real-time” transport. The sender sets timestamp in packet according to samples, and the receiver can recover from the packet to the original data according to the timestamp.

As a transport layer protocol, RTP moves many functions to application layer in order to simplify transport layer actions for better efficiency. These functions include synchronization and flow control. Besides, during RTP conversations we use different ports for data and control, which greatly improves the flexibility and simplicity.

1. RTCP (Real-Time Transport Control Protocol) packets are periodically transmitted between participants during a RTP conversation. These packets contain statistical results including numbers of delivered or lost data packets, etc. RTCP is responsible for the management of the transport efficiency.

As we discussed in RTP, 2 ports are used in RTP conversations: data port (RTP) and control port (RTCP). According to the information in RTCP packets, applications may dynamically change the transport rate and other values to maximize the efficiency.

1. Integration of RTP and JMF:

In JMF, we use RTPManager to send out either captured audio/video data or local media files.



Figure 5: RTP Sending

We can also use RTP Manager to receive RTP data flows and either play them

locally or save them to local files.



Figure 6: RTP Receiving

RTPManager objects can also generate events:



Figure 7: RTP events

1. **IMPLEMENTATION OF THE AUDIO-VIDEO CONFERENCING**
2. Capture Audio/Video Signals:

The first step for audio/video chat is the collection of audio/video signals from microphone and camera. In this module, we firstly get the list of capture devices, take the first device in the list, and get the signal. The activity diagram is as follows:

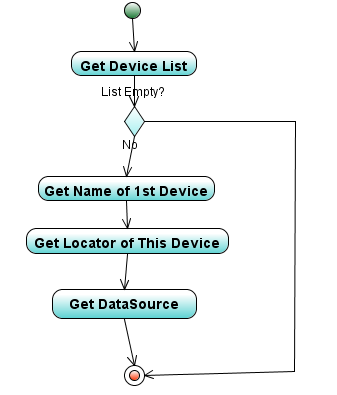


Figure 8: Capture Audio/Video Signals

Procedure:

1. Call CaptureDeviceManager.getDeviceList(Format) for deviceList.
2. If deviceList is empty, then finish. Else, use deviceList.elementAt(0) for the first available device, and cast it to CaptureDeviceInfo object.
3. Call CaptureDeviceInfo.getLocator() to get the MediaLocator.
4. Create DataSource from MediaLocator.
5. Process Audio/Video Signals:

The original signal can’t be directly transmitted via Internet. We need to change Content Type & Format. Call Processor.setContentDescriptor(ContentDescriptor) to set content type, and Processor.getTrackControls().setFormat(Format) to set the format. Activity diagram is as follows:

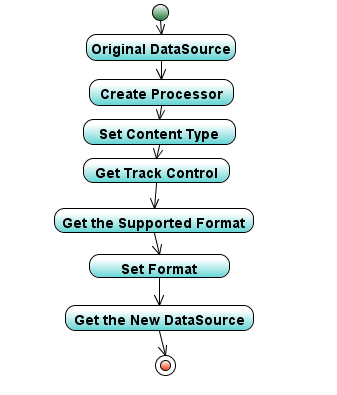


Figure 9: Process Audio/Video Signals

1. Transmit Audio/Video Signals

Here we use RTP for the transmission of the data. The activity diagram is as follows:

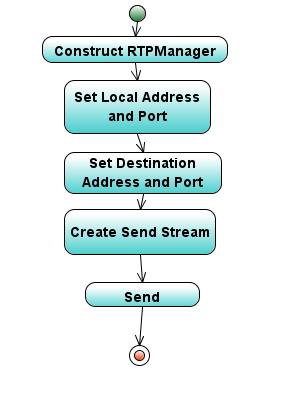


Figure 10: Transmit Audio/Video Signals

Procedure:

1. Create RTPManager by RTPManager.newInstance()/
2. Set the local and destination addresses and ports by RTPManager.initialize(RTPConnector)
3. Create sending stream by RTPManager.createSendStream(DataSource dataSource, int streamIndex).
4. Call sendStream.start().
5. Receive and Play Audio/Video Signals:

Here we use RTP to receive and play audio/video signals, and use RTCP for conversation management. We have the following 3 procedures and their corresponding activity diagrams:

Procedure 1 – Set RTP Manager:

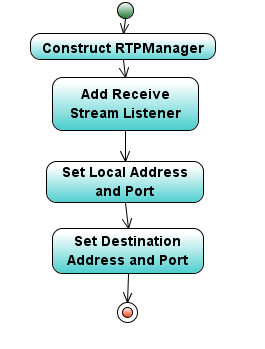


Figure 11: Set RTP Manager

1. Create and initialize RTPManager.
2. Call RTPManager.addReceiveStreamListener(ReceiveStreamListener) to add a listener to RTPManager.
3. Set local and destination addresses and ports.

Procedure 2 -- RTP event processing:



Figure 12: RTP Event Processing

1. ReceiveStreamListener notices the NewReceiveStreamEvent.
2. Get the ReceiveStream.
3. Get the received DataSource via ReceiveStream.getDataSouce().
4. Create Player based on received DataSouce.
5. Add Controller Listener to the Player to watch the state change of Player.
6. Change the Player state from unrealized to realized.

Procedure 3 – Player Event Processing:

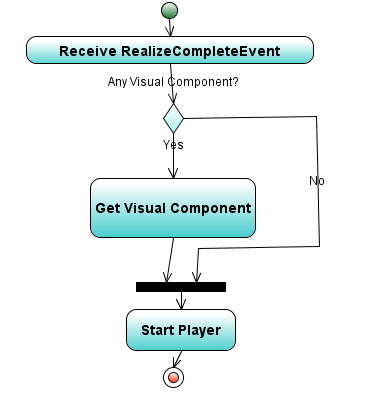


Figure 13: Player Event Processing

1. The Event Listener notices RealizeCompleteEvent.
2. Call getSourceController() from this event to get the Player.
3. Call Player.getVisualComponent() for the visual component. Judge whether this visual component is empty.
4. If not, then show the visual component and play DataSouce (for video signals). If yes, directly play DataSource (for audio signals).
5. Screenshots for The Project:

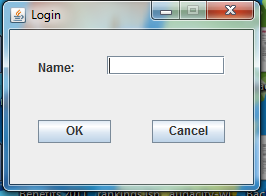


Figure 14: Login Frame

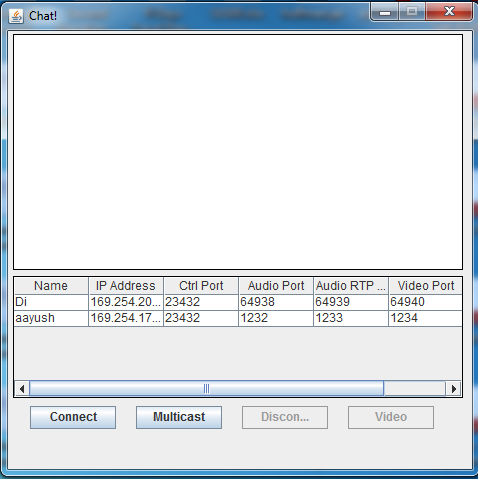


Figure 15: Main Frame

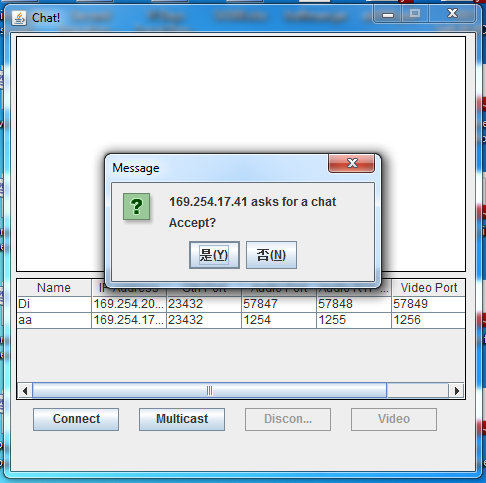


Figure 16: Connection Request

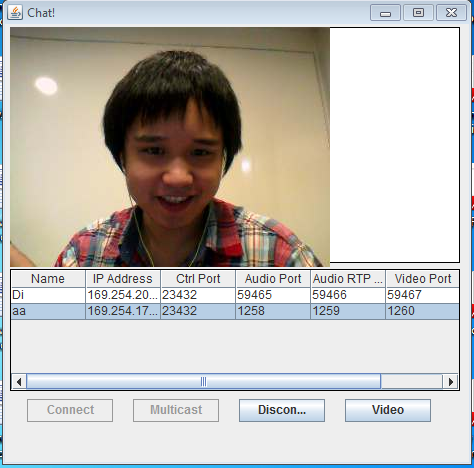


Figure 17: Unicast Connection

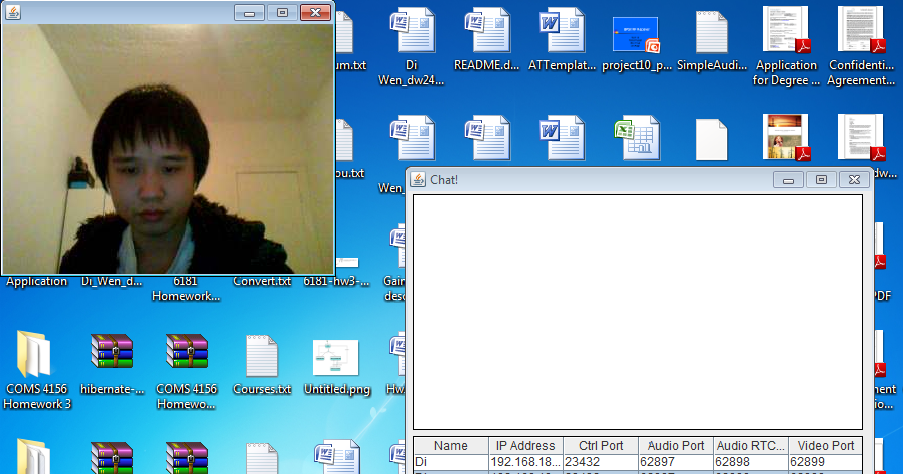


Figure 18: Multicast Connection

1. **LESSONS LEARNED & FUTURE PLAN**
2. Unimplemented Functionalities in Our Initial Plan
3. The conferencing system only runs on LAN due to NAT restrictions. We planned to make it run through web by introducing Stun servers and SIP.
4. Initially in the plan we wanted to include text chat and floor control as well.
5. Difficulties We Had During the Project
6. We took a lot of time setting up JMF. JMF doesn’t work with 64bit jdk. JMF also faces compatibility issues with the camera for some laptop manufacturers. Finally we overcome these difficulties by setting 32bit jdk and using laptops with cameras compatible with JMF.
7. JMF is an old and not up-to-date framework, which doesn’t support a lot of formats. But finally we are fine with it. Just don’t involve unsupported formats.
8. Lessons Learned
9. From this project, we better understand difficulties faced by VoIP applications with IPV4.
10. We learned valuable lessons w.r.t team management, schedule management, group collaboration, and communication between team members.
11. Future Plan
12. To use Stun servers (Simple Traversal of UDP through NATs) and SIP which allow traversal of media over Internet. Make our application a true WEB application.
13. Add Text Chat and Floor Control. Add Admin role. Attach our database schema so that we have constant users.

**REFERENCES:**

1. JMF 2.1 API Guide

<http://java.sun.com/products/java-media/jmf/2.1.1/specdownload.html>

1. “Understanding JMF”

<http://www.cs.odu.edu/~cs778/jmflects/lect2Understading.html>

1. <http://www.cs.uccs.edu/~cs525/jmf/jmf.html>
2. “JMF Understandings”

<http://members.jcom.home.ne.jp/3117216601/jmf2guide/JMFArchitecture.html>

1. “JMF RTP API”

<https://prof.hti.bfh.ch/myf1/www/projects/polyphem/www/documents/design-rdg-jmfrtpapi.html>

1. API for RTP and RTCP classes

<http://fmj-sf.net/doc/fmj/net/sf/fmj/media/rtp/>

1. IBM Tutorial for JMF

<http://www.ibm.com/developerworks/java/tutorials/j-jmf/>