2004

# JAVA synchronized collaborative multimedia toolkit: A collaborative communication tool

Rohit Chavan

# JAVA SYNCHRONIZED COLLABORATIVE MULTIMEDIA TOOLKIT, A COLLABORATIVE COMMUNICATION TOOL

---

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Rohit Chavan

March 2004

# JAVA SYNCHRONIZED COLLABORATIVE MULTIMEDIA TOOLKIT, A COLLABORATIVE COMMUNICATION TOOL

---

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Rohit Chavan

March 2004

Approved by:

[redacted]

2/26/04

Dr. Ernesto Gomez
Associate Professor, Computer Science

Date

[redacted]

Dr. Richard Botting
Professor, Computer Science

[redacted]

Dr. George Georgiou
Professor, Computer Science

## ABSTRACT

Since the advent of the Internet, the Computing and Communication industry has progressed very rapidly. It seems certain that in the near future every person no matter where located geographically, will be equipped with some sort of network computing capability, either by means of conventional desktop computing or through information appliances.

Collaborative computing and real-time conferencing is a great way to make developers more effective, increase productivity and teamwork, improve decision making, enabling technical and creative professionals to collaborate.

In this project a collaboration multimedia toolkit, JSCMT (Java Synchronized Collaborative Multimedia Toolkit) was developed which is intended to connect a group of people located in different geographical locations. JSCMT integrates different communication tools like, text based chat, real-time audio-video conferencing and audio chat into one collaborative application. JSCMT is designed as an Java Application and uses JMF API (Java Media Framework API) and JSDT (Java Shared Data Toolkit).

TABLE OF CONTENTS

ABSTRACT ........ iii

LIST OF FIGURES ........ ix

CHAPTER ONE: SOFTWARE REQUIREMENT SPECIFICATIONS

Introduction ........ 1

Purpose of the Project ........ 2

Scope of the Project ........ 3

Limitations of the Project ........ 6

Definitions, Acronyms, and Abbreviations ........ 6

Summary ........ 11

CHAPTER TWO: ARCHITECTURE

Overview ........ 12

Architecture ........ 12

JSCMT Packages ........ 14

Package Client ........ 20

JscmtLogin ........ 20

ClientApplication ........ 21

JscmtClient ........ 24

JscmtDebugFlags ........ 24

JscmtMessageConsumer ........ 25

JscmtSessionListener ........ 26

JscmtChannelListener ........ 27

PrivateMessageHandler ........ 28

PrivateMessageConsumer .................... 29

ReceiverClosedEvent ....................... 30

JscmtReceiverListener ..................... 31

ReceiverClosedListener .................... 31

JscmtReceiverMessenger .................... 32

Package Server ............................ 33

ServerClient .............................. 33

JscmtServer ............................... 33

JscmtSessionManager ....................... 34

JscmtChannelManager ....................... 35

Package Constants ......................... 36

JscmtConstants ............................ 36

Package User .............................. 37

UserTree .................................. 38

UserNode .................................. 39

UserInfo .................................. 40

Package Audiovideo ........................ 41

AudioReceiver ............................. 41

AudioTransmitter .......................... 42

PrivateAudioProcessor ..................... 43

AVReceiver ................................ 44

AVTransmitter ............................. 45

PrivateAVProcessor ........................ 47

WebcamRequestHandler .................... 48

Detailed Design ............................ 49

JSCMT Client Construction ................ 50

Add New User ............................. 52

Remove User .............................. 52

Text Message Broadcast ................... 53

Sending AV Broadcast ..................... 54

Receiving AV Broadcast ................... 56

Sending Audio Broadcast .................. 58

Receiving Audio Broadcast ................ 59

Inviting User for Private AV Conference ... 60

Process Response to Private
AV Invitation ............................ 62

Inviting User for Private
Audio Conference ......................... 64

Process Response to Private
Audio Invitation ......................... 66

Sending View Web Cam Request ............. 67

Viewing User's Web Cam ................... 69

Inviting User to View Web Cam ............ 70

Process Response to View
Web Cam Invitation ....................... 71

Inviting User for Private
Text Messaging ........................... 73

Process Private Text
Messaging Invitation ..................... 74

Summary .................................... 76

CHAPTER THREE: USER MANUAL

Overview .................................... 77

Logging In .................................... 78

Broadcast Text Message .......................... 80

Broadcast Audio-Video .......................... 82

Stop Audio-Video Broadcast ...................... 83

Broadcast Audio .............................. 84

Stop Audio Broadcast .......................... 86

Inviting User for Private
Audio-Video Conference .......................... 86

Close Audio-Video Session ...................... 90

Inviting User for Private Audio Conference ..... 91

Close Audio Session .......................... 95

Viewing Client's Web Cam ...................... 96

Inviting Client to View One's Web Cam .......... 98

Private Text Messaging ........................ 100

Summary .................................... 102

CHAPTER FOUR: CONCLUSIONS AND FUTURE DIRECTIONS

Conclusions .................................. 103

Future Directions .............................. 104

Refining Authentication Process .......... 104

Support for Multiple Sessions ............ 104

Support for File Transfer ................. 105

Support for Multiple Audio-Video Sessions ..................... 105

Summary ...................................... 105

REFERENCES ......................................... 106

LIST OF FIGURES

Figure 1.1 JSCMT Use Case Diagram ................ 5

Figure 2.1 JSCMT State Diagram ..................... 13

Figure 2.2 JSCMT Component Diagram ................. 14

Figure 2.3 JSCMT Packages .......................... 16

Figure 2.4 Package Client Class Diagram ............ 17

Figure 2.5 Package Server Class Diagram ............ 18

Figure 2.6 Package User Class Diagram .............. 18

Figure 2.7 Package Constants Class Diagram ......... 19

Figure 2.8 Package audiovideo Class Diagram ........ 19

Figure 2.9 JscmtLogin .............................. 21

Figure 2.10 ClientApplication ...................... 23

Figure 2.11 JscmtClient ............................ 24

Figure 2.12 JscmtDebugFlags ........................ 25

Figure 2.13 JscmtMessageConsumer ................... 26

Figure 2.14 JscmtSessionListener ................... 27

Figure 2.15 JscmtChannelListener ................... 28

Figure 2.16 PrivateMessageHandler .................. 29

Figure 2.17 PrivateMessageConsumer ................. 30

Figure 2.18 ReceiverClosedEvent .................... 31

Figure 2.19 JscmtReceiverListener .................. 31

Figure 2.20 ReceiverClosedListener ................. 32

Figure 2.21 JscmtReceiverMessenger ................. 32

Figure 2.22 ServerClient .......................... 33

Figure 2.23 JscmtServer .......................... 34

Figure 2.24 JscmtSessionManager .................... 35

Figure 2.25 JscmtChannelManager .................... 36

Figure 2.26 JscmtConstants ......................... 37

Figure 2.27 UserTree ............................... 39

Figure 2.28 UserNode ............................... 40

Figure 2.29 UserInfo ............................... 40

Figure 2.30 AudioReceiver .......................... 42

Figure 2.31 AudioTransmitter ....................... 43

Figure 2.32 PrivateAudioProcessor .................. 44

Figure 2.33 AVReceiver ............................. 45

Figure 2.34 AVTransmitter .......................... 47

Figure 2.35 PrivateAVProcessor ..................... 48

Figure 2.36 WebcamRequestHandler ................... 49

Figure 2.37 JSCMT Client Construction .............. 51

Figure 2.38 Add New User ........................... 52

Figure 2.39 Remove User ............................ 53

Figure 2.40 Text Message Broadcast ................. 54

Figure 2.41 Sending AV Broadcast ................... 56

Figure 2.42 Receiving AV Broadcast ................. 57

Figure 2.43 Sending Audio Broadcast ................ 59

Figure 2.44 Receiving Audio Broadcast .............. 60

Figure 2.45 Private AV Invitation .................. 62

Figure 2.46 Process Private AV Invitation ........... 64

Figure 2.47 Private Audio Invitation ................ 65

Figure 2.48 Process Private Audio Invitation ........ 67

Figure 2.49 Sending Request to View Web Cam ......... 68

Figure 2.50 Viewing User's Web Cam .................. 70

Figure 2.51 Inviting User to View Web Cam ........... 71

Figure 2.52 Process View Web Cam Invitation ......... 73

Figure 2.53 Inviting for Private Text Messaging ..... 74

Figure 2.54 Process Private Text Messaging .......... 76

Figure 3.1 JSCMT Login Window ...................... 79

Figure 3.2 JSCMT Client Graphical User Interface ... 80

Figure 3.3 Broadcast Text Message .................. 81

Figure 3.4 Broadcast Audio-Video ................... 83

Figure 3.5 Stop Audio-Video Broadcast .............. 84

Figure 3.6 Broadcast Audio ......................... 85

Figure 3.7 Stop Audio Broadcast .................... 86

Figure 3.8 Invite for Private AV Conference ........ 88

Figure 3.9 Receive Private AV
Conference Invitation ................... 89

Figure 3.10 Participating in Private
AV Conference ........................... 90

Figure 3.11 Close Audio Video Session .............. 91

Figure 3.12 Invite for Private Audio Conference ..... 93

Figure 3.13 Receive Private Audio Conference Invitation .................... 94

Figure 3.14 Participating in Private Audio Conference ......................... 95

Figure 3.15 Close Audio Session ..................... 96

Figure 3.16 Request to View Client's Web Cam ........ 97

Figure 3.17 Receiver View Web Cam Request ........... 98

Figure 3.18 Invite to View One's Web Cam ............ 99

Figure 3.19 View Inviting Client's Web Cam .......... 100

Figure 3.20 Invite for Private Text Messaging ....... 101

Figure 3.21 Participate In Private Text Messaging ... 102

# CHAPTER ONE

# SOFTWARE REQUIREMENT SPECIFICATIONS

## Introduction

As networks become ubiquitous and more users have a permanent connection, there is an increasing demand for other network services, such as real-time data feeds, group communication, and teleconferencing. Since the advent of the Internet, the Computing and Communication industry has progressed very rapidly. It seems certain that in the near future every person; no matter where located geographically, will be equipped with some sort of network computing capability, either by means of conventional desktop computing or through information appliances. This not only means that geographically distributed people will be able to easily communicate but also collaborate. Collaborative computing and real-time conferencing is a great way to make developers more effective, increase productivity and teamwork, improve decision making, enabling technical and creative professionals to collaborate.

Collaborative application provides platform to widely distributed users to work concurrently on a one big project.

## Purpose of the Project

Multimedia applications are beginning to play an important role in various aspects of our lives, including education, business, healthcare, publishing and entertainment. The recent advance in computing and networking technologies has fueled the emergence of these applications requiring real-time processing and high bandwidth. In order for these applications to be truly useful and effective, they must be able to operate in a distributed fashion, covering users possibly located in geographically distant locations. Availability of framework like Java Media Framework (JMF) API has made it easier to achieve streaming of audio and video over the network in the real-time.

Java Synchronized Collaborative Multimedia Toolkit (JSCMT) is a Java Application, intended to collaborate a group of people not located in the same geographical location but working on the same project. JSCMT allows users to collaborate by using communications tools like, text based chat, real-time audio-video conferencing and audio chat. JSCMT is developed using a platform independent programming language, Java (SDK 1.4), Java Media Framework (JMF) API and Java Shared Data Toolkit (JSDT). Java was chosen as programming language to implement this project with the assumption that users might be using different operating systems on their computers.

## Scope of the Project

JSCMT will aid group or team members from different geographical locations to interact and collaborate using the tools like, Text Chat, Live Voice Chat and Audio-Video Conferencing.

The intended audiences are:

- Software Developers: A typical software company is involved in several different types of projects at the same time. Any software development project is executed in a group of people. Team members who prefer to work from home, which is in fact encouraged these days will use JSCMT to interact with team members.
- Research groups: Usually more than one person collaborates on a research project. Team members of these research groups from distantly located universities will use JSCMT as a virtual meeting place for their peers in other universities.

- Other groups of people from different geographical locations but working on anything of common interest will find JSCMT very useful.

JSCMT has the following functionality for its users:

- Text Based Chat: The following operations related to text based chat are supported
    - Broadcast text messages to all the users
    - Send private text messages
- Audio Chat: The following operations related to audio chat are supported
    - Start audio broadcast to all the users
    - Stop audio broadcast
    - Start private audio session with a user
    - Stop private audio session
- Audio-Video: The following operations related to audio-video are supported
    - Start audio-video broadcast to all the users
    - Stop audio-video broadcast
    - Start private audio-video session with a user
    - Stop private audio-video session
    - View other user's web cam

- Invite user to view your web cam

Supporting the above functionality, the Use Case Diagram is shown in Figure 1

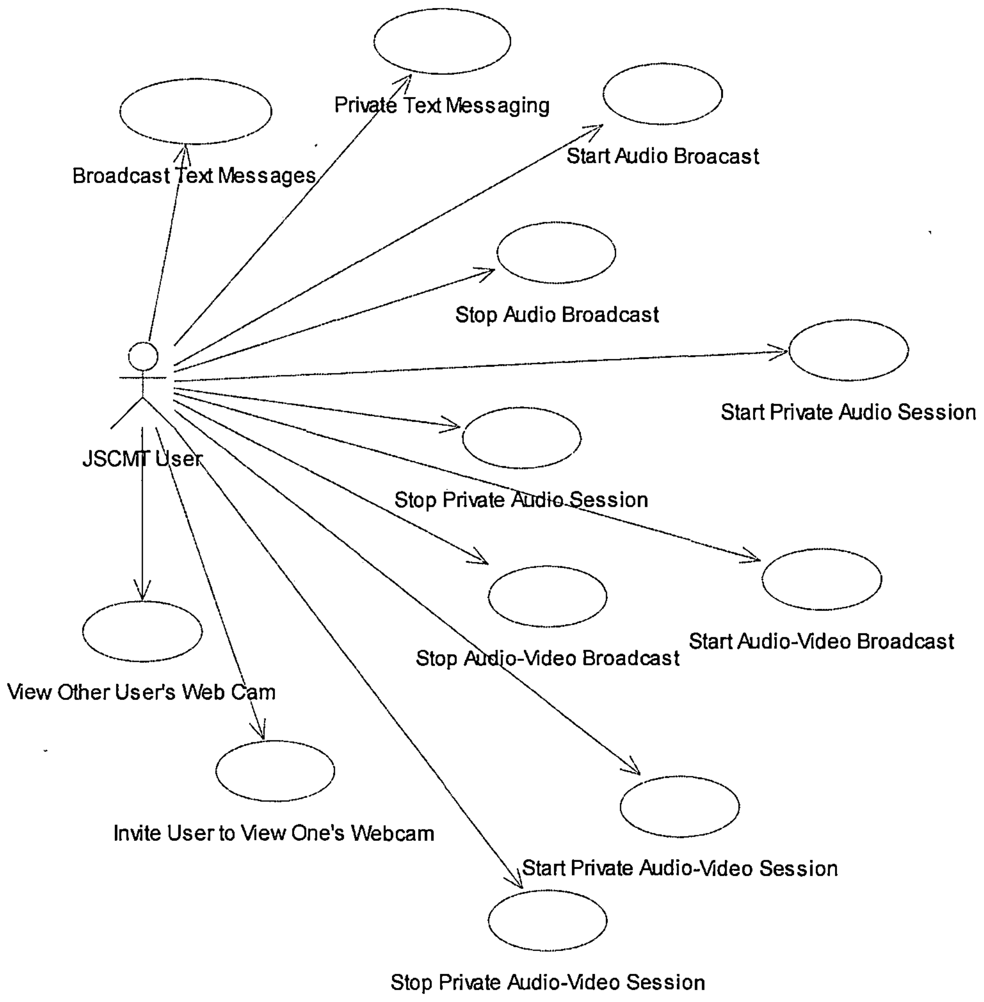

Private Text Messaging
Broadcast Text Messages
Start Audio Broacast
Stop Audio Broadcast
Start Private Audio Session
JSCMT User
Stop Private Audio Session
Start Audio-Video Broadcast
Stop Audio-Video Broadcast
View Other User's Web Cam
Invite User to View One's Webcam
Start Private Audio-Video Session
Stop Private Audio-Video Session

Figure 1.1 JSCMT Use Case Diagram

## Limitations of the Project

JSCMT has following limitations:

- User can participate in only one audio-video session at any given time. If the user is broadcasting audio-video to all the users, he/she can start private audio-video session only after audio-video broadcast is stopped.
- User can participate in only one audio session at any given time.
- JSCMT does not provide any support for transferring files between group members.

## Definitions, Acronyms, and Abbreviations

The definitions, acronyms, and abbreviations used in the document are described in this section.

- API: A set of routine that an application uses to request and carry out lower-level services performed by a computer's operating system. Also a set of calling conventions in programming that define how a service is invoked through the application.
- Channel: A Channel is specific instance of a potentially multi-party communications path

between two or more Clients within a given Session. All Client objects which register an interest in receiving from a given Channel will be given Data sent on that Channel.

- Client: A Client is an object which is part of a JSDT application or applet and is a participant in an instance of multiparty communication.
- Data: Data is a discrete unit of data(array of bytes) that is sent by a Client over a Channel to all of the Clients which have currently registered an interest in receiving data on the given Channel.
- GUI: Graphical User Interface, the graphical representation of physical or pseudo physical objects (such as buttons, labels, textfields) that allow the user to direct the flow of the program through the use of mouse or other pointing device.
- Internet: Internet is a computer network consisting of a worldwide network of computers that use the TCP/IP network protocol to facilitate data transition and exchange.

- Java: Java is object-oriented, cross-platform programming language from Sun Microsystems.
- JMF: Java Media Framework, high level Java API to extend Java support for multimedia development. JMF enables audio, video and other time-based media to be added to application and applets built on Java technology. This optional package, which can capture, playback, stream, and transcode multiple media formats, extends Java 2 Platform, Standard Edition (J2SE) for multimedia developers by providing a powerful toolkit to develop scalable, cross-platform technology
- JSDT: Java Shared Data Toolkit is a toolkit defined to support highly interactive, collaborative applications written in the Java programming language. It provides the basic abstraction of a session (i.e., a group of objects associated with some common communications pattern), and supports full-duplex multipoint communication among an arbitrary number of connected applications entities all over a variety of different types of networks.

- JVM: Java Virtual Machine, it provides run time environment for Java programs.
- Mbone: Internet Multicast Backbone. With Mbone a single packet is sent to an arbitrary number of receivers by replicating the packet within the network at fan-out points along a distribution tree rooted at the packet's source.
- Registry: The information for each Session needs to be kept somewhere that is easily accessable to application. This is where Registry fits in. The Registry can be started either in its own Java runtime environment, or as a thread within the server, on the host that is the server for each JSDT Session or Client.
- RTP: Real-Time Transport Protocol provides end-to-end network delivery services for the transmission of real-time data. RTP is network and transport-protocol independent, though it is often used over UDP. RTP can be used over both unicast and multicast network services.
- Session: A Session is a collection of related Clients which can exchange data via defined

communications paths. The Session maintains the state associated with the collection of clients and their associated communications paths, and may interact with an object which encapsulates a defined session management policy.

- Socket: Socket is an end-point for communication between two machines.
- Swing: Swing is the part of Java Foundation Classes (JFC) that implements a set of GUI components with a pluggable look and feel. The pluggable look and feel lets us design a single set of GUI component that can automatically have the look and feel of any Operating System Platform.
- TCP/IP: Transmission Control Protocol on top of the Internet Protocol provides a reliable, point-to-point communication channel that client-server applications on the internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To

communicate, the client and the server each reads from and writes to socket bound to the connection.

- UML: The Unified Modeling Language is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It simplifies the complex process of software design, making a "blueprint" for construction.

## Summary

This Chapter describes the software requirements for the JSCMT. It covered the purpose of JSCMT, scope and limitation of JSCMT, various technical terms and other basic requirement for the project.

In the next chapter JSCMT architecture is explained in detail and the implementation details are discussed.

# CHAPTER TWO

# ARCHITECTURE

## Overview

This chapter explains JSCMT architecture, JSCMT package structure, and use case realization with the help of sequence diagrams.

## Architecture

JSCMT is implemented as a 2-tier server/client application that does not require extra server resources for running the client. JSCMT is a framework for shared interactive multimedia applications developed using the support of JSDT API and various services that promote interactivity among online users. JSCMT design is based on the replicated architecture in which an instance of each application runs locally at each participant's site and the activity of each user is distributed to all the participants in a conference. The State diagram explaining the complete cycle of request and response processing is shown in Figure 2.1. Component diagram in Figure 2.2 give a brief idea about the JSCMT architecture.

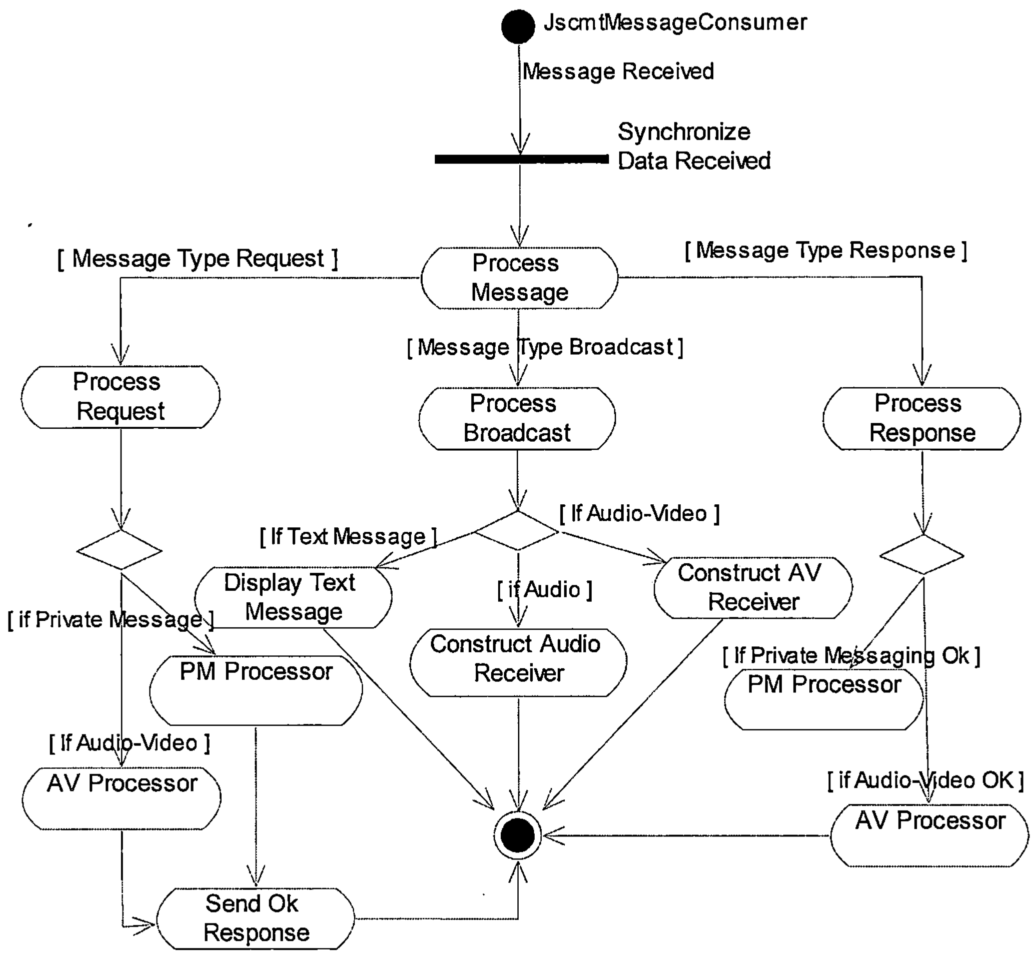


Figure 2.1 JSCMT State Diagram

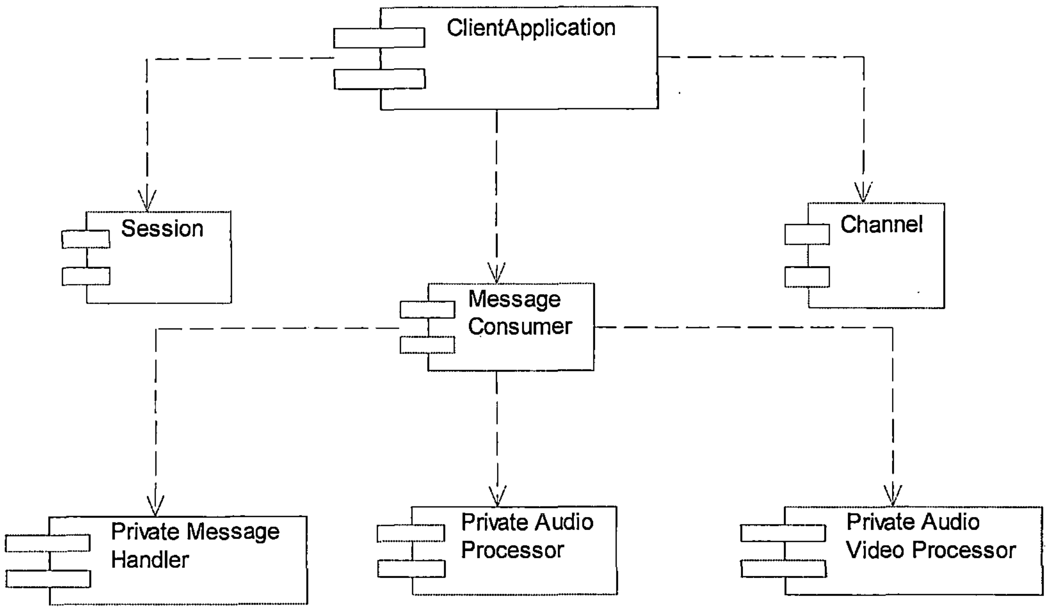


Figure 2.2 JSCMT Component Diagram

## JSCMT Packages

All the classes developed for JSCMT are properly organized in the packages. This helps the maintenance and future development of the project. Different packages used are shown in Figure 2.3. Various classes used in the project are packaged as follows:

- client: This package contains classes used by JSCMT client, which includes classes that are used to handle and process most of the client request and the classes that form the backbone of this collaborative application. These classes are shown in Figure 2.4.

- server: This package contains classes used by JSCMT server which includes classes that are responsible for creating Sessions and Channels which are then joined by the distributed clients. These classes are shown in Figure 2.5.
- user: This package contains classes, which are responsible for storing important information of all the clients and are very useful for most of collaborative activities between clients. These classes are shown in Figure 2.6.
- constants: This package contains the class which caters to all the constants used by other classes of the application. This class is shown in Figure 2.7.
- audiovideo: This package contains classes used by JSCMT clients to handle and process audio-video requests. These classes are shown in Figure 2.8.

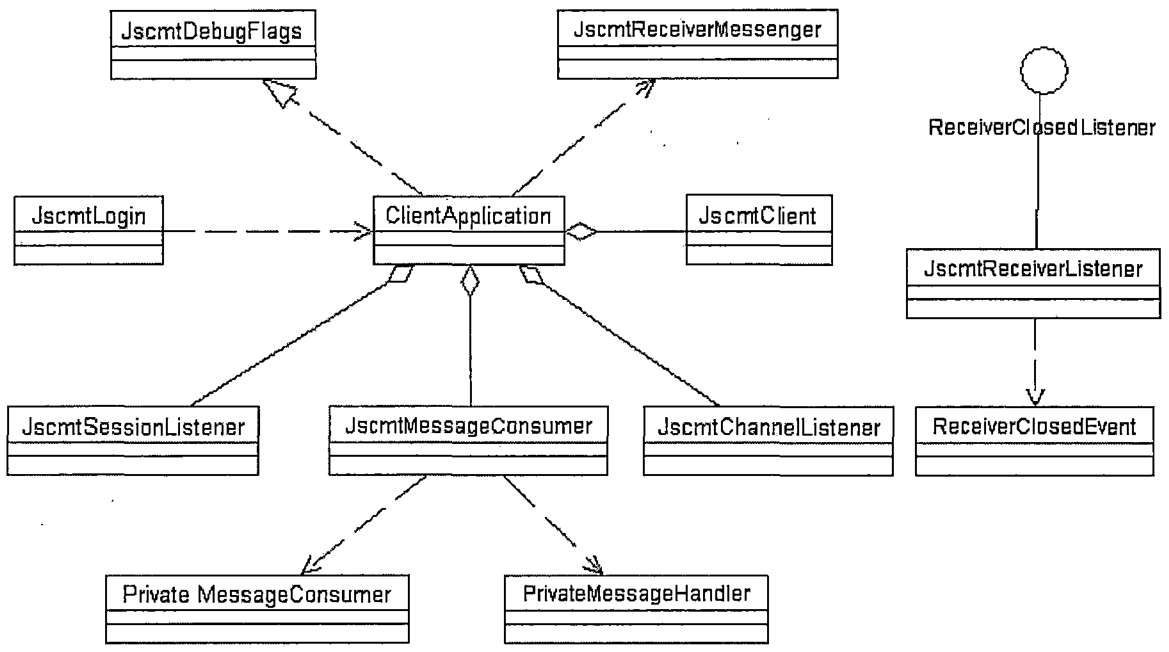


Figure 2.4 Package Client Class Diagram

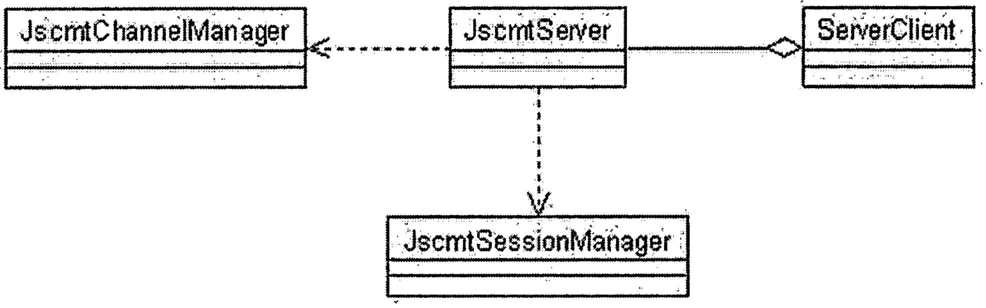


Figure 2.5 Package Server Class Diagram

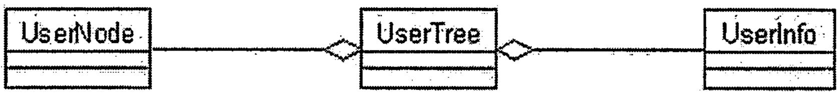


Figure 2.6 Package User Class Diagram

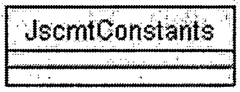


Figure 2.7 Package Constants Class Diagram

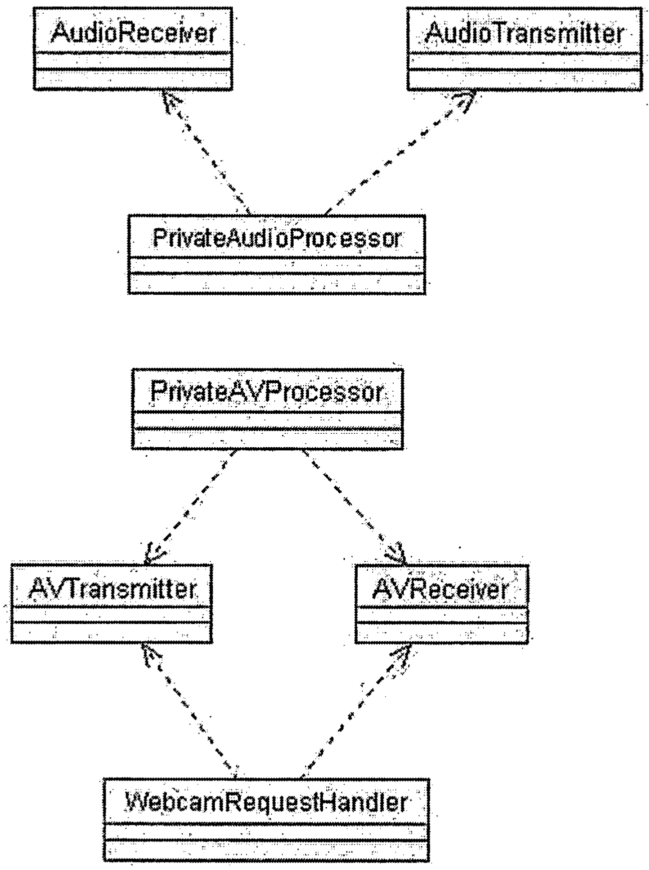


Figure 2.8 Package audiovideo Class Diagram

## Package Client

This package contains classes used by JSCMT client. Classes of this package form the backbone of the collaborative application by supporting various functionalities of the project like, text message broadcast, private text messaging. Classes from this package are responsible for directing client request to it's appropriate request processor.

JscmtLogin

To use the services provided by any collaborative application all the users must be identified by unique username. JscmtLogin class lets JSCMT client to log into the system by accepting the information that is further used to authentic the user. JscmtLogin class takes IP address of the machine where JSCMT server is running and connects the JSCMT client to the server by creating the instance of ClientApplication. JscmtLogin also lets the users to check which collaboration tools they will be using like, web cam, and microphone. JscmtLogin class is shown in Figure 2.9.

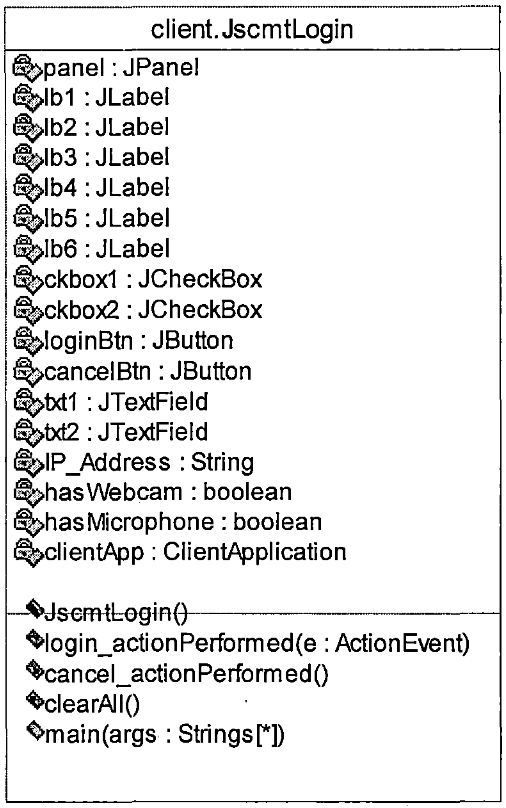
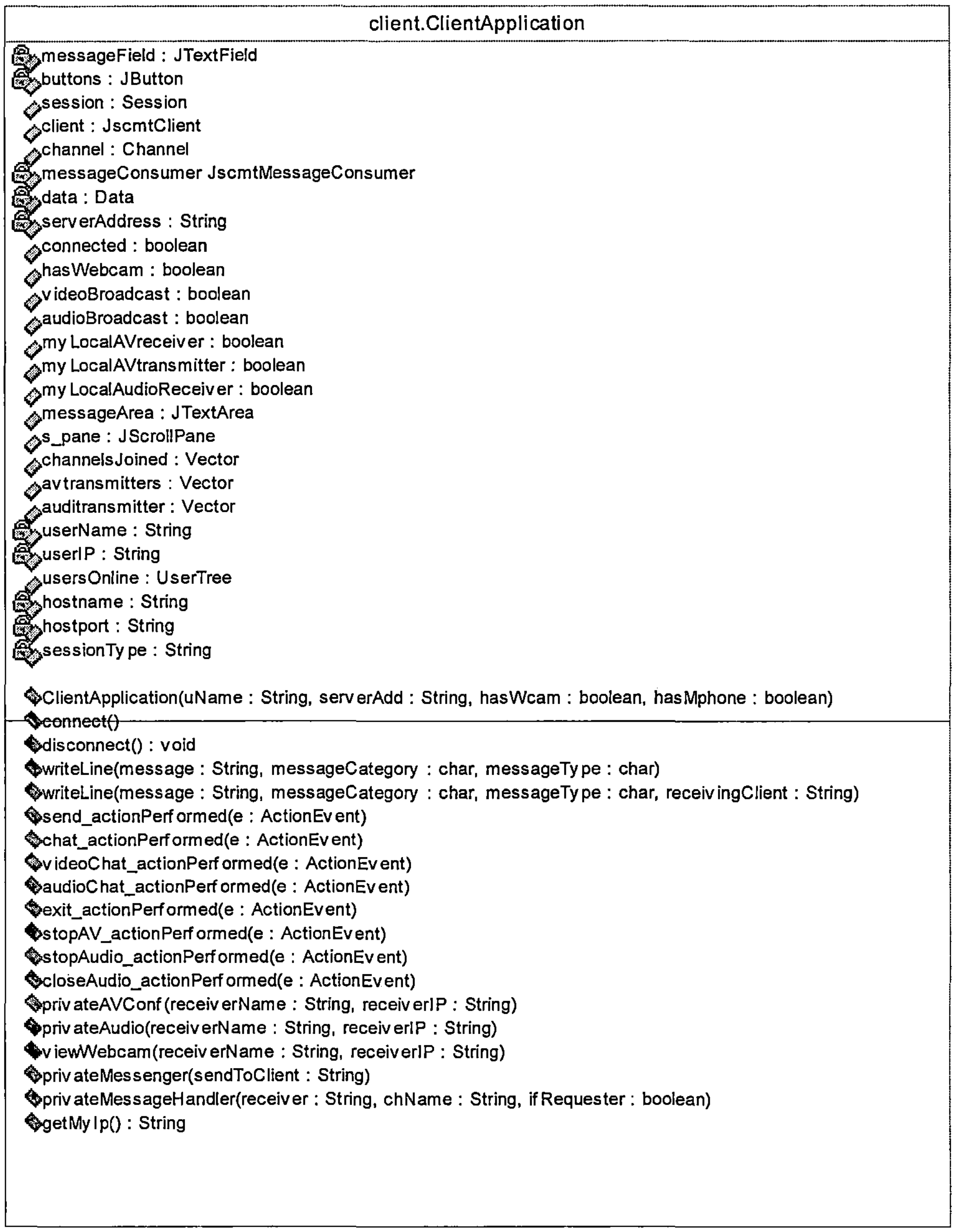


Figure 2.9 JscmtLogin

## ClientApplication

ClientApplication is the most important class of the JSCMT application. ClientApplication class can be considered as a GUI interface to all the services provided by JSCMT application. It creates the instance of JscmtClient which then joins the Session and Channel created by JSCMT server to collaborate with other users. It is responsible for delegating all the user requests like, inviting user for private text messaging, private audio video conferencing, and viewing web cam. ClientApplication also maintains the list of AVTransmitter, AudioTransmitter, and private Channels in the form of Vector. It creates the instance of UserTree which manages the user information. ClientApplication implements JscmtDebugFlags which is useful in debugging. ClientApplication class is shown in Figure 2.10.



```
client.ClientApplication

messageField : JTextField
buttons : JButton
session : Session
client : JscmtClient
channel : Channel
messageConsumer JscmtMessageConsumer
data : Data
serverAddress : String
connected : boolean
hasWebcam : boolean
videoBroadcast : boolean
audioBroadcast : boolean
myLocalAVreceiver : boolean
myLocalAVtransmitter : boolean
myLocalAudioReceiver : boolean
messageArea : JTextArea
s_pane : JScrollPane
channelsJoined : Vector
avtransmitters : Vector
auditransmitter : Vector
userName : String
userIP : String
usersOnline : UserTree
hostname : String
hostport : String
sessionType : String

ClientApplication(uName : String, serverAdd : String, hasWcam : boolean, hasMphone : boolean)
connect()
disconnect() : void
writeLine(message : String, messageCategory : char, messageType : char)
writeLine(message : String, messageCategory : char, messageType : char, receivingClient : String)
send_actionPerformed(e : ActionEvent)
chat_actionPerformed(e : ActionEvent)
videoChat_actionPerformed(e : ActionEvent)
audioChat_actionPerformed(e : ActionEvent)
exit_actionPerformed(e : ActionEvent)
stopAV_actionPerformed(e : ActionEvent)
stopAudio_actionPerformed(e : ActionEvent)
closeAudio_actionPerformed(e : ActionEvent)
privateAVConf(receiverName : String, receiverIP : String)
privateAudio(receiverName : String, receiverIP : String)
viewWebcam(receiverName : String, receiverIP : String)
privateMessenger(sendToClient : String)
privateMessageHandler(receiver : String, chName : String, ifRequester : boolean)
getMyIp() : String
```

Figure 2.10 ClientApplication

JscmtClient

JscmtClient class is an implementation of Client interface of JSDT. It is instantiated by ClientApplication and joins Channel and Session created by the JSCMT server. Authentication action is performed when JscmtClient joins a Channel or Session and when it creates or destroys Channel for private text messaging therefore, JscmtClient implements authenticate method of the Client interface. Unique username that client uses when logging into the system is used by JscmtClient in initializing client name. JscmtClient class is shown in Figure 2.11.

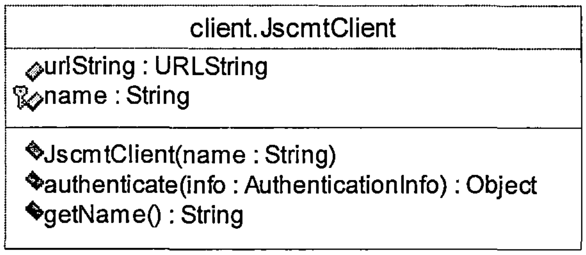

client.JscmtClient
urlString : URLString
name : String
JscmtClient(name : String)
authenticate(info : AuthenticationInfo) : Object
getName() : String

Figure 2.11 JscmtClient

JscmtDebugFlags

JscmtDebugFlags can be considered as a utility class that is used in debugging JSCMT application. It is a collection of Boolean variables. JscmtDebugFlags class is shown in Figure 2.12.

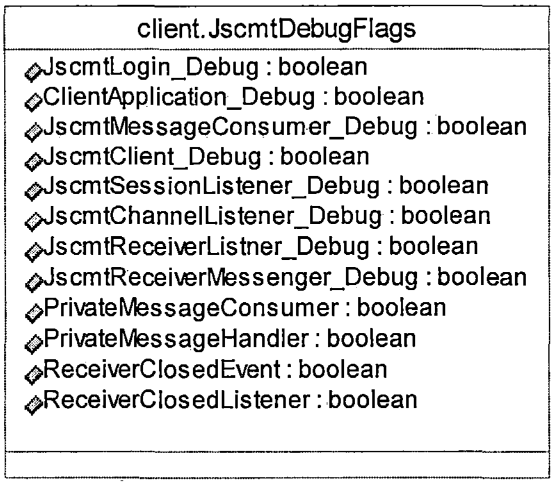
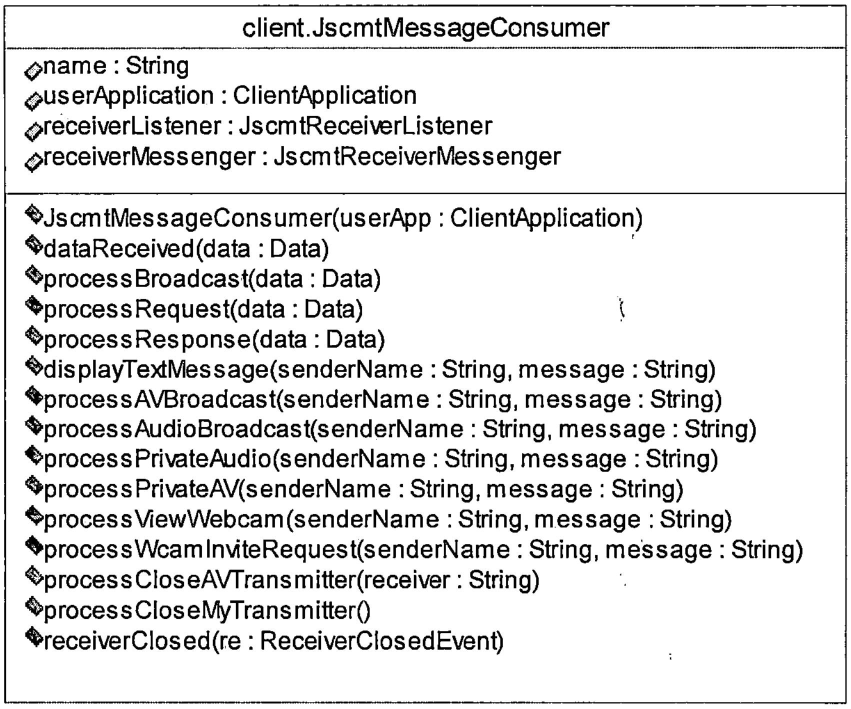

client.JscmtDebugFlags
JscmtLogin_Debug : boolean
ClientApplication_Debug : boolean
JscmtMessageConsumer_Debug : boolean
JscmtClient_Debug : boolean
JscmtSessionListener_Debug : boolean
JscmtChannelListener_Debug : boolean
JscmtReceiverListner_Debug : boolean
JscmtReceiverMessenger_Debug : boolean
PrivateMessageConsumer : boolean
PrivateMessageHandler : boolean
ReceiverClosedEvent : boolean
ReceiverClosedListener : boolean

Figure 2.12 JscmtDebugFlags

JscmtMessageConsumer

JscmtMessageConsumer class is the backbone of JSCMT application. Only one instance of this class is created per JSCMT application. JscmtMessageConsumer implements ChannelConsumer interface of JSDT API. This class consumes messages from the channel that client has joined, processes the message and directs it to the appropriate message handler. JSCMT messages are classified into three different categories i.e. broadcast, request, and response. When a request is received from another client, JscmtMessageConsumer checks if the request can be processed or not and sends appropriate response to the client. JscmtMessageConsumer class is shown in Figure 2.13.



```
client.JscmtMessageConsumer
name : String
userApplication : ClientApplication
receiverListener : JscmtReceiverListener
receiverMessenger : JscmtReceiverMessenger

JscmtMessageConsumer(userApp : ClientApplication)
dataReceived(data : Data)
processBroadcast(data : Data)
processRequest(data : Data)
processResponse(data : Data)
displayTextMessage(senderName : String, message : String)
processAVBroadcast(senderName : String, message : String)
processAudioBroadcast(senderName : String, message : String)
processPrivateAudio(senderName : String, message : String)
processPrivateAV(senderName : String, message : String)
processViewWebcam(senderName : String, message : String)
processWcamInviteRequest(senderName : String, message : String)
processCloseAVTransmitter(receiver : String)
processCloseMyTransmitter()
receiverClosed(re : ReceiverClosedEvent)
```

Figure 2.13 JscmtMessageConsumer

## JscmtSessionListener

JscmtSessionListener class takes care of various events taking place in the Session that JSCMT client has joined. It implements SessionListener interface of JSDT API. When private channel is created to handle private text messaging between two clients SessionEvent is fired which is processed by JscmtSessionListener. JscmtSessionListener class is shown in Figure 2.14.

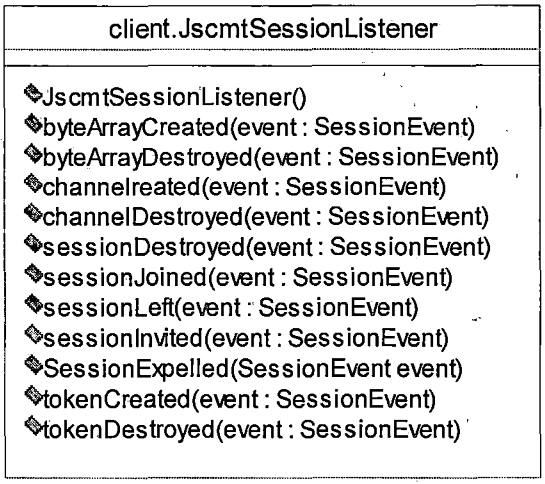

client.JscmtSessionListener
JscmtSessionListener()
byteArrayCreated(event : SessionEvent)
byteArrayDestroyed(event : SessionEvent)
channelreated(event : SessionEvent)
channelDestroyed(event : SessionEvent)
sessionDestroyed(event : SessionEvent)
sessionJoined(event : SessionEvent)
sessionLeft(event : SessionEvent)
sessionInvited(event : SessionEvent)
SessionExpelled(SessionEvent event)
tokenCreated(event : SessionEvent)
tokenDestroyed(event : SessionEvent)

Figure 2.14 JscmtSessionListener

<u>JscmtChannelListener</u>

JscmtChannelListener class handles Channel events by implementing ChannelListener interface of JSDT API. JscmtChannelListener plays important role when JSCMT client joins a Channel and leaves a Channel. When a new client joins or leaves the Channel created by the JSCMT server, ChannelEvent is fired which is processed by all the clients that are registered to receive the event. This assures that every client has an updated list of online users. JscmtChannelListener class is shown in Figure 2.15.

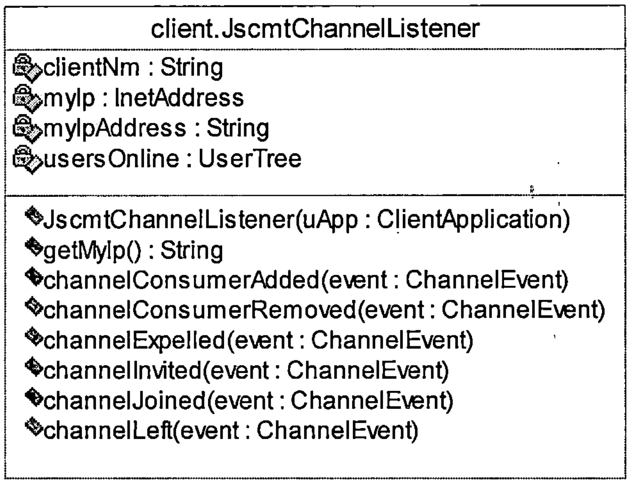

client.JscmtChannelListener
clientNm : String
myIp : InetAddress
myIpAddress : String
usersOnline : UserTree
JscmtChannelListener(uApp : ClientApplication)
getMyIp() : String
channelConsumerAdded(event : ChannelEvent)
channelConsumerRemoved(event : ChannelEvent)
channelExpelled(event : ChannelEvent)
channelInvited(event : ChannelEvent)
channelJoined(event : ChannelEvent)
channelLeft(event : ChannelEvent)

Figure 2.15 JscmtChannelListener

PrivateMessageHandler

PrivateMessageHandler class handles private text messaging between two JSCMT clients. ClientApplication instantiates PrivateMessageHandler object when user selects the option of private messaging. It creates a private Channel and sends request to other client to join the Channel. To consume private messages sent on the private Channel PrivateMessageHandler creates the instance of PrivateMessageConsumer. When the private message window is closed PrivateMessageHandler makes JSCMT client to leave the private channel. PrivateMessageHandler class is shown in Figure 2.16.

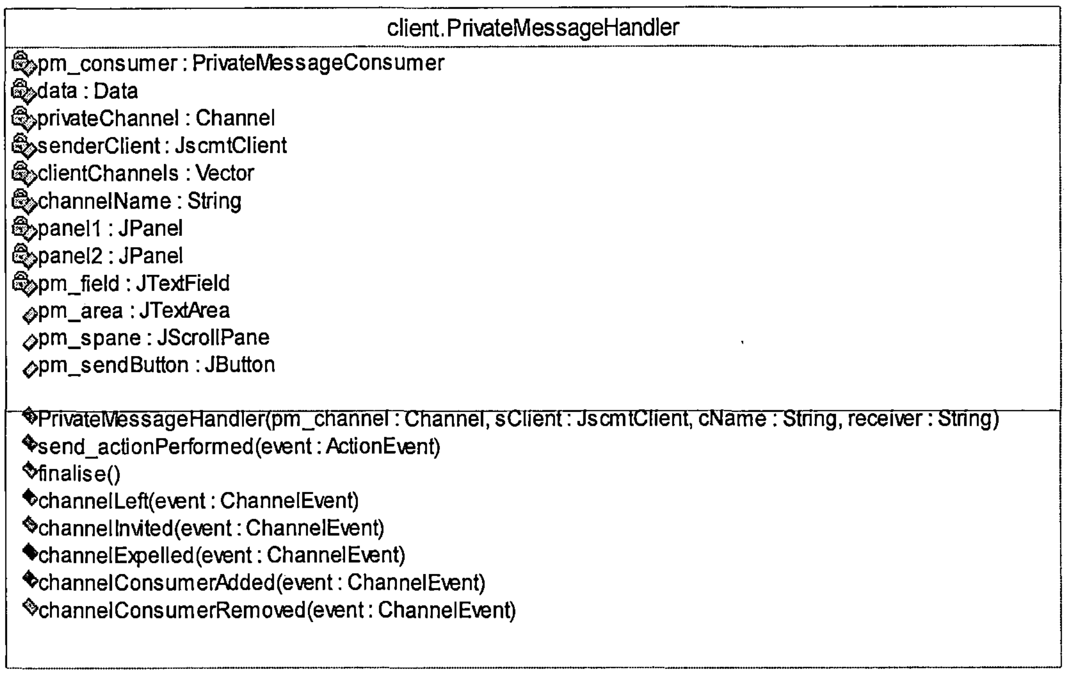

client.PrivateMessageHandler
pm_consumer : PrivateMessageConsumer
data : Data
privateChannel : Channel
senderClient : JscmtClient
clientChannels : Vector
channelName : String
panel1 : JPanel
panel2 : JPanel
pm_field : JTextField
pm_area : JTextArea
pm_spane : JScrollPane
pm_sendButton : JButton
PrivateMessageHandler(pm_channel : Channel, sClient : JscmtClient, cName : String, receiver : String)
send_actionPerformed(event : ActionEvent)
finalise()
channelLeft(event : ChannelEvent)
channelInvited(event : ChannelEvent)
channelExpelled(event : ChannelEvent)
channelConsumerAdded(event : ChannelEvent)
channelConsumerRemoved(event : ChannelEvent)

Figure 2.16 PrivateMessageHandler

PrivateMessageConsumer

PrivateMessageConsumer class consumes messages sent over private Channel created by PrivateMessageHandler between two JSCMT clients. It implements ChannelConsumer interface by defining dataReceived method. The life cycle of PrivateMessageConsumer is defined by PrivateMessageHandler, i.e., when private message window is closed, PrivateMessageConsumer associated with the Client object is removed. PrivateMessageConsumer class is shown in Figure 2.17.

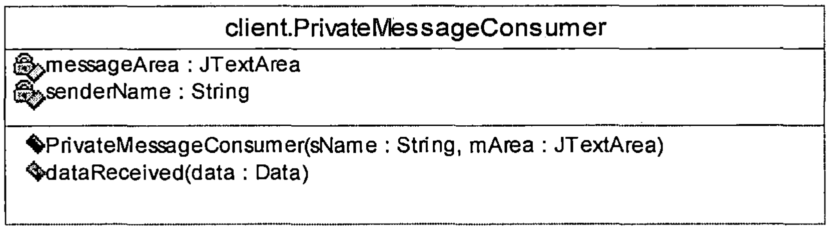

client.PrivateMessageConsumer
messageArea : JTextArea
senderName : String
PrivateMessageConsumer(sName : String, mArea : JTextArea)
dataReceived(data : Data)

Figure 2.17 PrivateMessageConsumer

ReceiverClosedEvent

When a JSCMT client who is transmitting audio-video to one or more clients chooses to close the audio-video transmitter, all the receiving clients needs to be notified about it so that they can close the receiver for the transmitting client. The job of notifying all the receiving clients about this event is done by ReceiverClosedEvent class. This class extends the functionality of EventObject by defining methods that will be useful to process the event by the JSCMT clients. ReceiverClosedEvent class is shown in Figure 2.18.

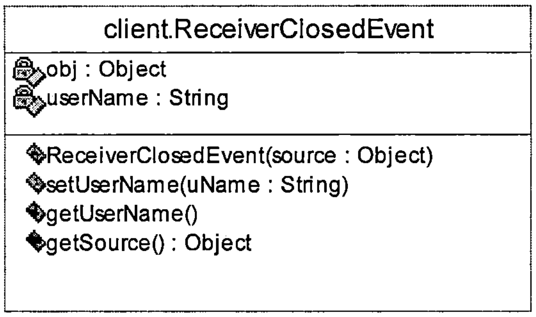

client.ReceiverClosedEvent
obj : Object
userName : String
ReceiverClosedEvent(source : Object)
setUserName(uName : String)
getUserName()
getSource() : Object

Figure 2.18 ReceiverClosedEvent

JscmtReceiverListener

JscmtReceiverListener class implements ReceiverClosedListener interface of JSCMT. It handles ReceiverClosed event which is fired when any of audio-video or audio receiver is closed. JscmtReceiverListener class is shown in Figure 2.19.

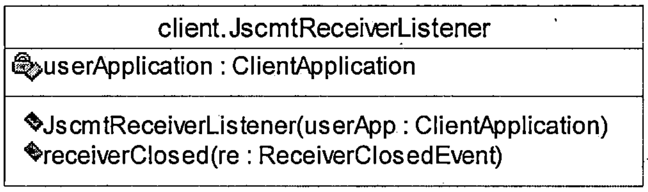

client.JscmtReceiverListener
userApplication : ClientApplication
JscmtReceiverListener(userApp : ClientApplication)
receiverClosed(re : ReceiverClosedEvent)

Figure 2.19 JscmtReceiverListener

ReceiverClosedListener

ReceiverClosedListener is a interface with only one method that is to be defined by the class implementing it. ReceiverClosedListener interface is shown in Figure 2.20.

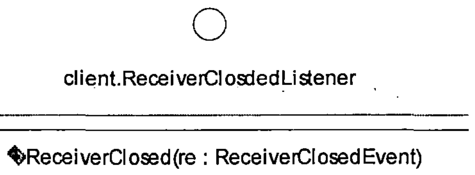

client.ReceiverClosdedListener
ReceiverClosed(re : ReceiverClosedEvent)

Figure 2.20 ReceiverClosedListener

<u>JscmtReceiverMessenger</u>

JscmtReceiverMessenger class maintains the list of objects that have registered to be notified about the ReceiverClosedEvent. This list is maintained by EventListnerList object which is static transient type so that all the JscmtReceiverMessenger have the some copy. When the ReceiverClosedEvent occurs all the ReceiverClosedListener classes in EventListnerList are notified about the event. JscmtReceiverMessenger class is shown in Figure 2.21.

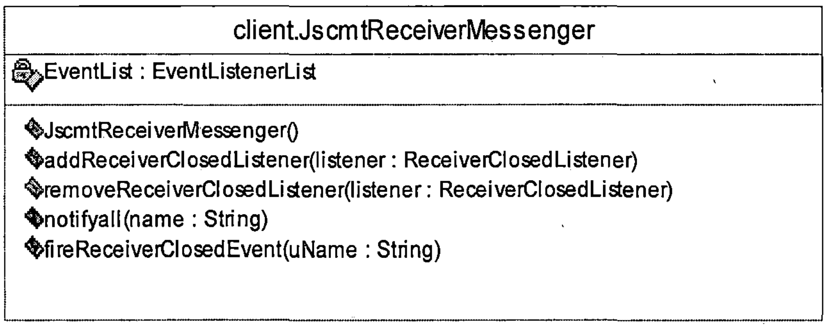

client.JscmtReceiverMessenger
EventList : EventListenerList
JscmtReceiverMessenger()
addReceiverClosedListener(listener : ReceiverClosedListener)
removeReceiverClosedListener(listener : ReceiverClosedListener)
notifyall(name : String)
fireReceiverClosedEvent(uName : String)

Figure 2.21 JscmtReceiverMessenger

## Package Server

This package contains classes used by JSCMT server which includes classes that are responsible for creating Sessions and Channels which are then joined by the distributed clients. Session and Channel manager classes are responsible for authenticating JSCMT clients when they join the Session and Channel respectively.

<u>ServerClient</u>

ServerClient class is an implementation of Client interface of JSDT. It is instantiated by JscmtServer. ServerClient is not required to join the Session and Channel created by JscmtServer which excludes it from explicitly defining authenticate method of the Client interface. ServerClient class is shown in Figure 2.22.

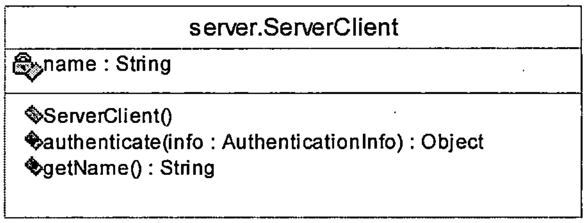


Figure 2.22 ServerClient

<u>JscmtServer</u>

JscmtServer class creates all the basic collaborative components of the JSCMT application. JscmtServer creates socket based Registry in which the Session is then created at the port which is passed as a command line parameter. Channel is then created in the specified Session. JscmtServer associates Session and Channel managers at the time of Session and Channel creation respectively. These managers play important role in monitoring JSCMT client activities. Since ServerClient does not participate in the collaboration, it does not join any Session and Channel created. JscmtServer class is shown in Figure 2.23.

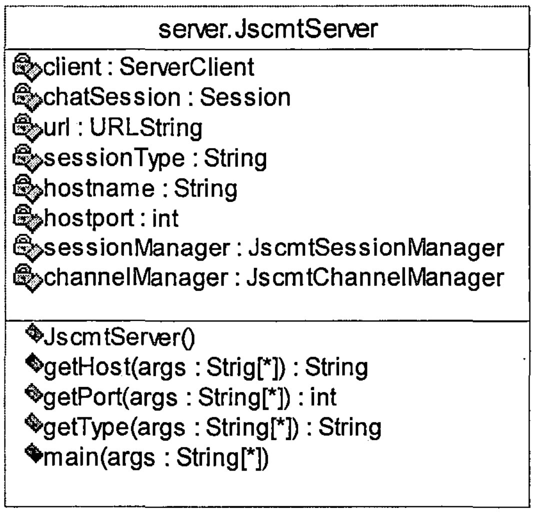

server.JscmtServer
client : ServerClient
chatSession : Session
url : URLString
sessionType : String
hostname : String
hostport : int
sessionManager : JscmtSessionManager
channelManager : JscmtChannelManager
JscmtServer()
getHost(args : Strig[*]) : String
getPort(args : String[*]) : int
getType(args : String[*]) : String
main(args : String[*])

Figure 2.23 JscmtServer

<u>JscmtSessionManager</u>

JscmtSessionManager class is associated with the Session at the time Session creation. It is responsible for authenticating JSCMT client when the client tries to create or destroy a Channel. Whenever JSCMT client is trying to create a new Channel JscmtSessionManager sends a challenge to the client and waits for the response. Client is allowed to create a new Channel only if the authentication of the response is successful. JscmtSessionManager class is shown in Figure 2.24.

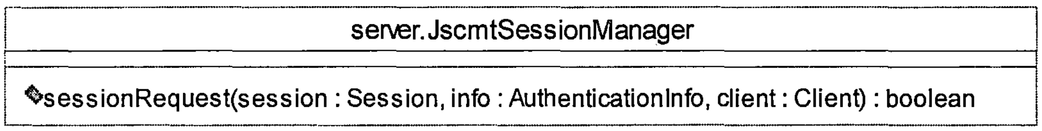

server.JscmtSessionManager
sessionRequest(session : Session, info : AuthenticationInfo, client : Client) : boolean

Figure 2.24 JscmtSessionManager

## JscmtChannelManager

JscmtChannelManager class is associated with the Channel at the time Channel creation. It is responsible for authenticating JSCMT client when the client tries to join or leave a Channel. Whenever JSCMT client is trying to join a Channel JscmtChannelManager sends a challenge to the client and waits for the response. Client is allowed to join a Channel only if the authentication of the response is successful. JscmtChannelManager class is shown in Figure 2.25.

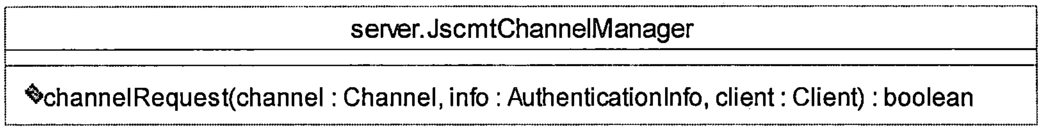

server.JscmtChannelManager
channelRequest(channel : Channel, info : AuthenticationInfo, client : Client) : boolean

Figure 2.25 JscmtChannelManager

## Package Constants

This package contains the class which caters to all the constants used by other classes of the JSCMT application.

### JscmtConstants

JscmtConstants class stores all the constants used by various classes of JSCMT application. JscmtConstants class is shown in Figure 2.26.

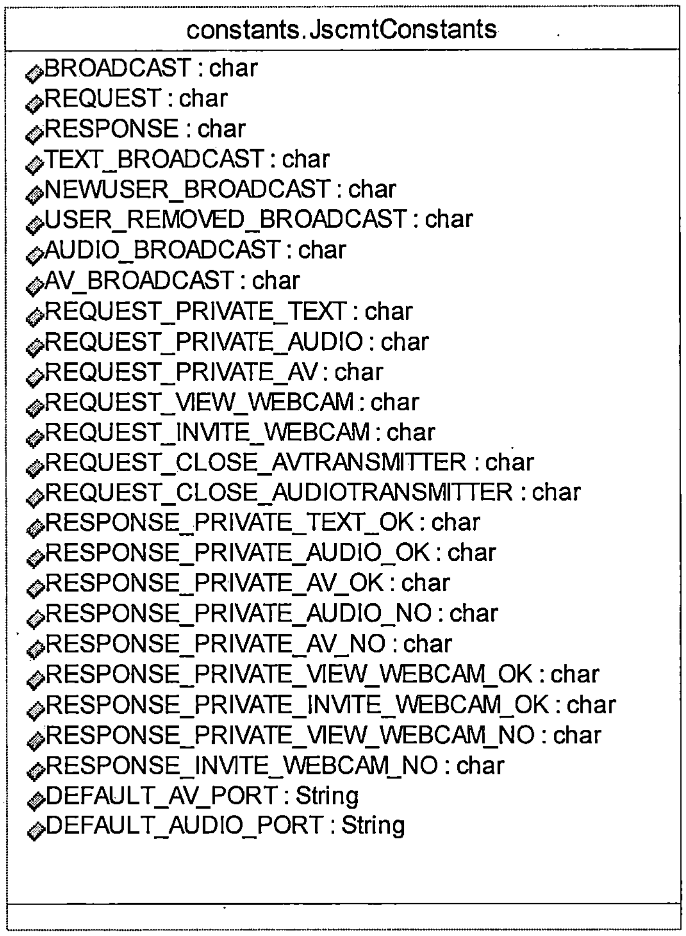

constants.JscmtConstants
BROADCAST : char
REQUEST : char
RESPONSE : char
TEXT_BROADCAST : char
NEWUSER_BROADCAST : char
USER_REMOVED_BROADCAST : char
AUDIO_BROADCAST : char
AV_BROADCAST : char
REQUEST_PRIVATE_TEXT : char
REQUEST_PRIVATE_AUDIO : char
REQUEST_PRIVATE_AV : char
REQUEST_VIEW_WEBCAM : char
REQUEST_INVITE_WEBCAM : char
REQUEST_CLOSE_AVTRANSMITTER : char
REQUEST_CLOSE_AUDIOTRANSMITTER : char
RESPONSE_PRIVATE_TEXT_OK : char
RESPONSE_PRIVATE_AUDIO_OK : char
RESPONSE_PRIVATE_AV_OK : char
RESPONSE_PRIVATE_AUDIO_NO : char
RESPONSE_PRIVATE_AV_NO : char
RESPONSE_PRIVATE_VIEW_WEBCAM_OK : char
RESPONSE_PRIVATE_INVITE_WEBCAM_OK : char
RESPONSE_PRIVATE_VIEW_WEBCAM_NO : char
RESPONSE_INVITE_WEBCAM_NO : char
DEFAULT_AV_PORT : String
DEFAULT_AUDIO_PORT : String

Figure 2.26 JscmtConstants

## Package User

This package contains classes, which are responsible for storing important information of all the clients and are very useful for most of collaborative activities between clients.

## <u>UserTree</u>

JSCMT client has to maintain a current list of all the users logged into the system. UserTree class is used to maintain updated list of JSCMT clients that is done by storing client particular information in the form of a data structure. Taking into consideration graphical display of all the users logged into the system, JTree is used as data structure to store client information. UserTree organizes this information using UserNode and UserInfo objects. As JTree supports pop up menus most of the client requests are primarily handled by UserTree which are then directed to ClientApplication. UserTree class is shown in the Figure 2.27.

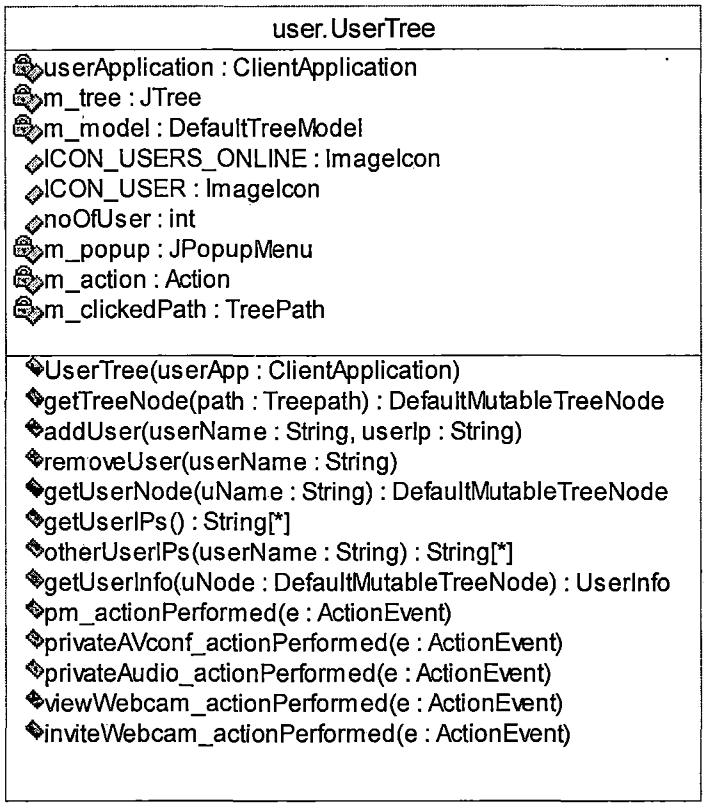

user.UserTree
userApplication : ClientApplication
m_tree : JTree
m_model : DefaultTreeModel
ICON_USERS_ONLINE : ImageIcon
ICON_USER : ImageIcon
noOfUser : int
m_popup : JPopupMenu
m_action : Action
m_clickedPath : TreePath
UserTree(userApp : ClientApplication)
getTreeNode(path : Treepath) : DefaultMutableTreeNode
addUser(userName : String, userIp : String)
removeUser(userName : String)
getUserNode(uName : String) : DefaultMutableTreeNode
getUserIPs() : String[*]
otherUserIPs(userName : String) : String[*]
getUserInfo(uNode : DefaultMutableTreeNode) : UserInfo
pm_actionPerformed(e : ActionEvent)
privateAVconf_actionPerformed(e : ActionEvent)
privateAudio_actionPerformed(e : ActionEvent)
viewWebcam_actionPerformed(e : ActionEvent)
inviteWebcam_actionPerformed(e : ActionEvent)

Figure 2.27 UserTree

## UserNode

UserNode class is the sub component of the data structure that forms a UserTree. When a new client logs into the system a new instance of UserNode is created and added to the UserTree. UserNode class is shown in the Figure 2.28.

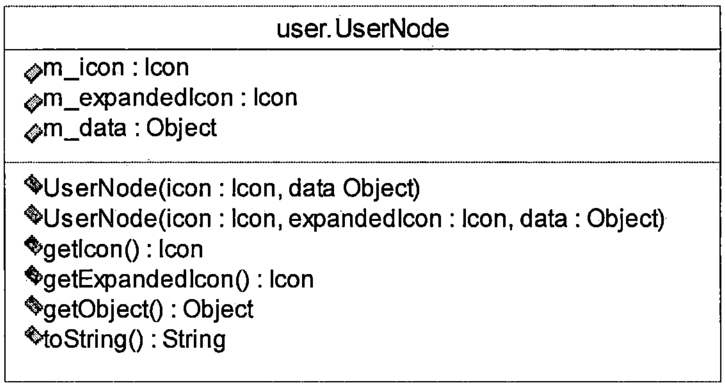


Figure 2.28 UserNode

<u>UserInfo</u>

UserInfo class encapsulates JSCMT client information and forms data component of UserNode. UserInfo stores client's username and IP address. UserInfo class is shown in the Figure 2.29.

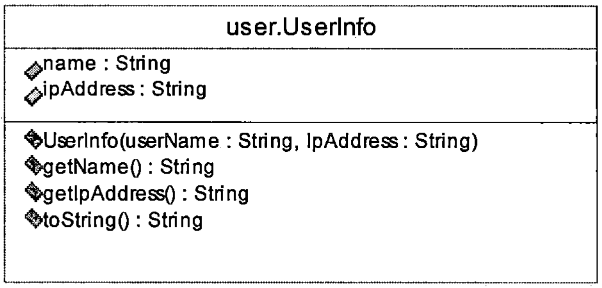


Figure 2.29 UserInfo

## Package Audiovideo

This package contains classes used by JSCMT clients to handle and process audio-video requests.

### AudioReceiver

AudioReceiver class processes all the incoming audio streams from different JSCMT clients. At any given time only one instance of AudioReceiver class exists for a JSCMT application. AudioReceiver runs as a Thread. It opens RTP session at the specified port and listens to the audio streams transmitted at the port. When a new audio stream is received it is checked for the right supported audio format, processed and then finally rendered using Player. When the client transmitting audio stream stops the transmission, ByeEvent is fired and AudiReceiver listening to the event closes the Player that is rendering the received audio stream. AudioReceiver class is shown in Figure 2.30.

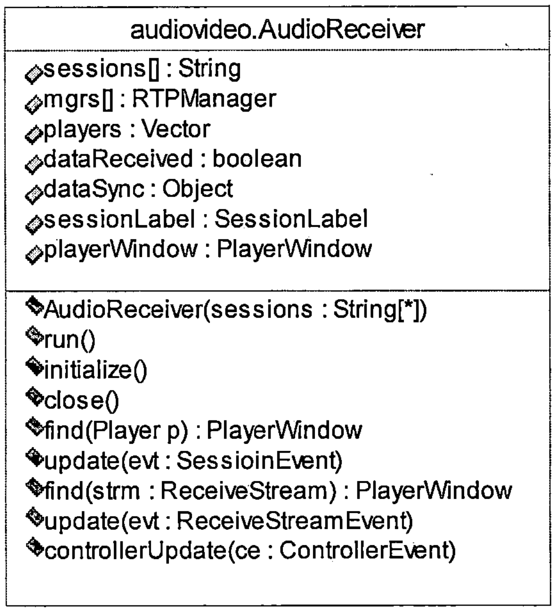
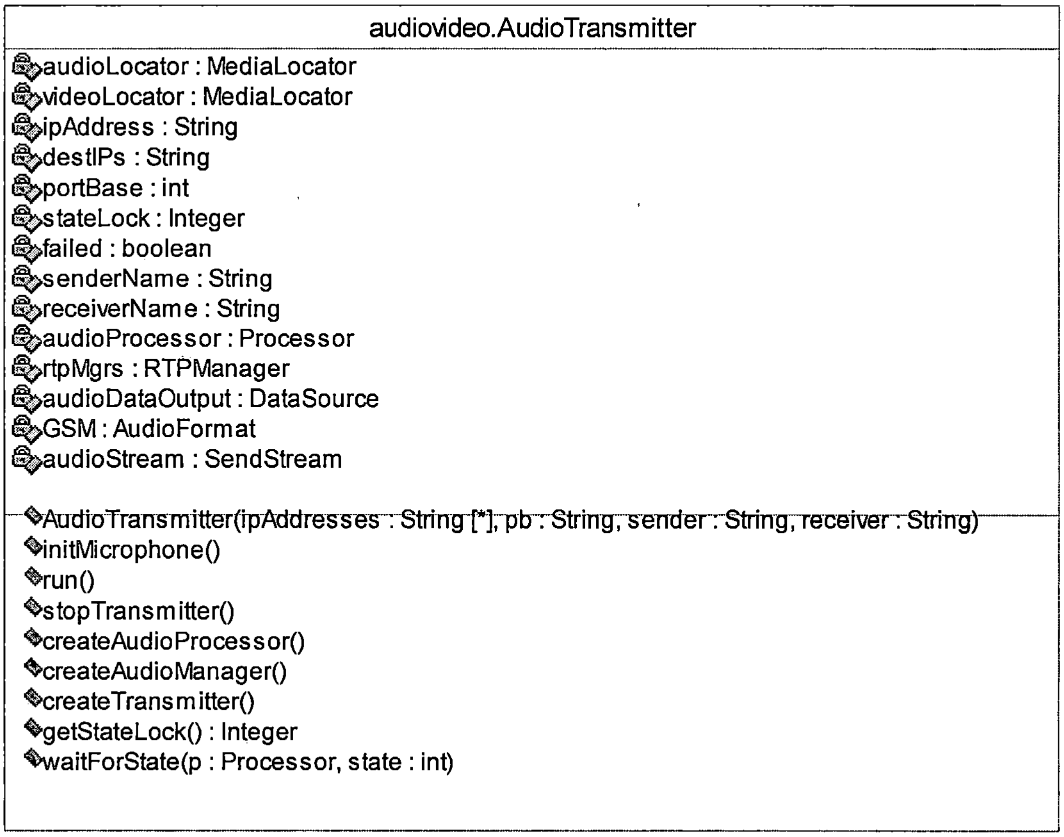

audiovideo.AudioReceiver
sessions[] : String
mgrs[] : RTPManager
players : Vector
dataReceived : boolean
dataSync : Object
sessionLabel : SessionLabel
playerWindow : PlayerWindow
AudioReceiver(sessions : String[*])
run()
initialize()
close()
find(Player p) : PlayerWindow
update(evt : SessioinEvent)
find(strm : ReceiveStream) : PlayerWindow
update(evt : ReceiveStreamEvent)
controllerUpdate(ce : ControllerEvent)

Figure 2.30 AudioReceiver

<u>AudioTransmitter</u>

AudioTransmitter class transmits audio to one or more JSCMT clients. AudioTransmitter runs as a Thread. At any given time only one instance of AudioTransmitter can be created. Instance of AudioTransmitter is created when user chooses to broadcast audio to all the users online or participate in private audio session. It creates RTP session at the specified port. AudioTransmitter initializes the microphone to capture the audio and creates DataSource, which is then used to create Processor. It sets GSM audio format for the audio track to be transmitted. AudioTransmitter is provided with the array of destination

IPs, interested in receiving audio stream, at the time of its creation. AudioTransmitter class is shown in Figure 2.31.



```
audiovideo.AudioTransmitter
audioLocator : MediaLocator
videoLocator : MediaLocator
ipAddress : String
destIPs : String
portBase : int
stateLock : Integer
failed : boolean
senderName : String
receiverName : String
audioProcessor : Processor
rtpMgrs : RTPManager
audioDataOutput : DataSource
GSM : AudioFormat
audioStream : SendStream

AudioTransmitter(ipAddresses : String [*], pb : String, sender : String, receiver : String)
initMicrophone()
run()
stopTransmitter()
createAudioProcessor()
createAudioManager()
createTransmitter()
getStateLock() : Integer
waitForState(p : Processor, state : int)
```

Figure 2.31 AudioTransmitter

<u>PrivateAudioProcessor</u>

When JSCMT client invites another client for private audio session, the request is processed by PrivateAudioProcessor class. PrivateAudioProcessor instantiates AudioTransmitter that transmits Real-time audio captured by client's microphone. It also instantiates AVReceiver if it does not exists to receive Real-time audio transmitted by another client. PrivateAudioProcessor class is shown in Figure 2.32.

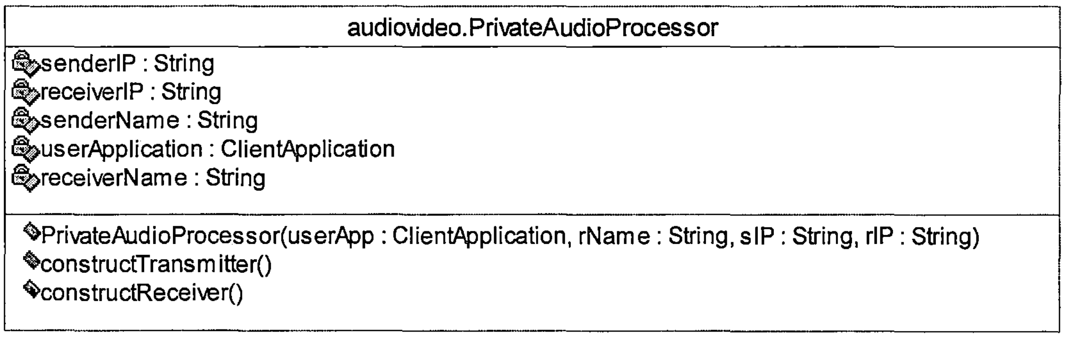
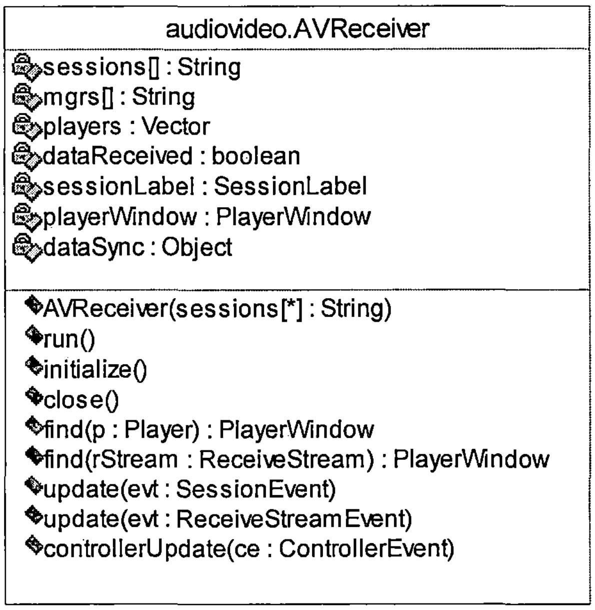

audiovideo.PrivateAudioProcessor
senderIP : String
receiverIP : String
senderName : String
userApplication : ClientApplication
receiverName : String
PrivateAudioProcessor(userApp : ClientApplication, rName : String, sIP : String, rIP : String)
constructTransmitter()
constructReceiver()

Figure 2.32 PrivateAudioProcessor

<u>AVReceiver</u>

AVReceiver class processes all the incoming audio and video streams from different JSCMT clients. At any given time only one instance of AVReceiver class exists for a JSCMT application. AVReceiver runs as a Thread. It opens RTP sessions for audio and video at the specified ports and listens to the audio and video streams transmitted at those ports. When a new audio or video stream is received it is checked for the right supported formats, processed and then finally rendered using Player. When the client transmitting audio and video streams stops the transmission ByeEvent is fired and AVReceiver listening to the event closes the Player that is rendering the received audio and video streams. AVReceiver class is shown in Figure 2.33.

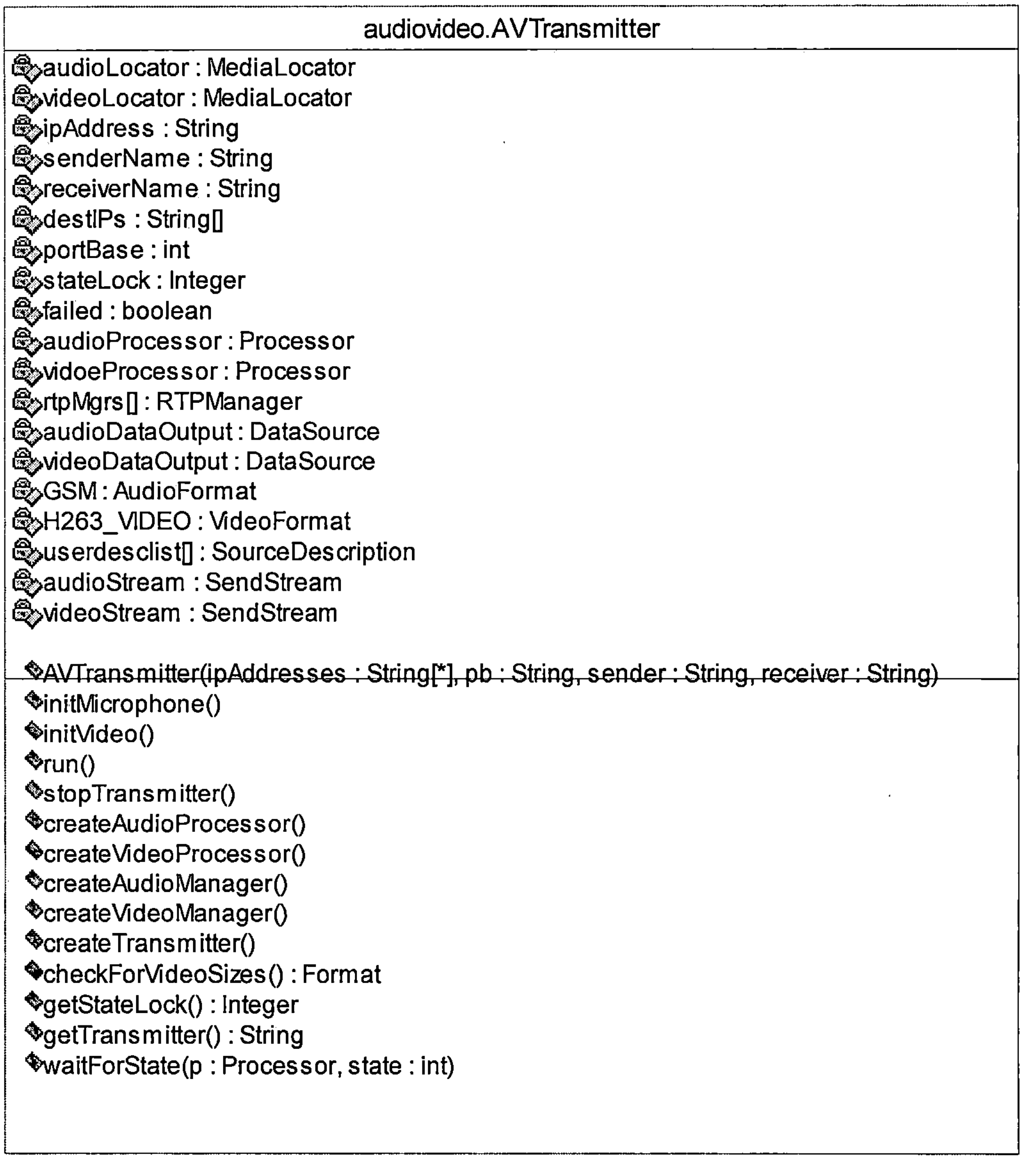


audiovideo.AVReceiver
sessions[] : String
mgrs[] : String
players : Vector
dataReceived : boolean
sessionLabel : SessionLabel
playerWindow : PlayerWindow
dataSync : Object
AVReceiver(sessions[*] : String)
run()
initialize()
close()
find(p : Player) : PlayerWindow
find(rStream : ReceiveStream) : PlayerWindow
update(evt : SessionEvent)
update(evt : ReceiveStreamEvent)
controllerUpdate(ce : ControllerEvent)

Figure 2.33 AVReceiver

<u>AVTransmitter</u>

AVTransmitter class transmits audio and video to one or more JSCMT clients. AVTransmitter runs as a Thread. At any given time only one instance of AVTransmitter can be created. Instance of AVTransmitter is created when user chooses to broadcast audio-video to all the users online or participate in private audio-video session. It creates RTP session at the specified port. AudioTransmitter initializes microphone to capture audio and web cam to capture video, to create audio and video data sources respectively. These data sources are used to create audio and video Processor. It sets GSM audio format for audio track and H263_RTP video format for video track. AVTransmitter is provided with the array of destination IPs, interested in receiving audio and video streams, at the time of its creation. AVTransmitter class is shown in Figure 2.34.



audiovideo.AVTransmitter

```
audioLocator : MediaLocator
videoLocator : MediaLocator
ipAddress : String
senderName : String
receiverName : String
destIPs : String[]
portBase : int
stateLock : Integer
failed : boolean
audioProcessor : Processor
vidoeProcessor : Processor
rtpMgrs[] : RTPManager
audioDataOutput : DataSource
videoDataOutput : DataSource
GSM : AudioFormat
H263_VIDEO : VideoFormat
userdesclist[] : SourceDescription
audioStream : SendStream
videoStream : SendStream

AVTransmitter(ipAddresses : String[*], pb : String, sender : String, receiver : String)
initMicrophone()
initVideo()
run()
stopTransmitter()
createAudioProcessor()
createVideoProcessor()
createAudioManager()
createVideoManager()
createTransmitter()
checkForVideoSizes() : Format
getStateLock() : Integer
getTransmitter() : String
waitForState(p : Processor, state : int)
```

Figure 2.34 AVTransmitter

<u>PrivateAVProcessor</u>

When JSCMT client invites another client for private audio-video session the request is processed by PrivateAVProcessor class. PrivateAVProcessor instantiates

AVTransmitter that transmits Real-time audio and video. It also instantiates AVReceiver if it does not exists to receive Real-time audio and video transmitted by another client. PrivateAVProcessor class is shown in Figure 2.35.

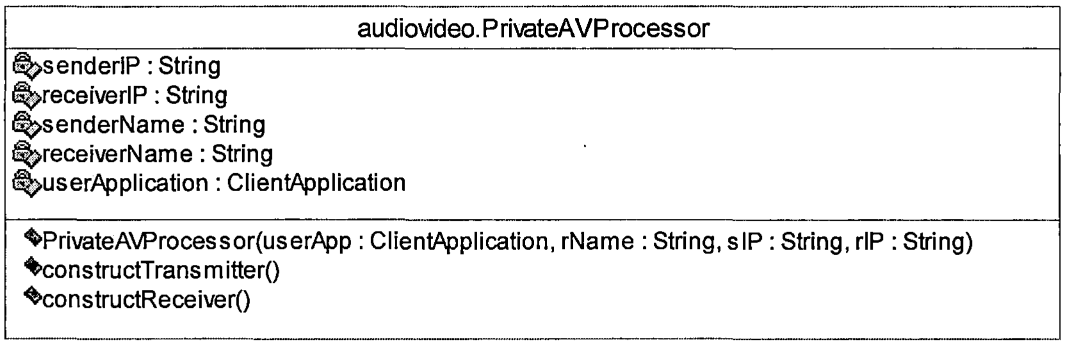

audiovideo.PrivateAVProcessor
senderIP : String
receiverIP : String
senderName : String
receiverName : String
userApplication : ClientApplication
PrivateAVProcessor(userApp : ClientApplication, rName : String, sIP : String, rIP : String)
constructTransmitter()
constructReceiver()

Figure 2.35 PrivateAVProcessor

## WebcamRequestHandler

When JSCMT client invites another client to its web cam the request is handled by WebcamRequestHandler class. WebcamRequestHandler class is shown in Figure 2.36.

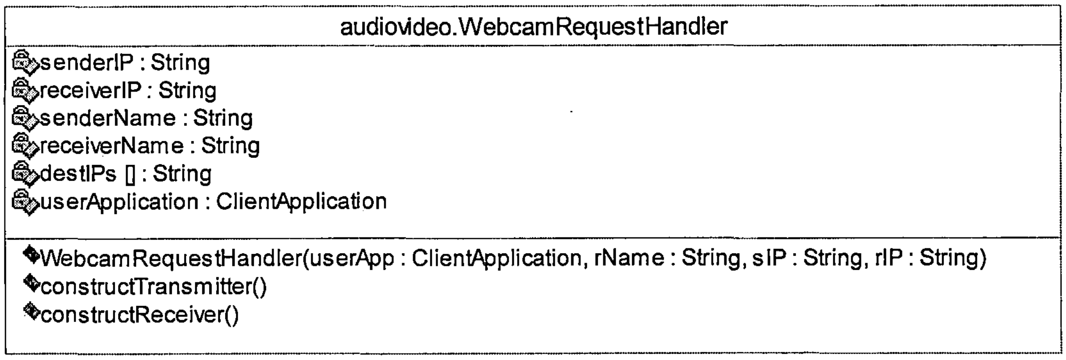

audiovideo.WebcamRequestHandler
senderIP : String
receiverIP : String
senderName : String
receiverName : String
destIPs [] : String
userApplication : ClientApplication
WebcamRequestHandler(userApp : ClientApplication, rName : String, sIP : String, rIP : String)
constructTransmitter()
constructReceiver()

Figure 2.36 WebcamRequestHandler

## Detailed Design

Detailed design describes various functionalities of the application by defining interaction between different smaller components of the application. Detail design discusses application design by describing the structure to be used via narrative, tables, flow charts, etc. Sequence diagram is one of the most widely used software design tools in the process of detailed design.

Functionality of JSCMT application can be explained by applying a scenario and narrating the interaction between various components of JSCMT application. Different scenarios were developed to cover various collaborative functionalities and sequence diagrams were used to describe the data transferred between different JSCMT units.

## JSCMT Client Construction

JSCMT client construction is most important process of the JSCMT system. Several components are constructed during client's initialization. When JscmtLogin's "Login" button is clicked instance of ClientApplication is created. It uses server IP address and port number to connect to server. ClientApplication creates instance of JscmtClient that is authenticated by SessionManager and ChannelManager on the server side, when it joins the Session and Channel respectively. JscmtMessageConsumer is created as the part of client construction process which is responsible for handling and processing most of the client requests. Sequence diagram explaining JSCMT client construction is shown in Figure 2.37.

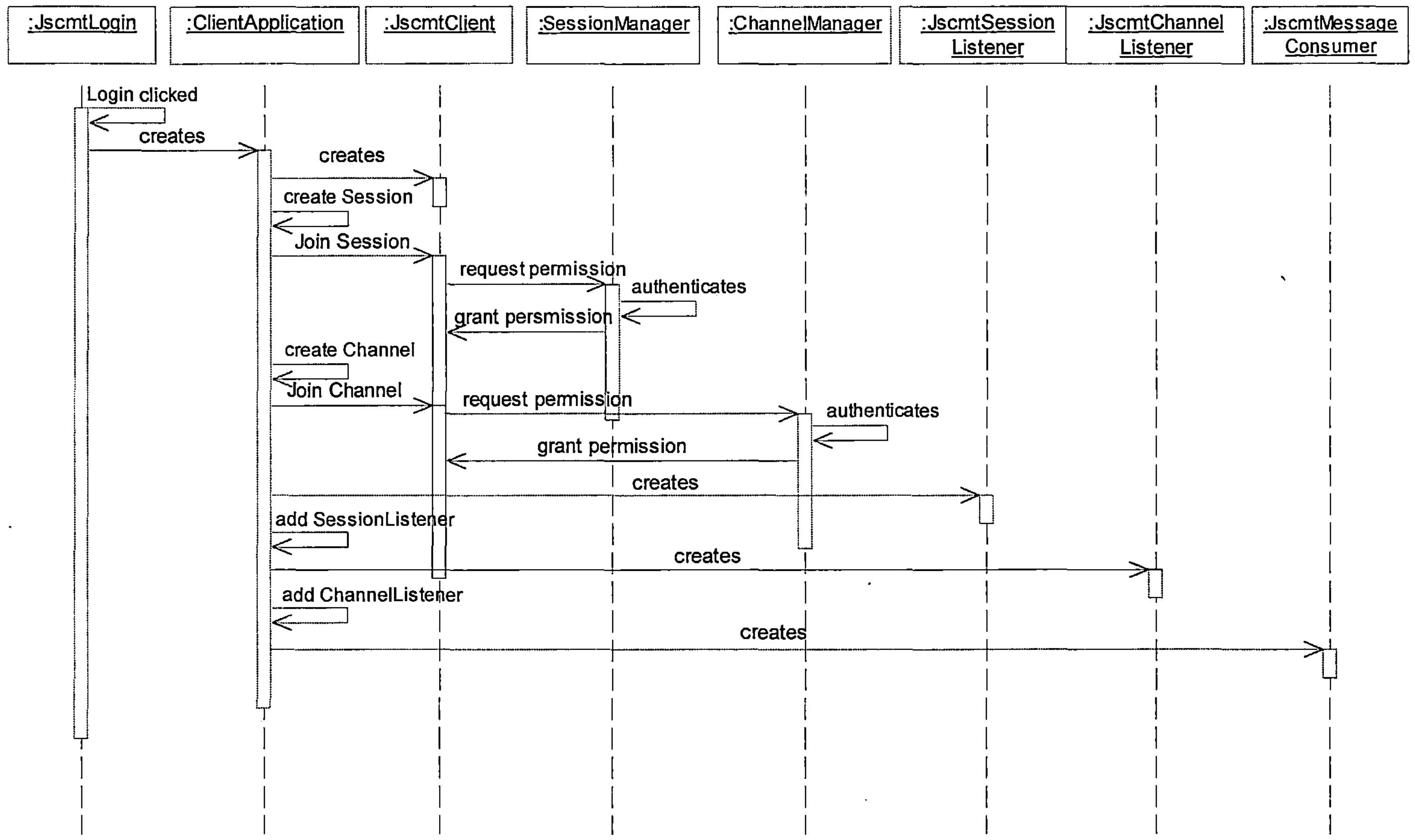


Figure 2.37 JSCMT Client Construction

## Add New User

When a new user joins the JSCMT application it broadcasts "New User" message to all the other users online. JscmtMessageConsumer receives this message and sends new user information to its instance of UserTree which creates a new UserNode. UserNode stores new user information in UserInfo object and is added to the UserTree. Updated UserTree is displayed in the online user display area. Sequence diagram that explains add new user scenario is shown in Figure 2.38.

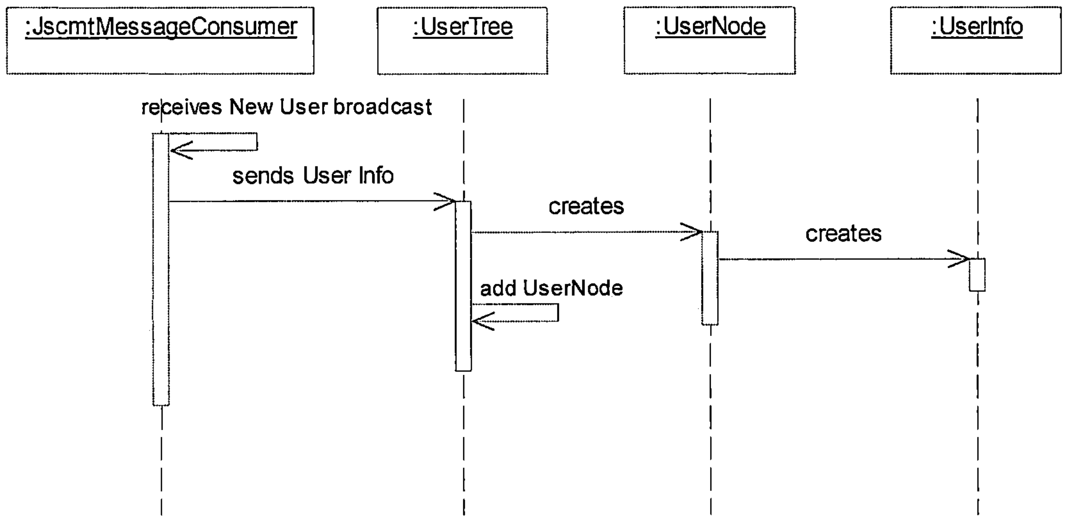


Figure 2.38 Add New User

## Remove User

When JSCMT client leaves the application all the clients logged into the system receive

ChannelConsumerRemoved event. Client's UserTree locates the UserNode for the user to be removed and is deleted. Updated UserTree is displayed in the online users display area. Removing user is shown in Figure 2.39.

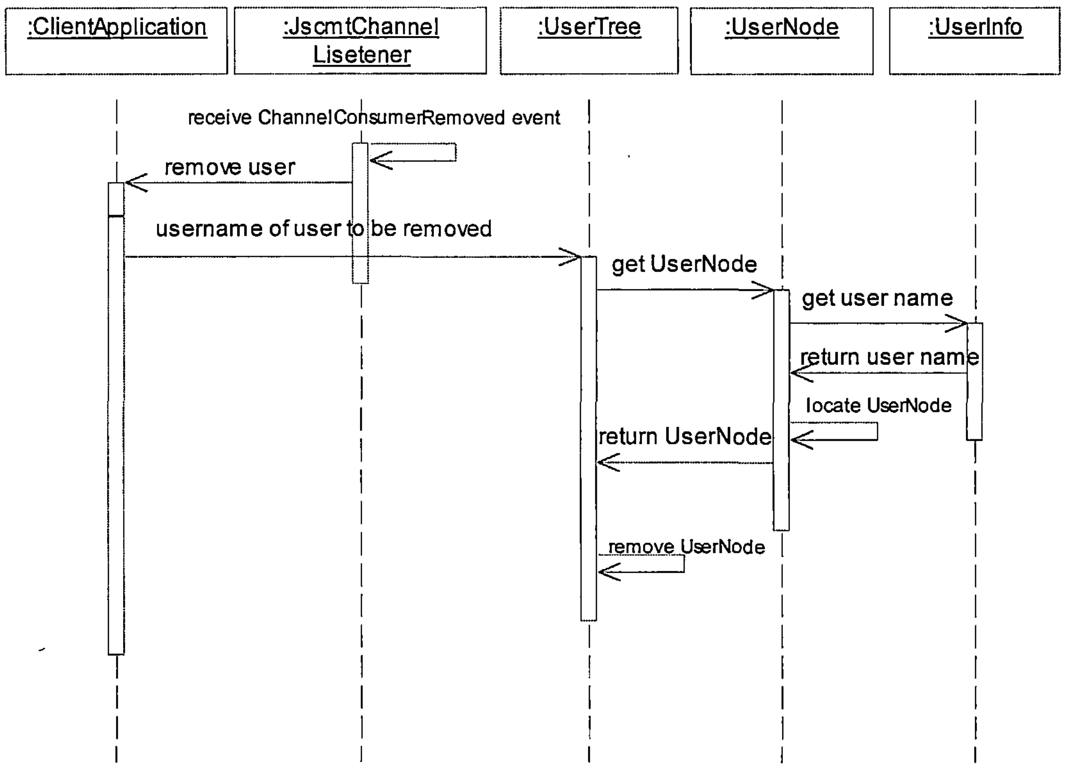


Figure 2.39 Remove User

## Text Message Broadcast

When JSCMT client clicks ClientApplication's "Send" button, txtFiled.getText() method grabs the message typed in the message text field and broadcasts it to all the other clients online. This broadcast message is received by

JscmtMessageConsumers of all the receiving clients. Received message is then displayed by ClientApplication in its display area. Sequence diagram explaining text broadcast scenario is shown in Figure 2.40.

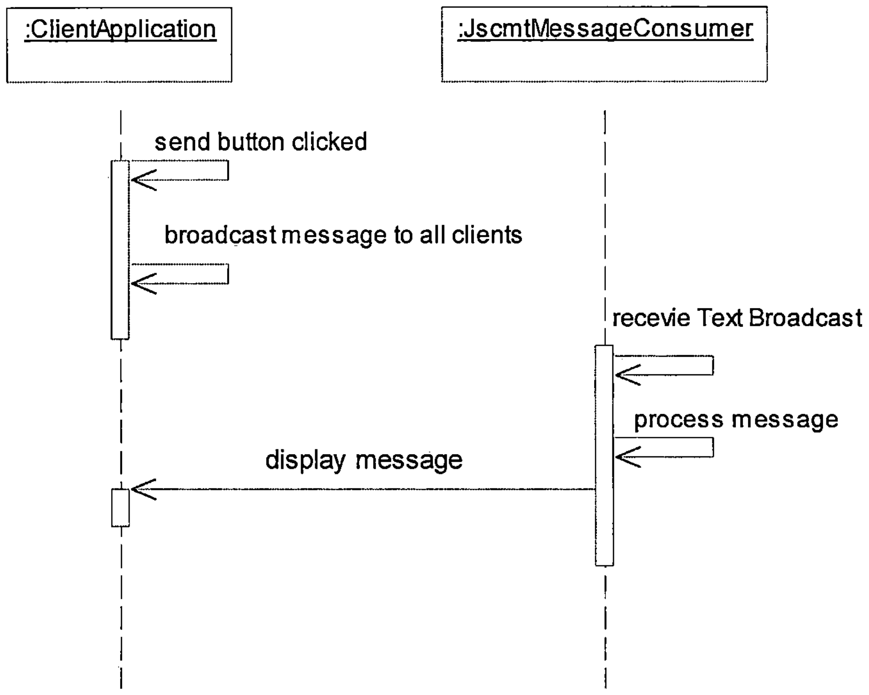


Figure 2.40 Text Message Broadcast

Sending AV Broadcast

Upon clicking ClientApplication's "Video" button check is performed if the client is already broadcasting audio-video. If the client is not broadcasting audio-video and has web cam and microphone dialogue box pops up on the screen to confirm the audio-video broadcast action. Upon action confirmation ClientApplication requests list of IP addresses of online users from UserTree. UserTree processes the request and sends array of IPs to ClientApplication. This array of IPs is send to AVTransmitter at it's time of creation which is used to form list of targets to receive audio and video streams transmitted by the client. Sending video broadcast is explained using sequence diagram shown in Figure 2.41.

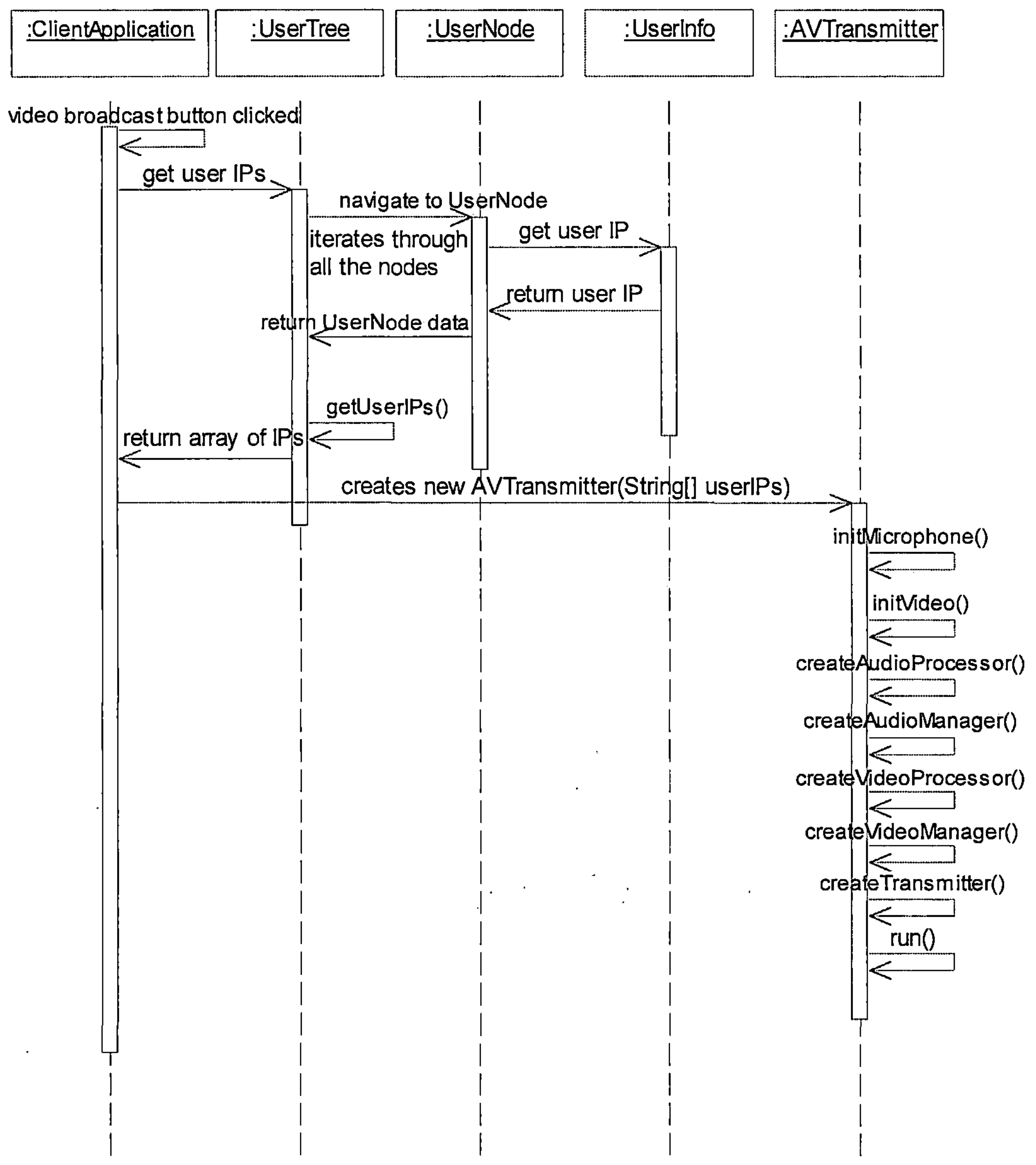


Figure 2.41 Sending AV Broadcast

## Receiving AV Broadcast

JscmtMessageConsumer receives AV broadcast message. It checks with Client if AVReceiver exists. This is done to assure that at any given time only one instance for AVReceiver exists. If AVReceiver does not exist, JscmtMessageConsumer creates an instance of AVReceiver. AVReceiver creates RTP sessions for audio and video at different ports and listens to new incoming audio and video streams at these ports. When a new audio or video stream is received, AVReceiver creates a player to render the stream. Sequence diagram for receiving AV broadcast is shown in Figure 2.42.

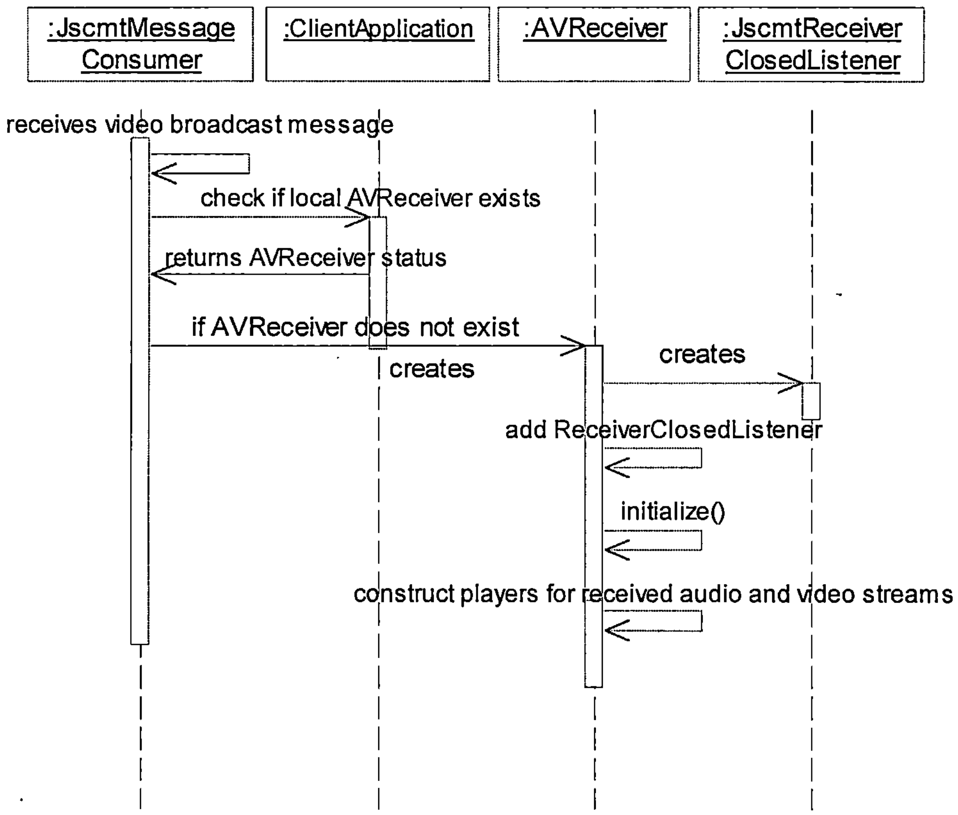

:JscmtMessage Consumer
:ClientApplication
:AVReceiver
:JscmtReceiver ClosedListener
receives video broadcast message
check if local AVReceiver exists
returns AVReceiver status
if AVReceiver does not exist
creates
creates
add ReceiverClosedListener
initialize()
construct players for received audio and video streams

Figure 2.42 Receiving AV Broadcast

## Sending Audio Broadcast

Upon clicking ClientApplication's "Audio" button check is performed if the client is already broadcasting audio. If the client is not broadcasting audio microphone dialogue box pops up on the screen to confirm the audio broadcast action. Upon action confirmation ClientApplication requests list of IP addresses of online users from UserTree. UserTree processes the request and sends array of IPs to ClientApplication. This array of IPs is send to AudioTransmitter at it's time of creation which is used to form list of targets to receive audio streams transmitted by the client. Sending audio broadcast is explained using sequence diagram shown in Figure 2.43.

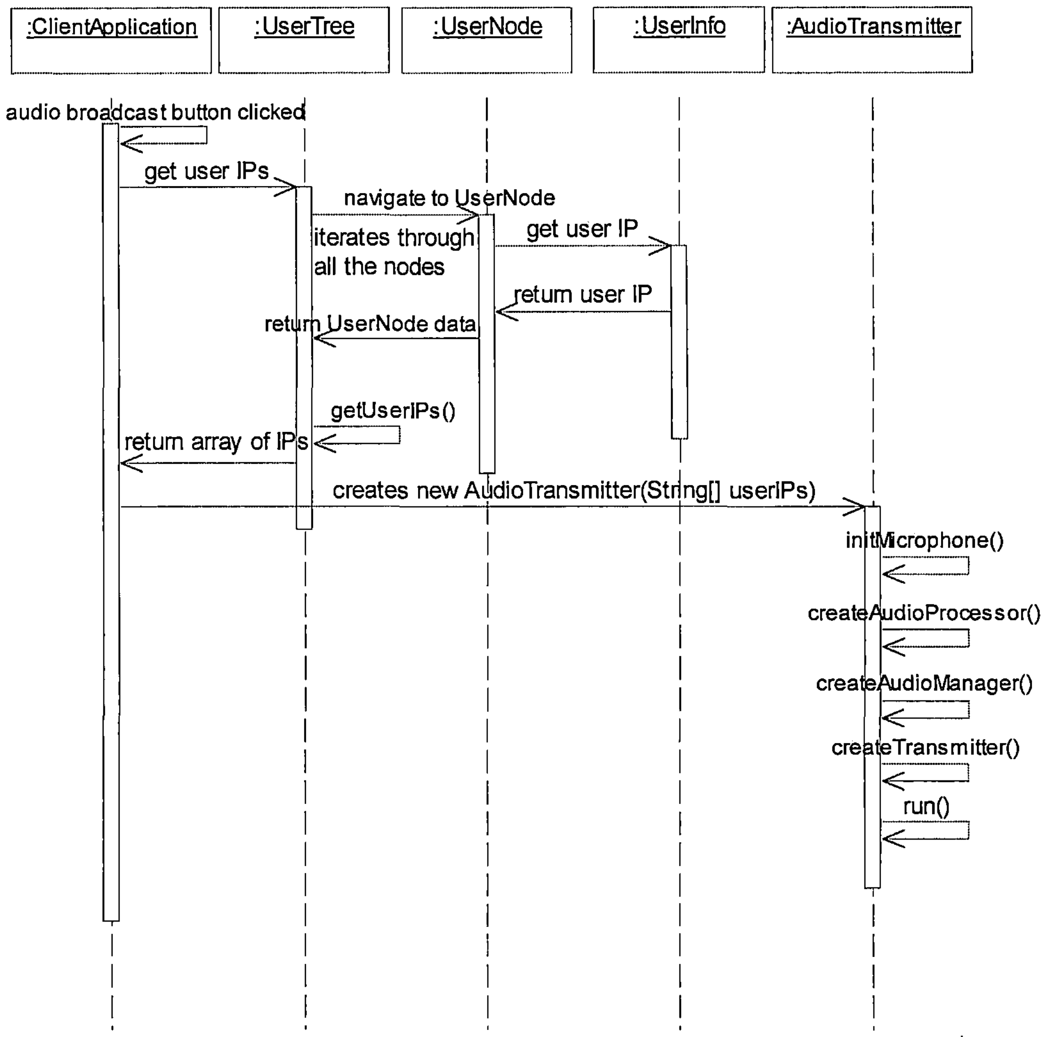


Figure 2.43 Sending Audio Broadcast

Receiving Audio Broadcast

JscmtMessageConsumer receives audio broadcast message. It checks with Client if AudioReceiver exists. This is done to assure that at any given time one instance for AudioReceiver exists. If AudioReceiver does not exist, JscmtMessageConsumer creates an instance of AudioReceiver.

AudioReceiver creates RTP sessions for audio at specified port and listens to new incoming audio streams at this port. When a new audio stream is received, AudioReceiver creates a player to render the stream. Sequence diagram for receiving audio broadcast is shown in Figure 2.44.

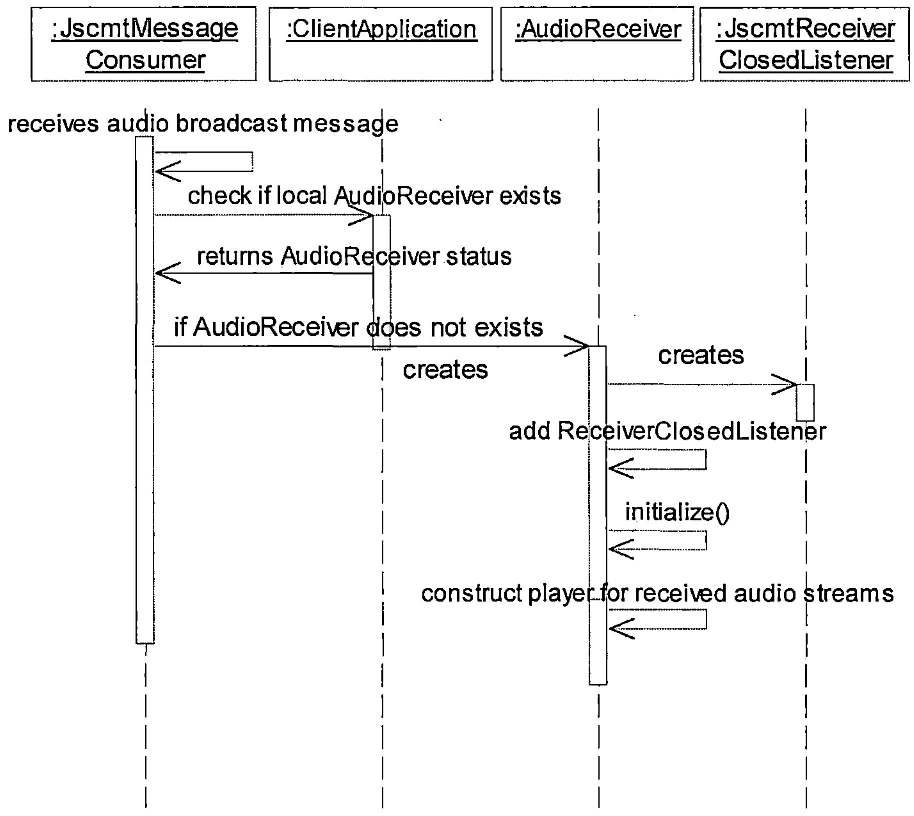

:JscmtMessage Consumer
:ClientApplication
:AudioReceiver
:JscmtReceiver ClosedListener
receives audio broadcast message
check if local AudioReceiver exists
returns AudioReceiver status
if AudioReceiver does not exists
creates
creates
add ReceiverClosedListener
initialize()
construct player for received audio streams

Figure 2.44 Receiving Audio Broadcast

## Inviting User for Private AV Conference

When JSCMT client right clicks on any of the users displayed in online users display area, a pop up menu appears showing all the available options to communicate with the other client. On selecting "Invite for Audio-Video Session" action is processed by UserTree. UserTree locates the UserNode for the user selected and gets the required information from the UserInfo data object of UserNode. ClientApplication uses this user information to invite the selected JSCMT client for private audio-video conference. The sequence diagram to explain this scenario is shown in Figure 2.45.

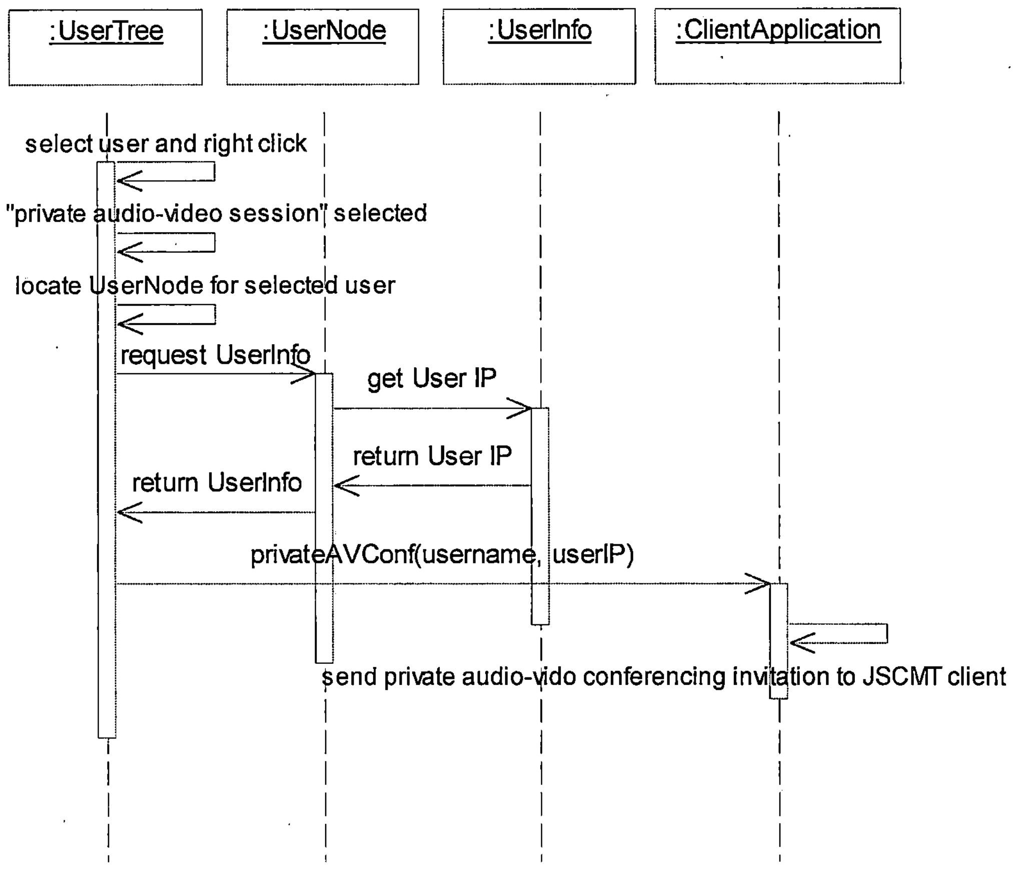

:UserTree
:UserNode
:UserInfo
:ClientApplication
select user and right click
"private audio-video session" selected
locate UserNode for selected user
request UserInfo
get User IP
return User IP
return UserInfo
privateAVConf(username, userIP)
send private audio-vido conferencing invitation to JSCMT client

Figure 2.45 Private AV Invitation

Process Response to Private AV Invitation

On accepting the invitation for private audio-video conference JSCMT client sends "OK" response to the inviting client. Response is received by the JsmctConsumer of the inviting client. If the response is "OK" instance of PrivateAVProcessor is created. PrivateAVProcessor creates AVTransmitter that configures the web cam and microphone and transmits audio and video streams to the other client.

PrivateAVProcessor checks with ClientApplication if AVReceiver exists and creates one in case it does not. AVReceiver constructs player to render every new incoming media stream. The scenario of processing the response to private audio-video conference invitation is explained using sequence diagram shown in figure 2.46.

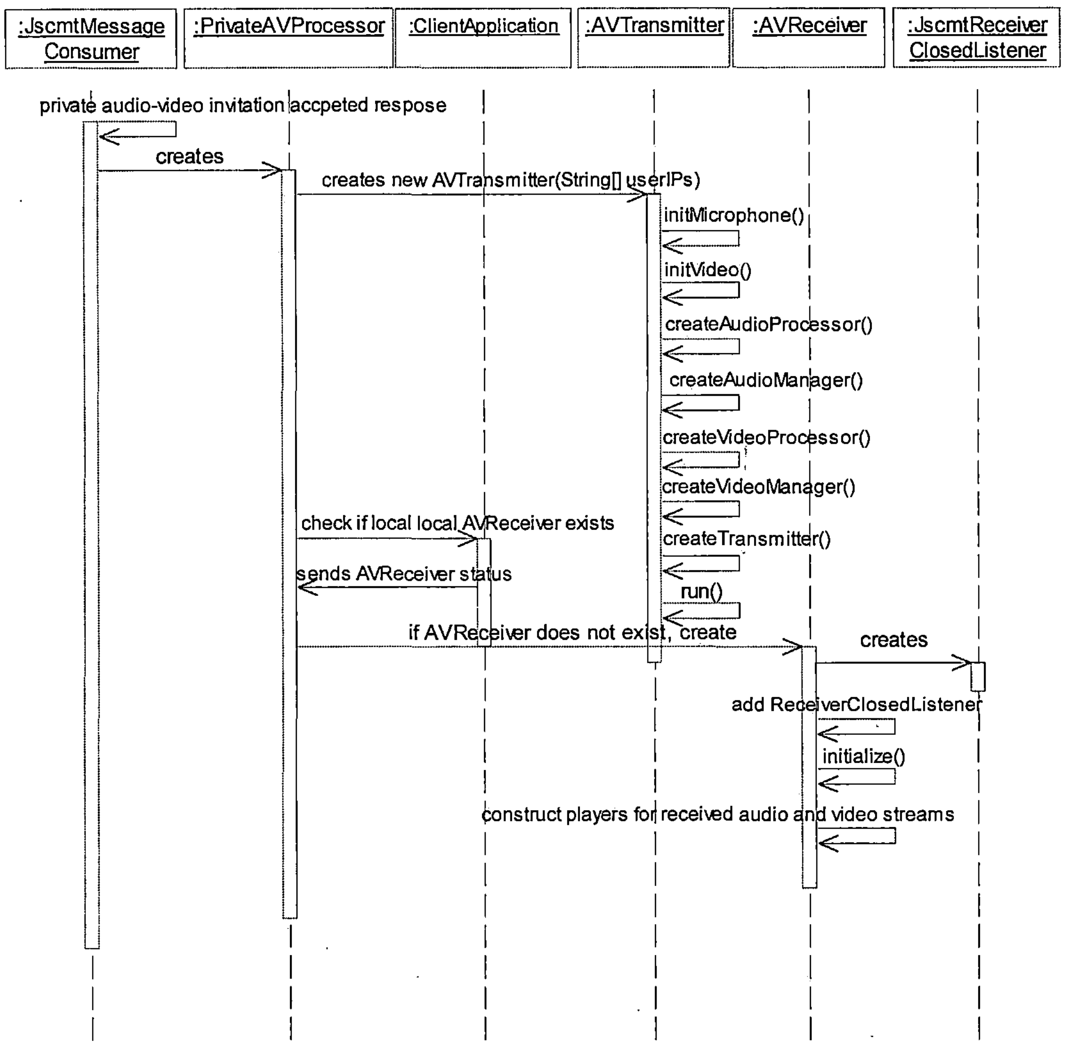

:JscmtMessage Consumer
:PrivateAVProcessor
:ClientApplication
:AVTransmitter
:AVReceiver
:JscmtReceiver ClosedListener
private audio-video invitation accpeted respose
creates
creates new AVTransmitter(String[] userIPs)
initMicrophone()
initVideo()
createAudioProcessor()
createAudioManager()
createVideoProcessor()
createVideoManager()
check if local local AVReceiver exists
createTransmitter()
sends AVReceiver status
run()
if AVReceiver does not exist, create
creates
add ReceiverClosedListener
initialize()
construct players for received audio and video streams

Figure 2.46 Process Private AV Invitation

## <u>Inviting User for Private Audio Conference</u>

When JSCMT client right clicks on any of the users displayed in online users display area, a pop up menu appears showing all the available options to communicate with the other client. On selecting "Invite for Audio

Session" action is processed by UserTree. UserTree locates the UserNode for the user selected and gets the required information from the UserInfo data object of UserNode. ClientApplication uses this user information to invite the selected JSCMT client for private audio conference. The sequence diagram to explain this scenario is shown in Figure 2.47.

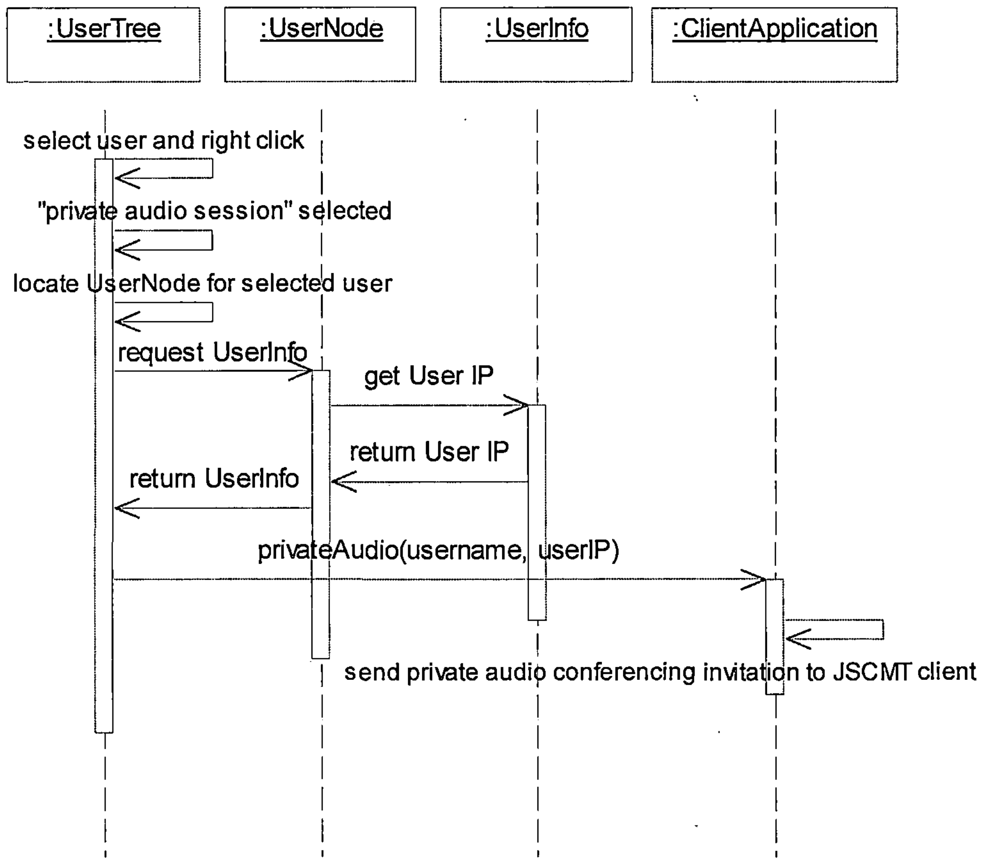

:UserTree
:UserNode
:UserInfo
:ClientApplication
select user and right click
"private audio session" selected
locate UserNode for selected user
request UserInfo
get User IP
return User IP
return UserInfo
privateAudio(username, userIP)
send private audio conferencing invitation to JSCMT client

Figure 2.47 Private Audio Invitation

## Process Response to Private Audio Invitation

On accepting the invitation for private audio conference JSCMT client sends "OK" response to the inviting client. Response is received by the JsmctConsumer of the inviting client. If the response is "OK" instance of PrivateAudioProcessor is created. PrivateAudioProcessor creates AudioTransmitter that configures microphone and transmits audio streams to the other client. PrivateAudioProcessor checks with ClientApplication if AudioReceiver exists and creates one in case it does not. AudioReceiver constructs player to render every new incoming media stream. The scenario of processing the response to private audio conference invitation is explained using sequence diagram shown in figure 2.48.

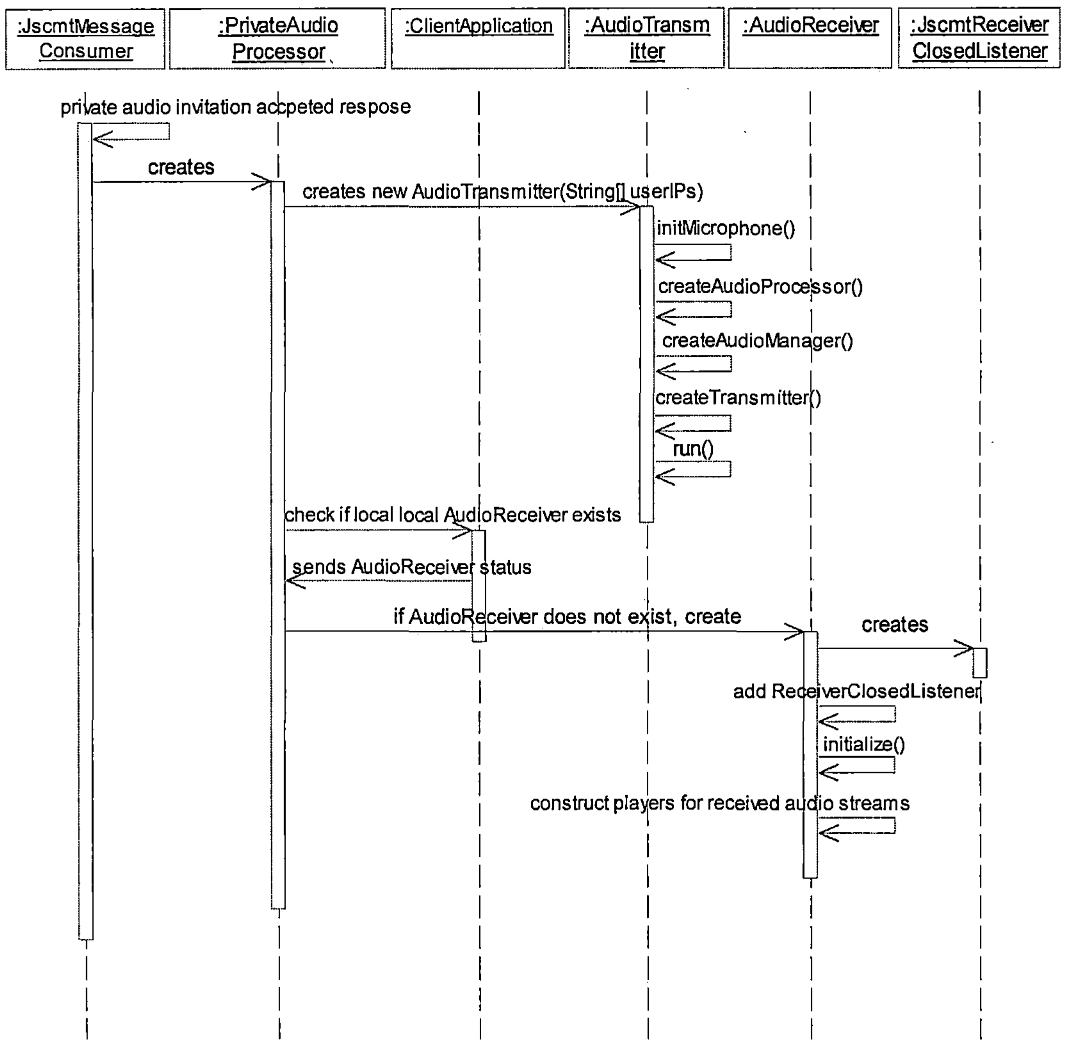

:JscmtMessage Consumer
:PrivateAudio Processor
:ClientApplication
:AudioTransmitter
:AudioReceiver
:JscmtReceiver ClosedListener
private audio invitation accpeted respose
creates
creates new AudioTransmitter(String[] userIPs)
initMicrophone()
createAudioProcessor()
createAudioManager()
createTransmitter()
run()
check if local local AudioReceiver exists
sends AudioReceiver status
if AudioReceiver does not exist, create
creates
add ReceiverClosedListener
initialize()
construct players for received audio streams

Figure 2.48 Process Private Audio Invitation

<u>Sending View Web Cam Request</u>

When JSCMT client right clicks on any of the users displayed in online users display area, a pop up menu appears showing all the available options to communicate with the other client. On selecting "View Web Cam", action is processed by UserTree. UserTree locates the UserNode for the user selected and gets the required information from the UserInfo data object of UserNode. ClientApplication uses this user information to request to view other client's web cam. The sequence diagram to explain this scenario is shown in Figure 2.49.

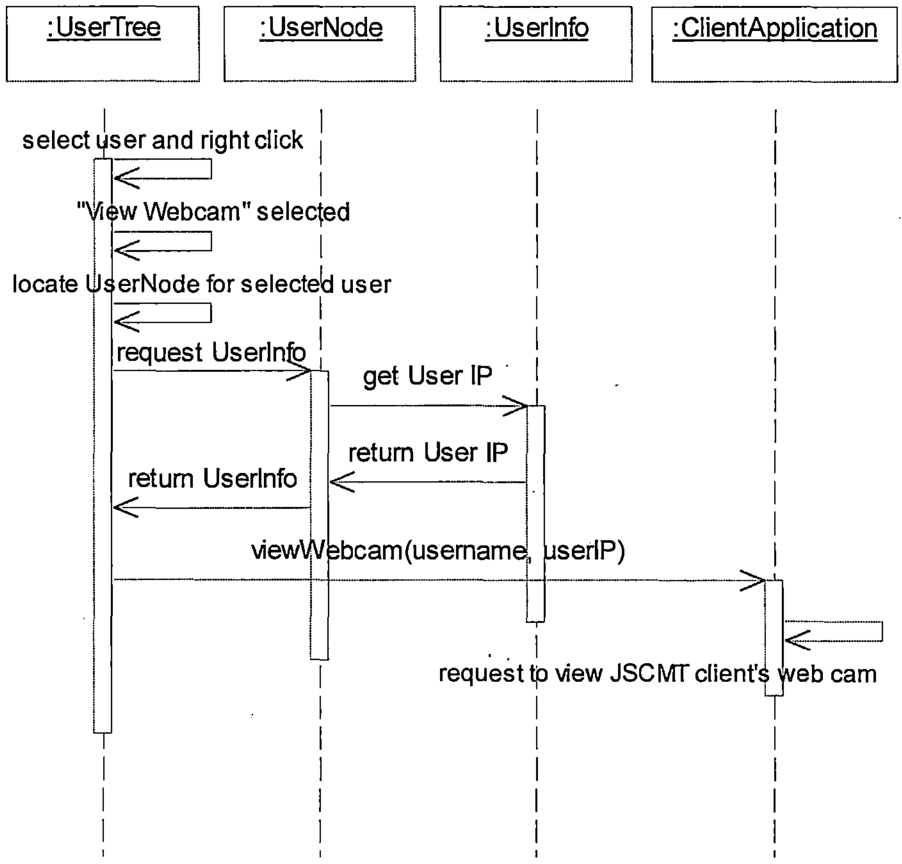

:UserTree
:UserNode
:UserInfo
:ClientApplication
select user and right click
"View Webcam" selected
locate UserNode for selected user
request UserInfo
get User IP
return User IP
return UserInfo
viewWebcam(username, userIP)
request to view JSCMT client's web cam

Figure 2.49 Sending Request to View Web Cam

### Viewing User's Web Cam

On accepting JSCMT client's request to view web cam client sends "OK" response to the client requesting the permission. JscmtMessageConsumer of the requesting client receives the "OK" response. JscmtMessageConsumer checks with ClientApplication if AVReceiver has been created. If AVReceiver does not exist JscmtMessageConsumer creates AVReceiver to receive the media streams from the requested client. The sequence diagram that explains the scenario of viewing user's web cam is shown in the Figure 2.50.

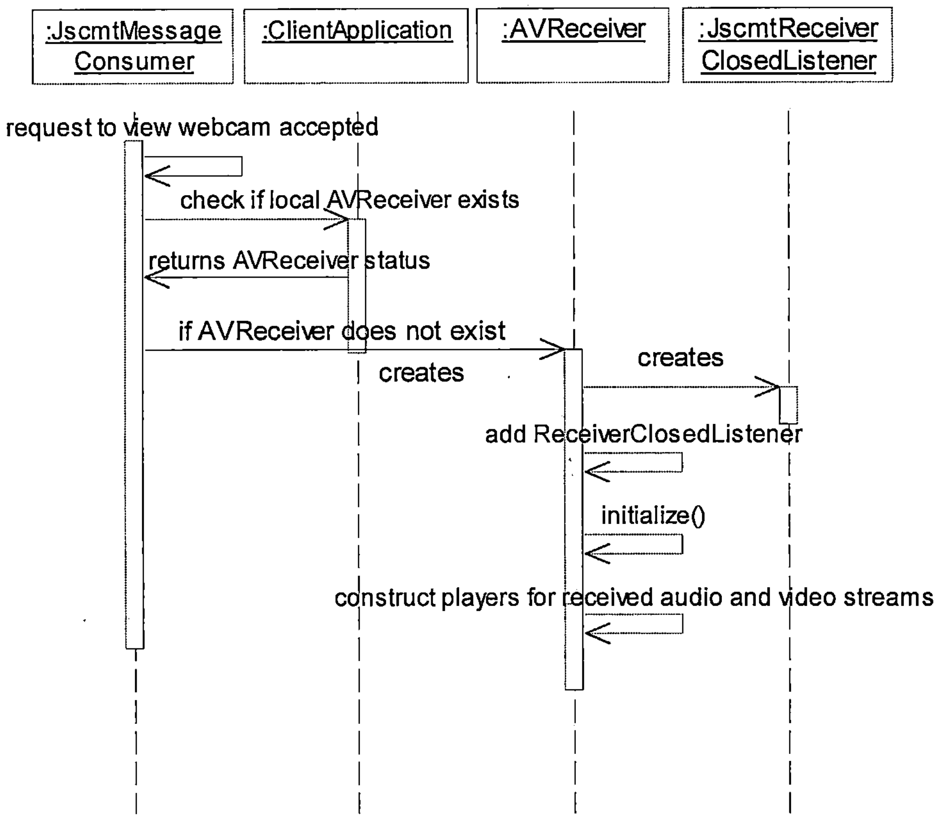

:JscmtMessage Consumer
:ClientApplication
:AVReceiver
:JscmtReceiver ClosedListener
request to view webcam accepted
check if local AVReceiver exists
returns AVReceiver status
if AVReceiver does not exist
creates
creates
add ReceiverClosedListener
initialize()
construct players for received audio and video streams

Figure 2.50 Viewing User's Web Cam

## Inviting User to View Web Cam

When JSCMT client right clicks on any of the users displayed in online users display area, a pop up menu appears showing all the available options to communicate with the other clients. On selecting "Invite to View My Web Cam", action is processed by UserTree. UserTree locates the UserNode for the user selected and gets the required information from the UserInfo data object of UserNode. ClientApplication uses this user information to invite other client to view one's web cam. The sequence diagram to explain this scenario is shown in Figure 2.51.

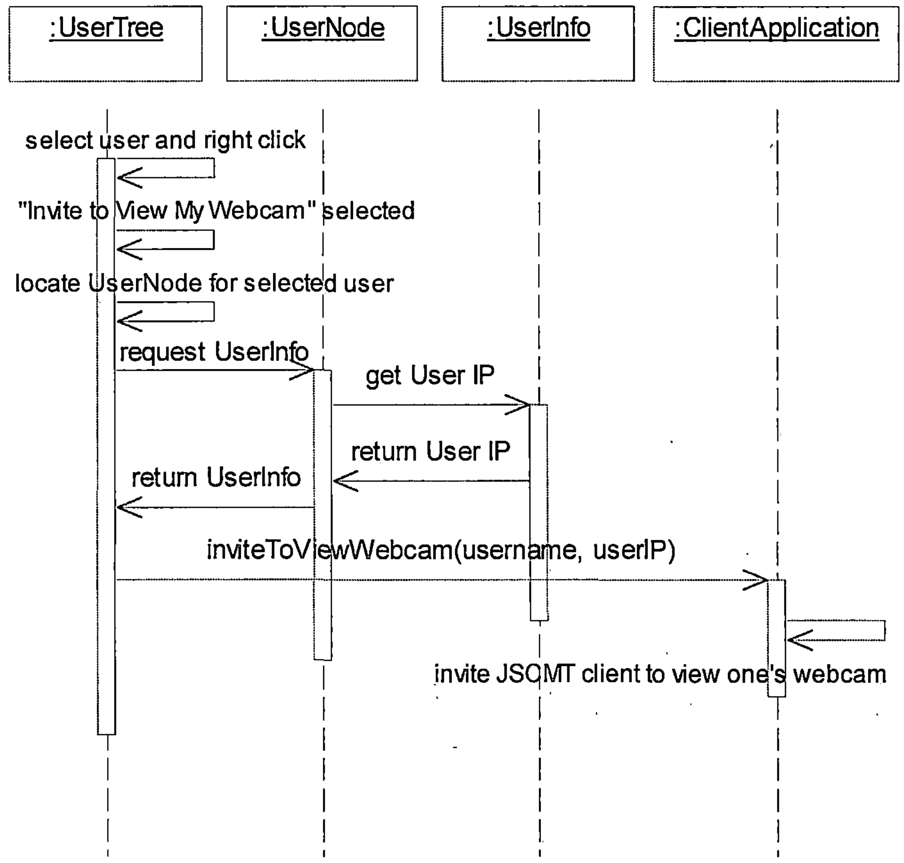

:UserTree
:UserNode
:UserInfo
:ClientApplication
select user and right click
"Invite to View My Webcam" selected
locate UserNode for selected user
request UserInfo
get User IP
return User IP
return UserInfo
inviteToViewWebcam(username, userIP)
invite JSCMT client to view one's webcam

Figure 2.51 Inviting User to View Web Cam

<u>Process Response to View Web Cam Invitation</u>

On accepting the invitation to view inviting client's web cam JSCMT client sends "OK" response to the inviting client. Response is received by the JsmctConsumer of the inviting client. If the response is "OK" instance of WebcamRequestHandler is created. WebcamRequestHandler creates AVTransmitter that configures the web cam and microphone and transmits audio and video streams to the other client. WebcamRequestHandler checks with ClientApplication, if AVReceiver exists and creates one in case it does not. AVReceiver constructs player to render every new incoming media stream. The scenario of processing the response to view web cam invitation is explained using sequence diagram shown in figure 2.52.

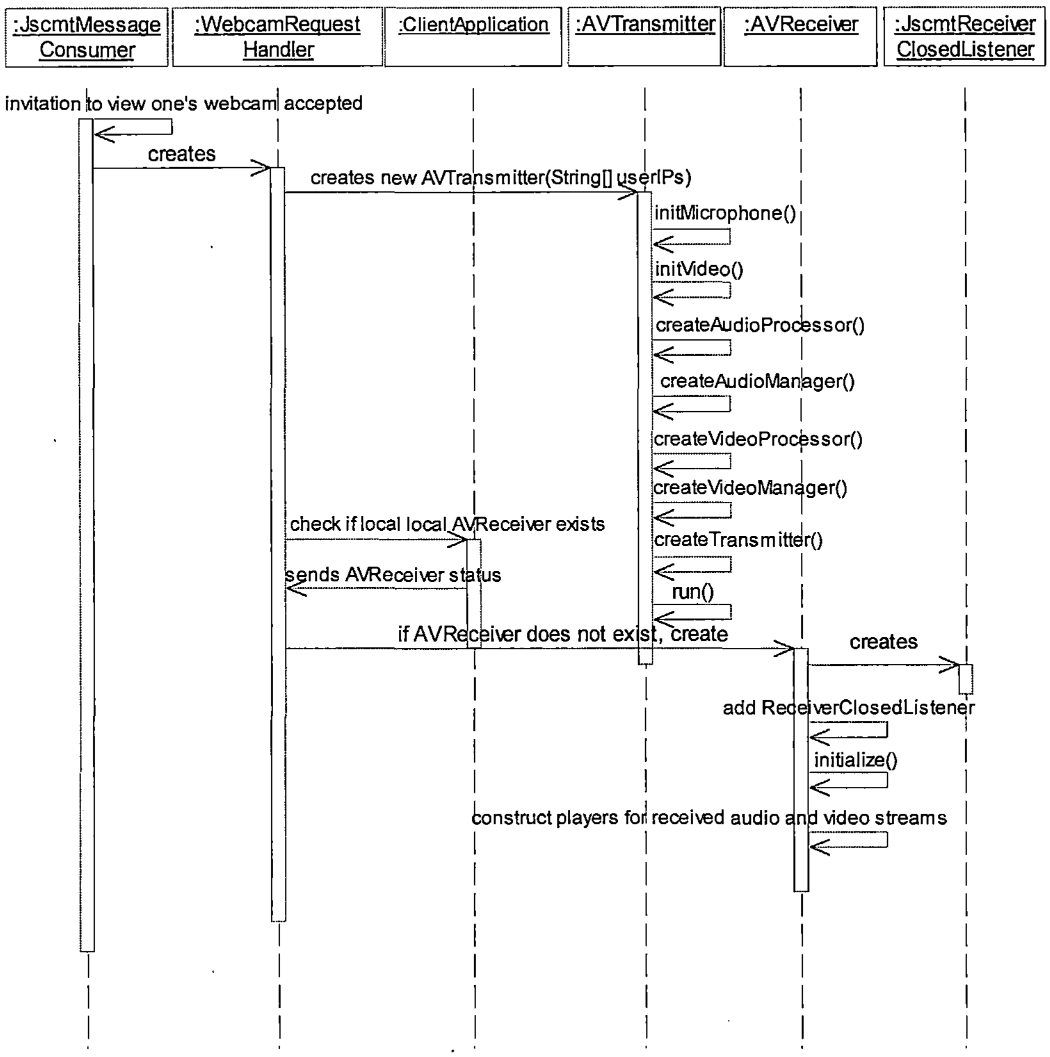


Figure 2.52 Process View Web Cam Invitation

## Inviting User for Private Text Messaging

On selecting "Send Private Message" option from the available communication options, action is processed by UserTree. UserTree locates the UserNode for the user selected and gets the required information from the

UserInfo data object of UserNode. ClientApplication uses this user information to invite other client for private text messaging. The sequence diagram to explain this scenario is shown in Figure 2.53.

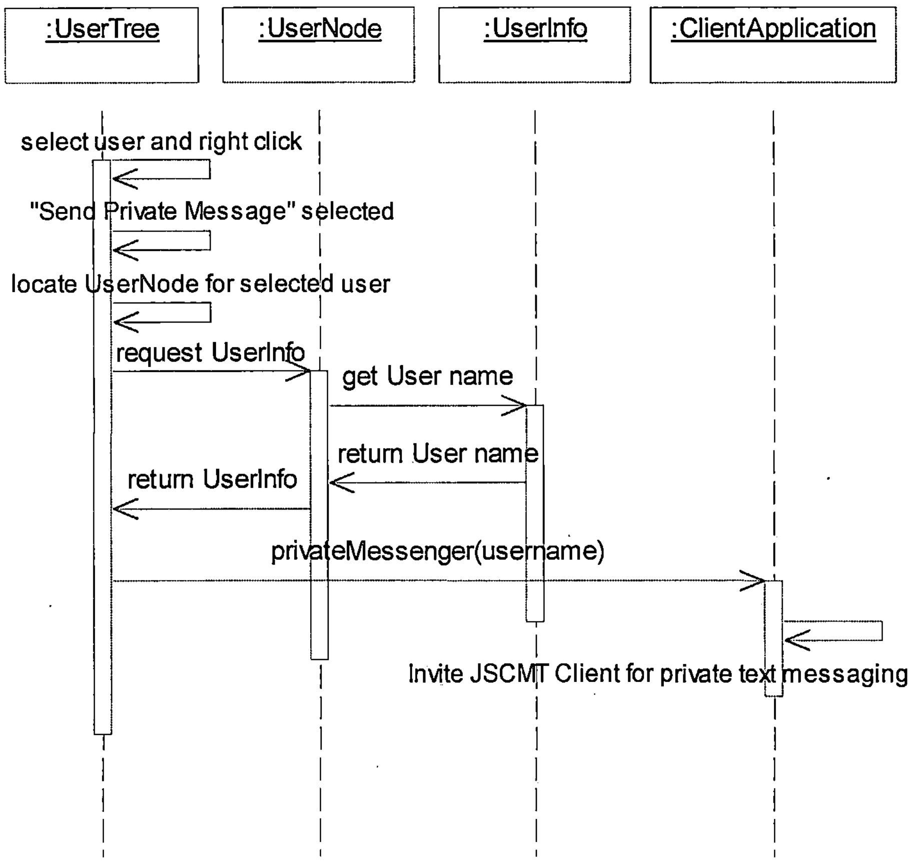

:UserTree
:UserNode
:UserInfo
:ClientApplication
select user and right click
"Send Private Message" selected
locate UserNode for selected user
request UserInfo
get User name
return User name
return UserInfo
privateMessenger(username)
Invite JSCMT Client for private text messaging

Figure 2.53 Inviting for Private Text Messaging

<u>Process Private Text Messaging Invitation</u>

On accepting JSCMT client's invitation for private text messaging client sends "OK" response to the inviting client. If the response is "OK" it is directed to

ClientApplication. It creates private Channel if it does not exists, which is used to exchange private messages between two clients. To manage private text messaging PrivateMessageHandler is created. PrivateMessageHandler associates the private Channel with PrivateMessageConsumer. Private message window is then displayed which acts as an interface for private text messaging. The scenario to process private text messaging invitation is explained by the sequence diagram shown in Figure 2.54.

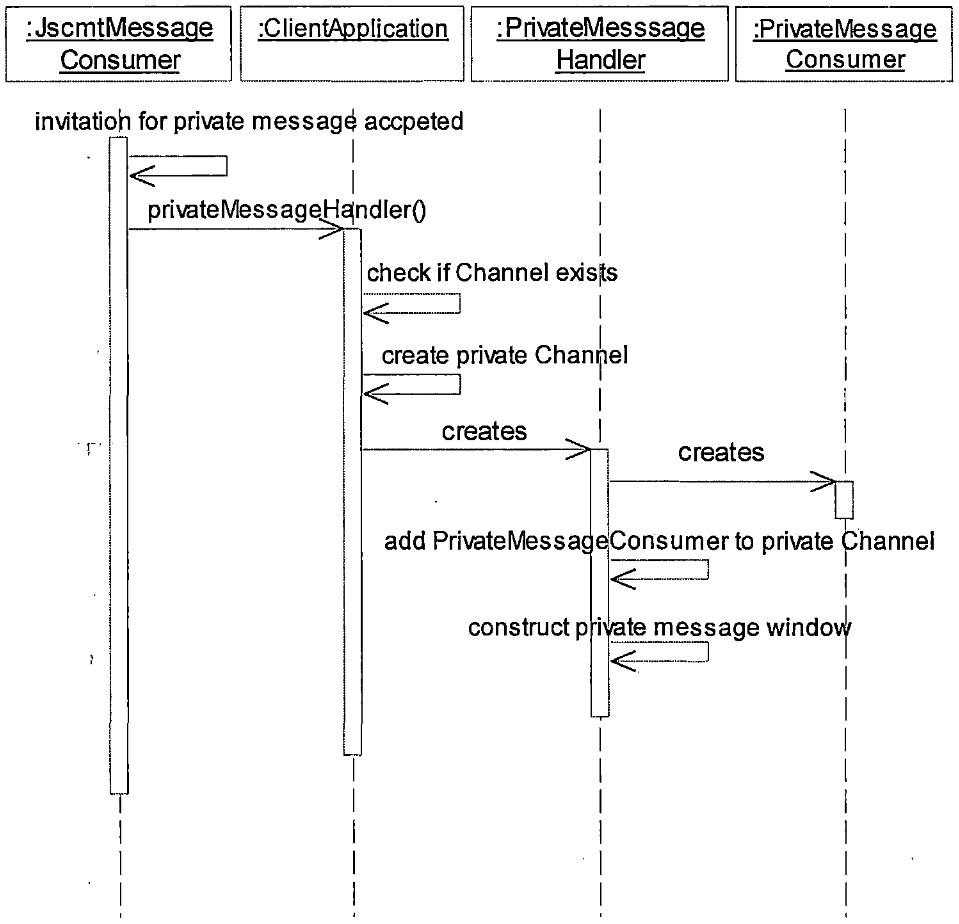


Figure 2.54 Process Private Text Messaging

## Summary

In this chapter JSCMT architecture, JSCMT package structure, JSCMT components were explained. Data flow between various components to accomplish the functionality defined by the application was explained using sequence diagrams.

# CHAPTER THREE

# USER MANUAL

## Overview

Java Synchronized Collaborative Multimedia Toolkit (JSCMT) is a Java application that brings together users from different geographical location. Users collaborate through various communication tools integrated by the JSCMT application.

Using JSCMT users can communicate by exchanging text messages with each other. JSCMT clients can collaborate by exchanging real-time voice messages with each other. JSCMT's audio-video support enables client to broadcast audio-video or participate in private audio-video conferencing. JSCMT uses microphone to capture real-time audio and web cam to capture real-time video. JSCMT also supports private text messaging between two clients. At any given time JSCMT client can participate in only one audio-video or audio session because at present creation of only one audio-video or audio transmitter is supported by JSCMT.

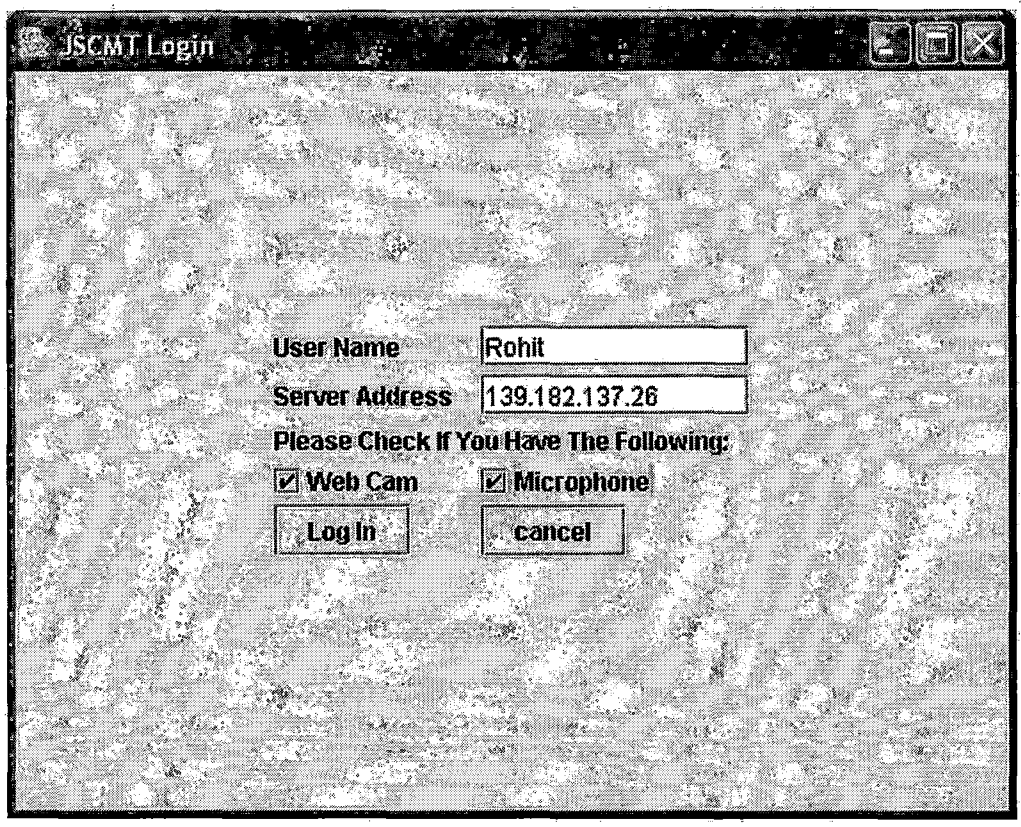
The user manual guides the user in using different functionalities of JSCMT. It elaborates on available features of the application and explains step by step procedures for using them.

## Logging In

Upon starting the application, you will be prompted with a log in screen which is shown in Figure 3.1. Steps involved in logging in are:

- Type username in the username text field.
- Type password in the password text field.
- Type IP address of the server in the server text field.
- Check microphone check box if microphone is available.
- Check web cam check box if web cam is available.
- Click on the "Connect" button.

Once you have successfully logged in, a graphical user interface to all the services provided by JSCMT is created and is shown in Figure 3.2.



User Name: Rohit

Server Address: 139.182.137.26

Please Check If You Have The Following:

[✓] Web Cam [✓] Microphone

Figure 3.1 JSCMT Login Window

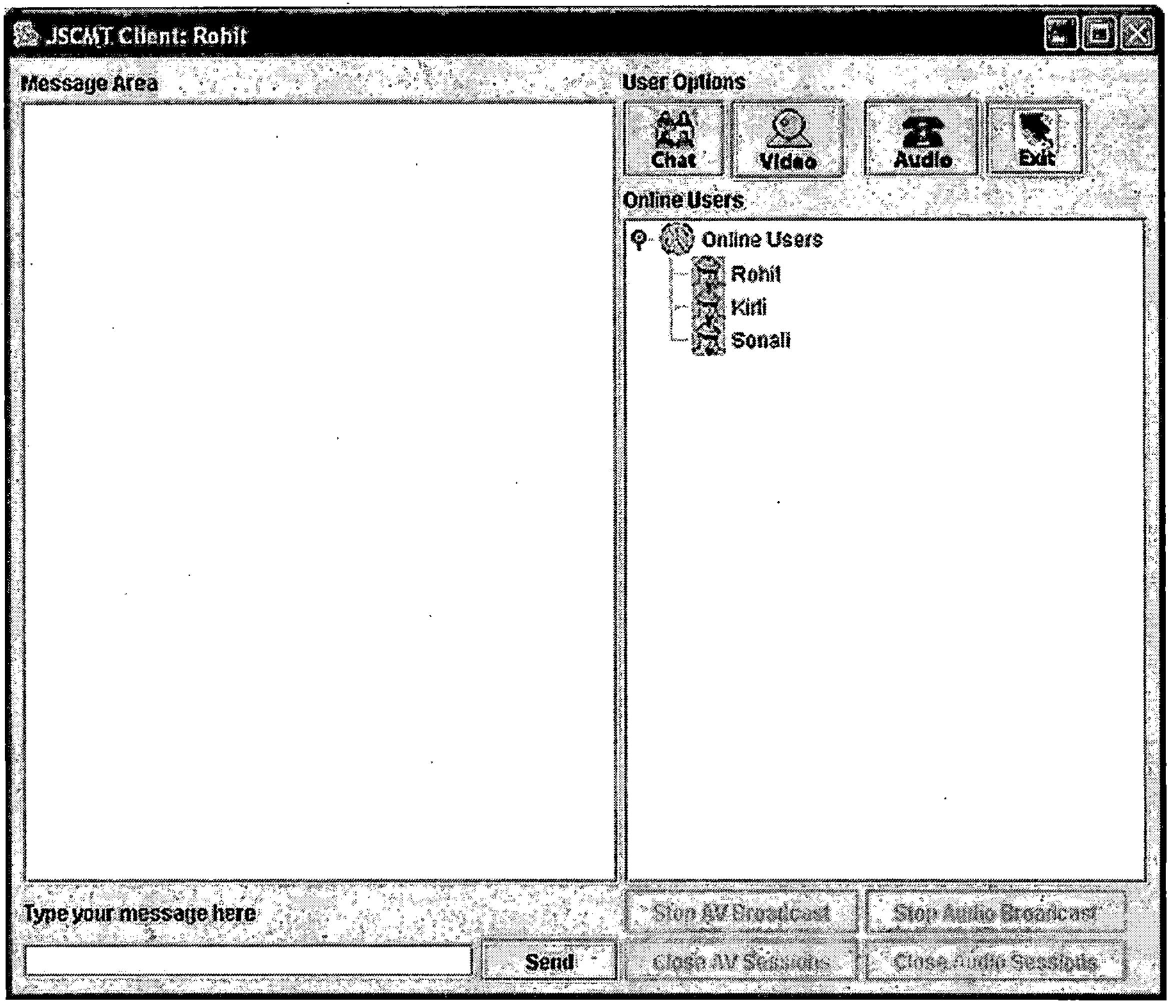


Figure 3.2 JSCMT Client Graphical User Interface

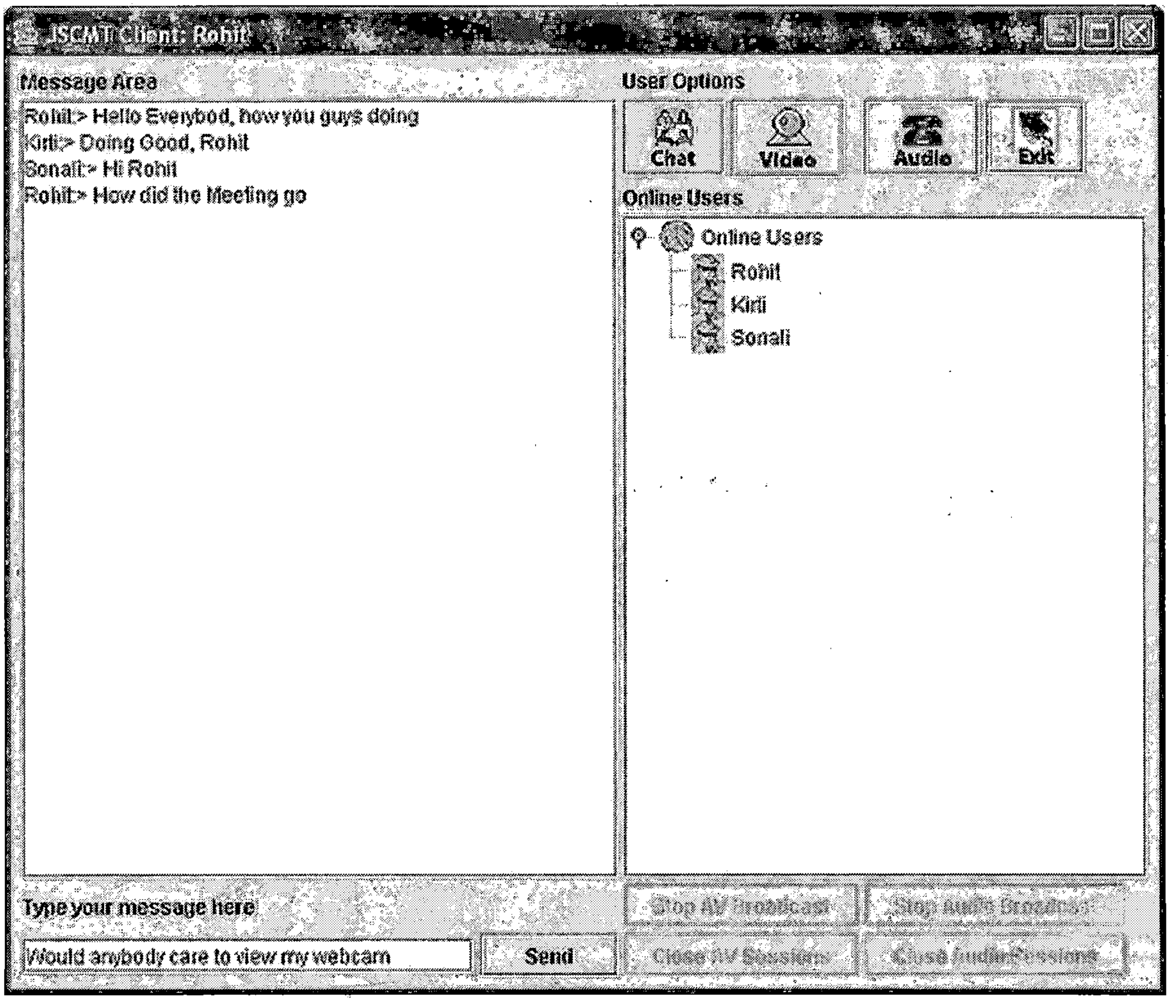
## Broadcast Text Message

To broadcast text message to all the other clients logged into the system the steps involved are:

- Click on the button with label "Chat", which is the part of the user options button group of client application window.

- Type message to be broadcasted in the message field.
- Click on the "Send" button.

JSCMT client broadcasting text message is shown in Figure 3.3.



Rohit:> Hello Everybod, how you guys doing
Kirti:> Doing Good, Rohit
Sonali:> Hi Rohit
Rohit:> How did the Meeting go

Would anybody care to view my webcam

Figure 3.3 Broadcast Text Message

## Broadcast Audio-Video

JSCMT client needs to have web cam and microphone to broadcast audio-video to all the users. The steps involved in broadcasting audio-video are:

- Click on the button with label "Video" and camera icon, which is the part of the user options button group of client application window.
- You are prompted to confirm the audio-video broadcast action.
- Upon confirming the action, audio-video is broadcasted to all the users

JSCMT client broadcasting audio-video is shown in Figure 3.4.

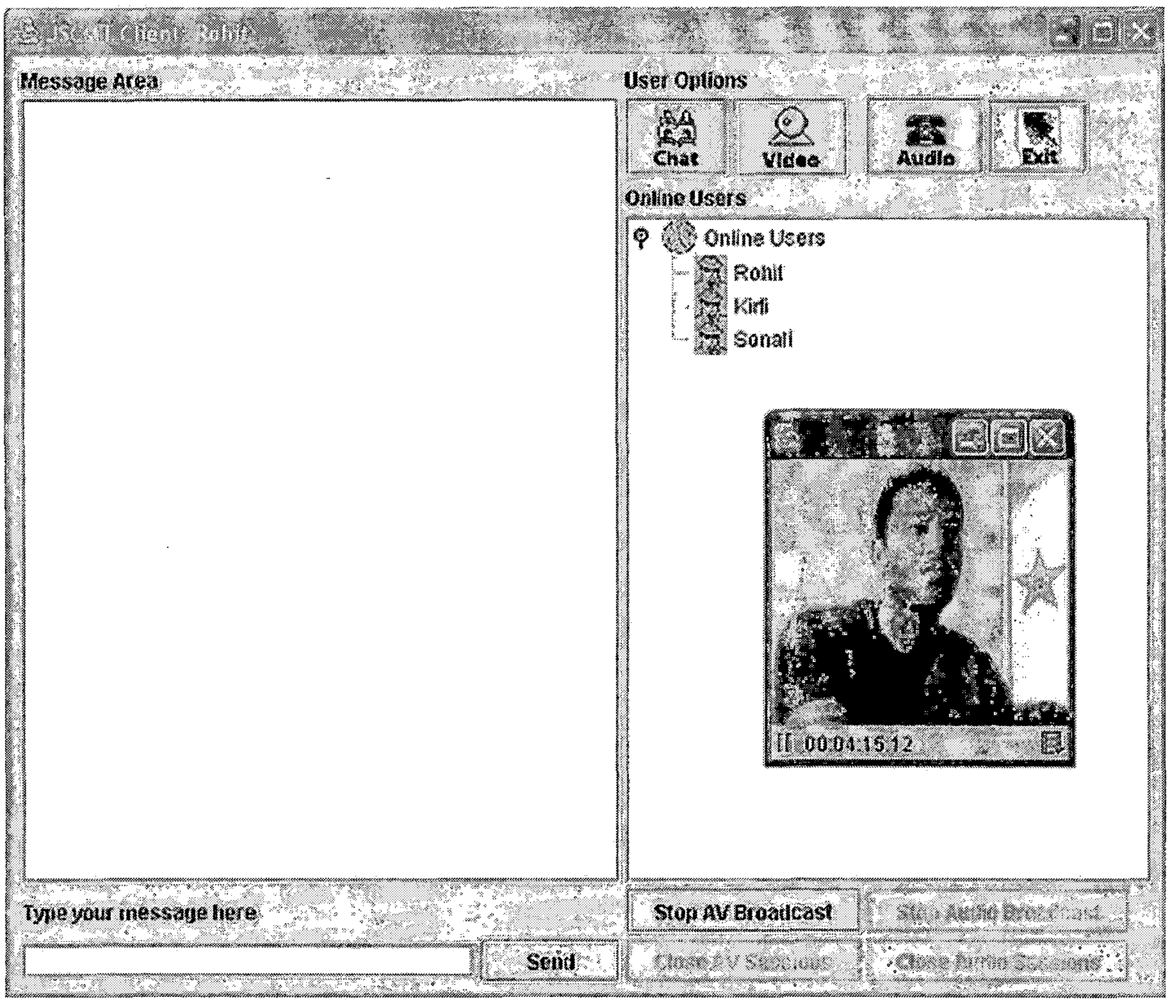
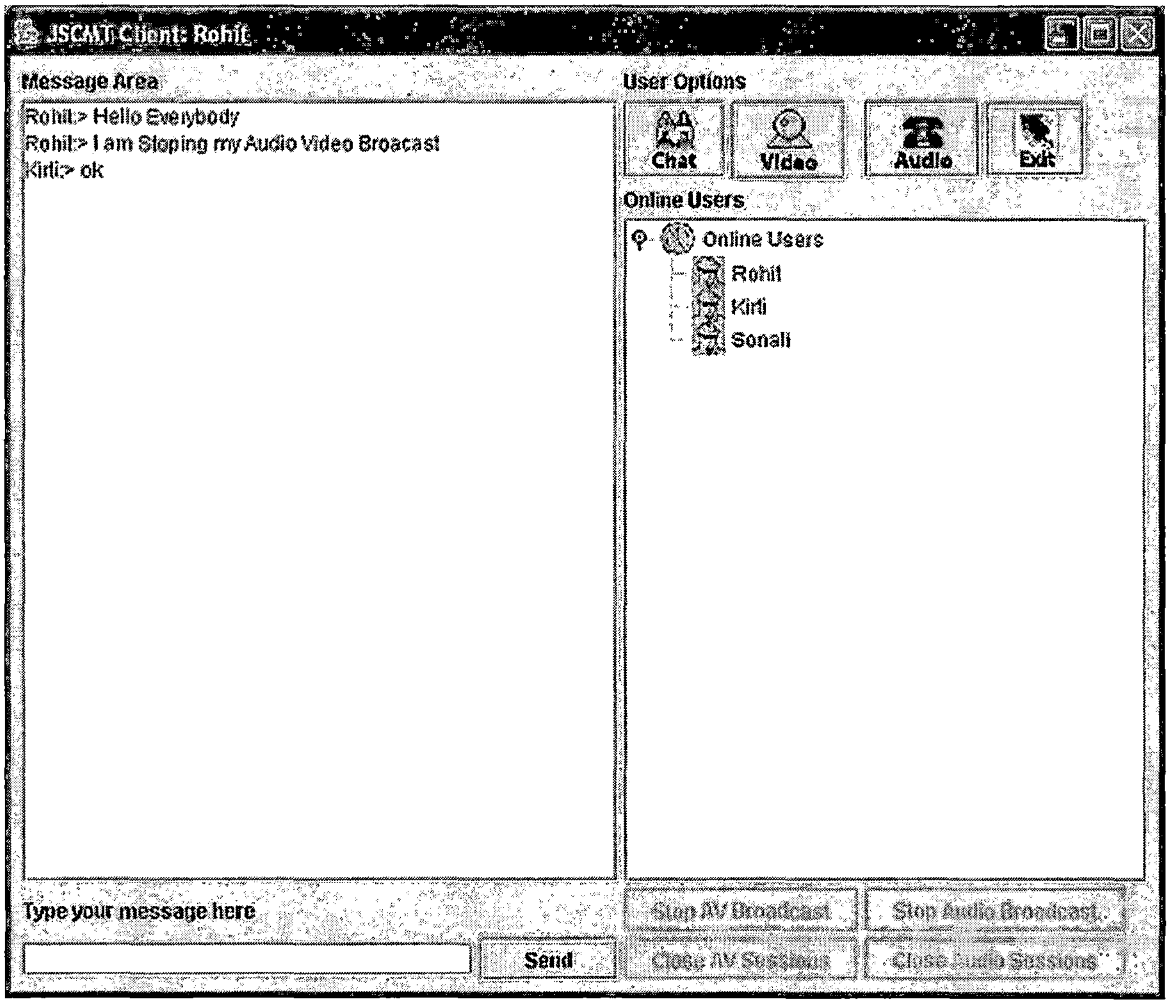


Figure 3.4 Broadcast Audio-Video

Stop Audio-Video Broadcast

Audio-Video Broadcast can be stopped by clicking "Stop AV Broadcast" button. This is shown in Figure 3.5.



Rohit:> Hello Everybody
Rohit:> I am Stoping my Audio Video Broacast
Kirti:> ok

Figure 3.5 Stop Audio-Video Broadcast

## Broadcast Audio

JSCMT client needs to have microphone to broadcast audio to all the users. The Steps involved in broadcasting audio are:

- Click on the button with label "Audio" and phone icon, which is the part of the user options button group of client application window.

- You are prompted to confirm the audio broadcast action by a dialogue box.
- Upon confirming the action, audio is broadcasted to all the users.

JSCMT client broadcasting audio is shown in Figure 3.6.

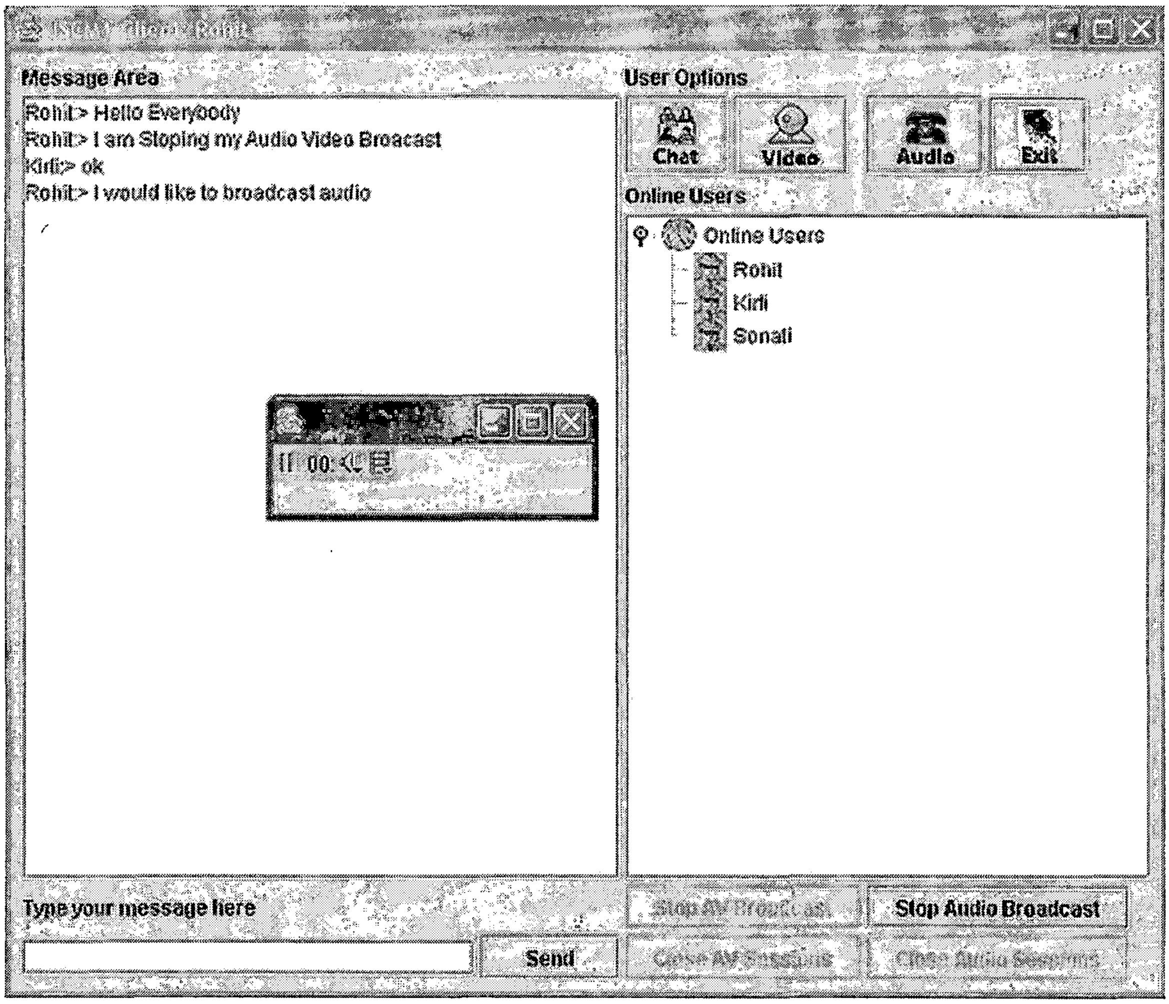

Rohit> Hello Everybody
Rohit> I am Stoping my Audio Video Broacast
Kirti> ok
Rohit> I would like to broadcast audio

Figure 3.6 Broadcast Audio

## Stop Audio Broadcast

Audio Broadcast can be stopped by clicking "Stop Audio Broadcast" button. This is shown in Figure 3.7.

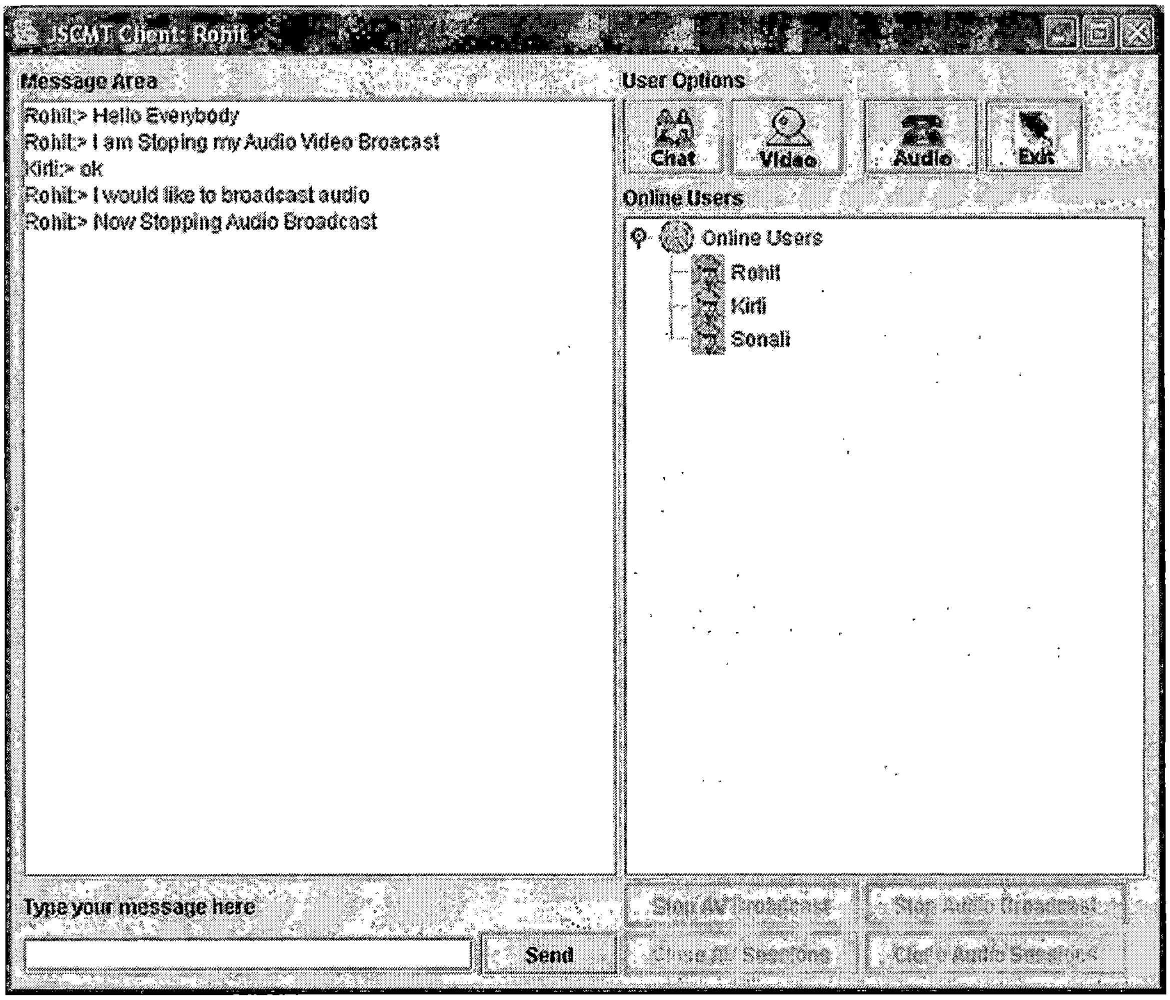

Rohit:> Hello Everybody
Rohit:> I am Stoping my Audio Video Broacast
Kirti:> ok
Rohit:> I would like to broadcast audio
Rohit:> Now Stopping Audio Broadcast

Figure 3.7 Stop Audio Broadcast

## Inviting User for Private Audio-Video Conference

The steps involved in inviting JSCMT client for private audio-video conference are:

- Select the user from the list of online users displayed in the users online display area.
- Right click to pop up the context menu.
- Select "Invite for Audio-Video Session" option.
- You are prompted to confirm the action.
- Upon confirming the action, invitation is sent to the selected client.

JSCMT client inviting another client for private audio-video conferencing is shown in Figure 3.8 and Figure 3.9 shows the client receiving this invitation. Both the clients participating in a private audio-video conferencing can be seen in Figure 3.10.

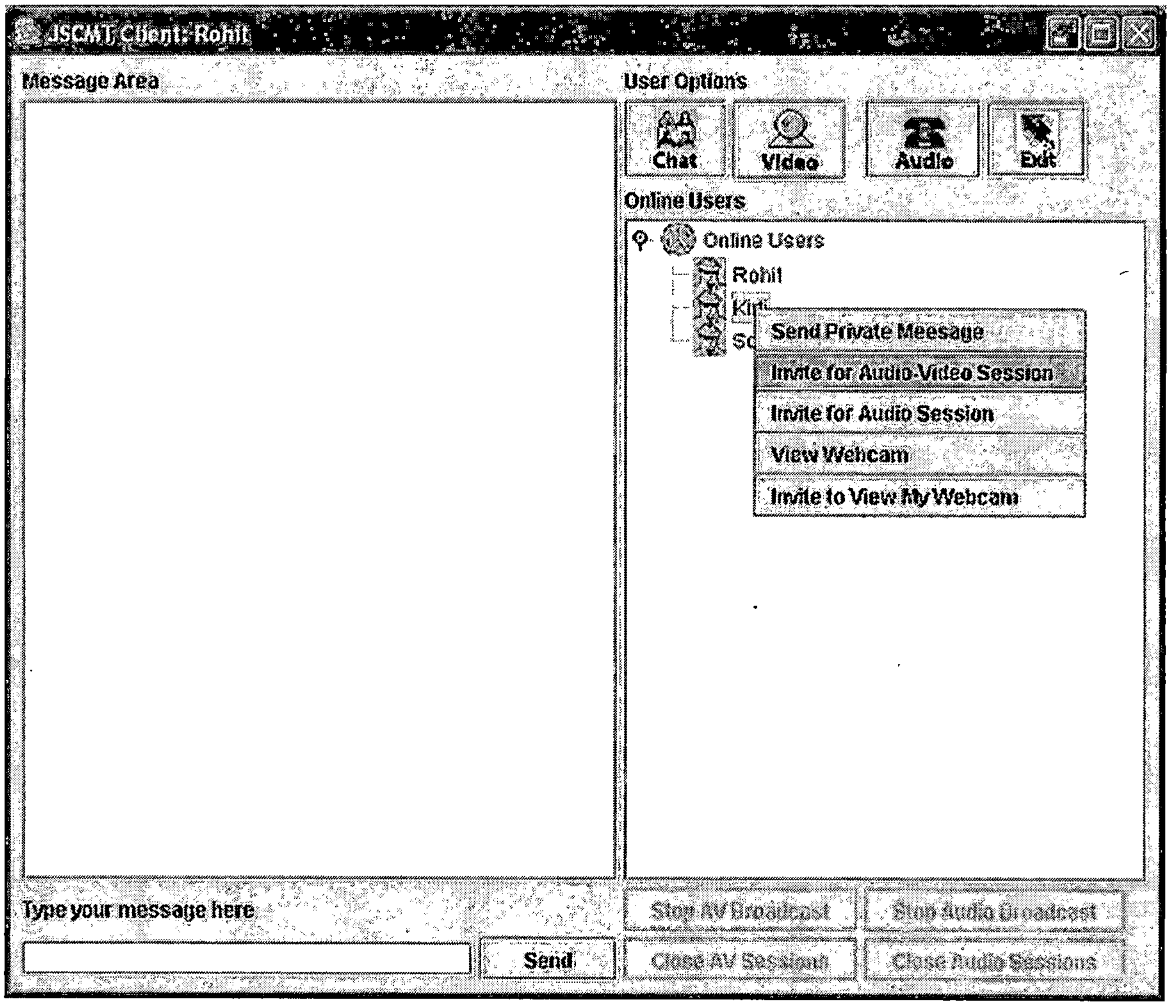
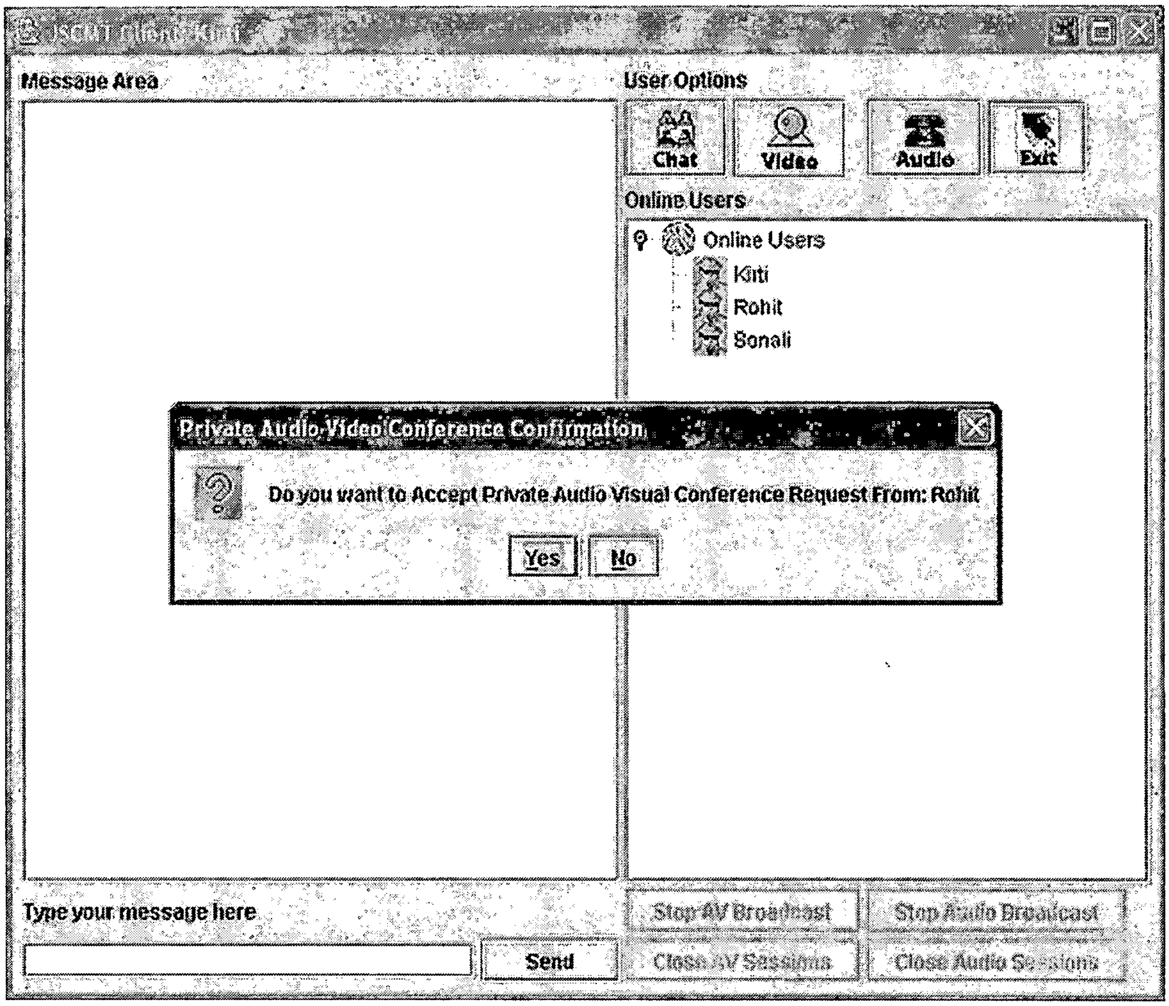


Figure 3.8 Invite for Private AV Conference



Do you want to Accept Private Audio Visual Conference Request From: Rohit

Figure 3.9 Receive Private AV Conference Invitation

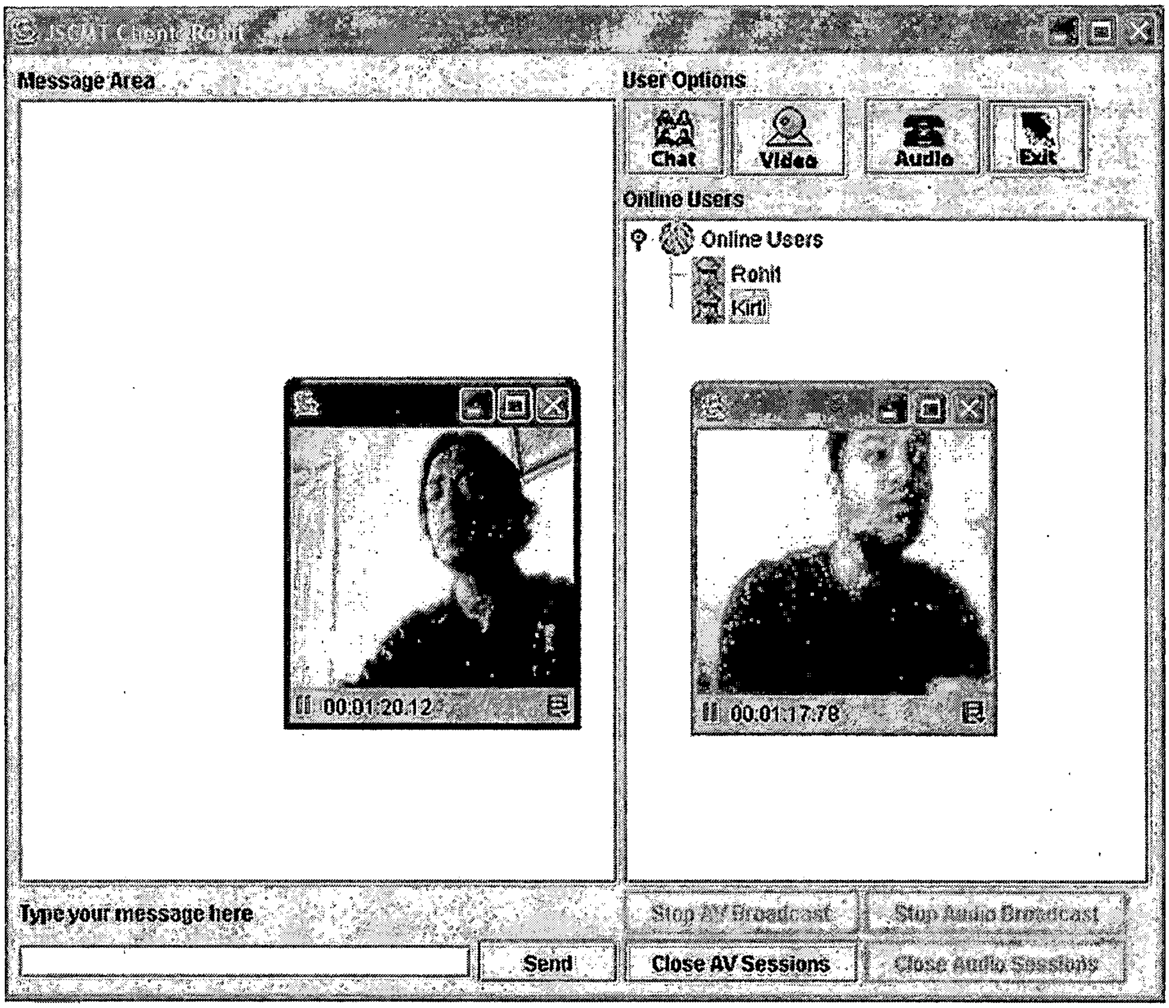


Figure 3.10 Participating in Private AV Conference

Close Audio-Video Session

JSCMT client participating in private audio-video session can close audio-video session by clicking "Close AV Sessions" button. This is shown in Figure 3.11.

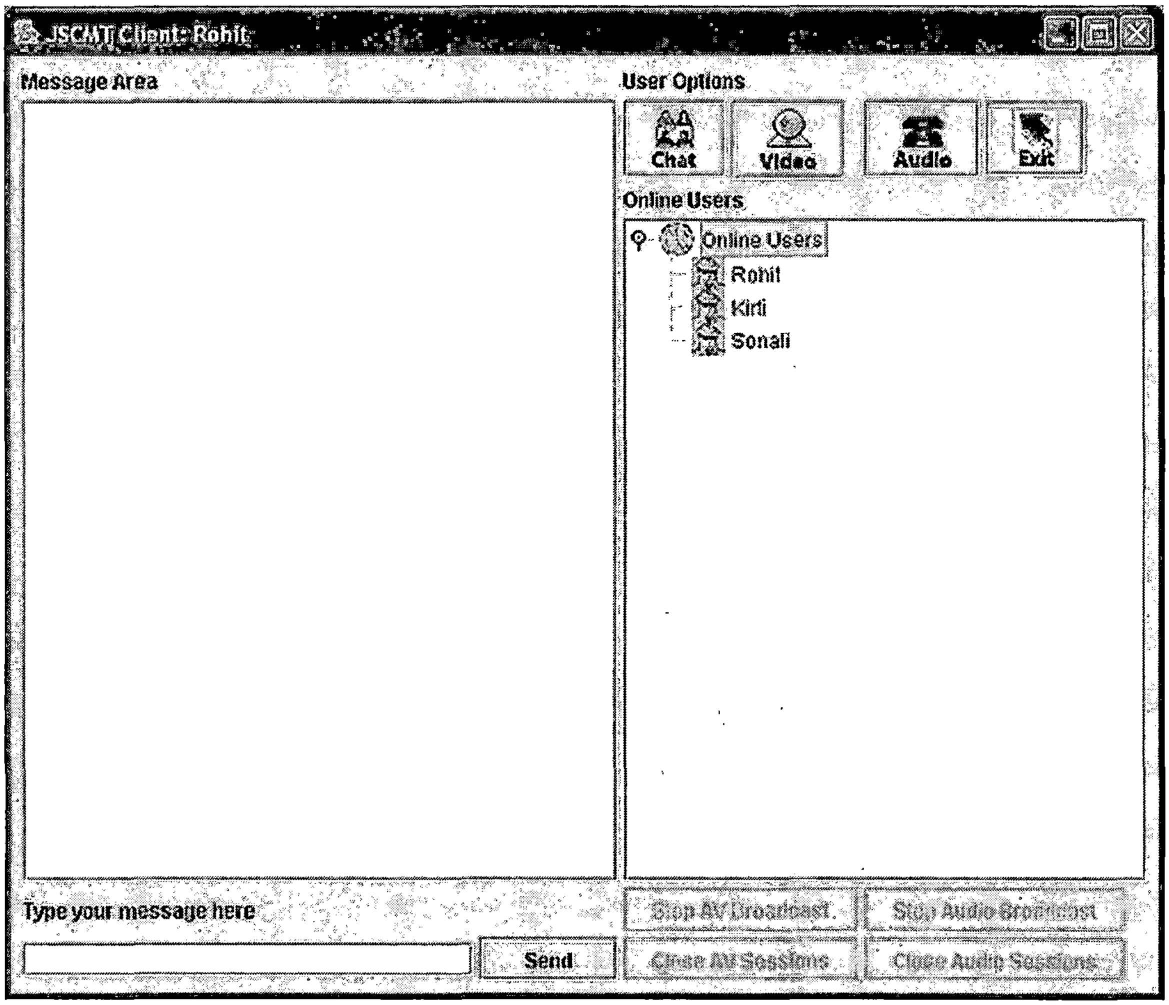


Figure 3.11 Close Audio Video Session

Inviting User for Private Audio Conference

The steps involved in inviting JSCMT client for private audio conference are:

- Select the user from the list of online users displayed in the users online display area.
- Right click to pop up the context menu.
- Select "Invite for Audio Session" option.

- You are prompted to confirm the action.
- Upon confirming the action, invitation is sent to the selected client.

JSCMT client inviting another client for private audio conferencing is shown in Figure 3.12 and Figure 3.13 shows the client receiving this invitation. Both the clients participating in a private audio conferencing can be seen in Figure 3.14.

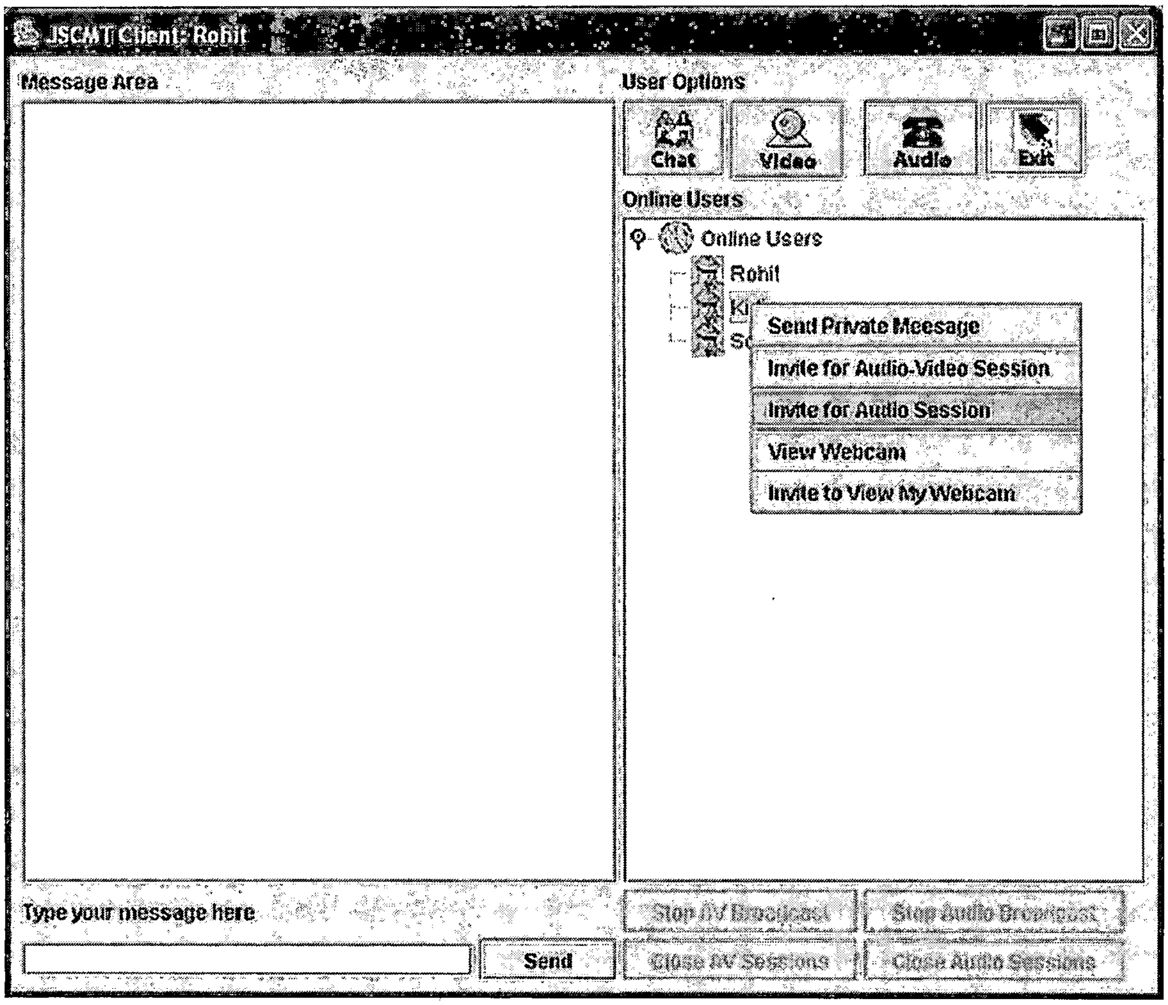
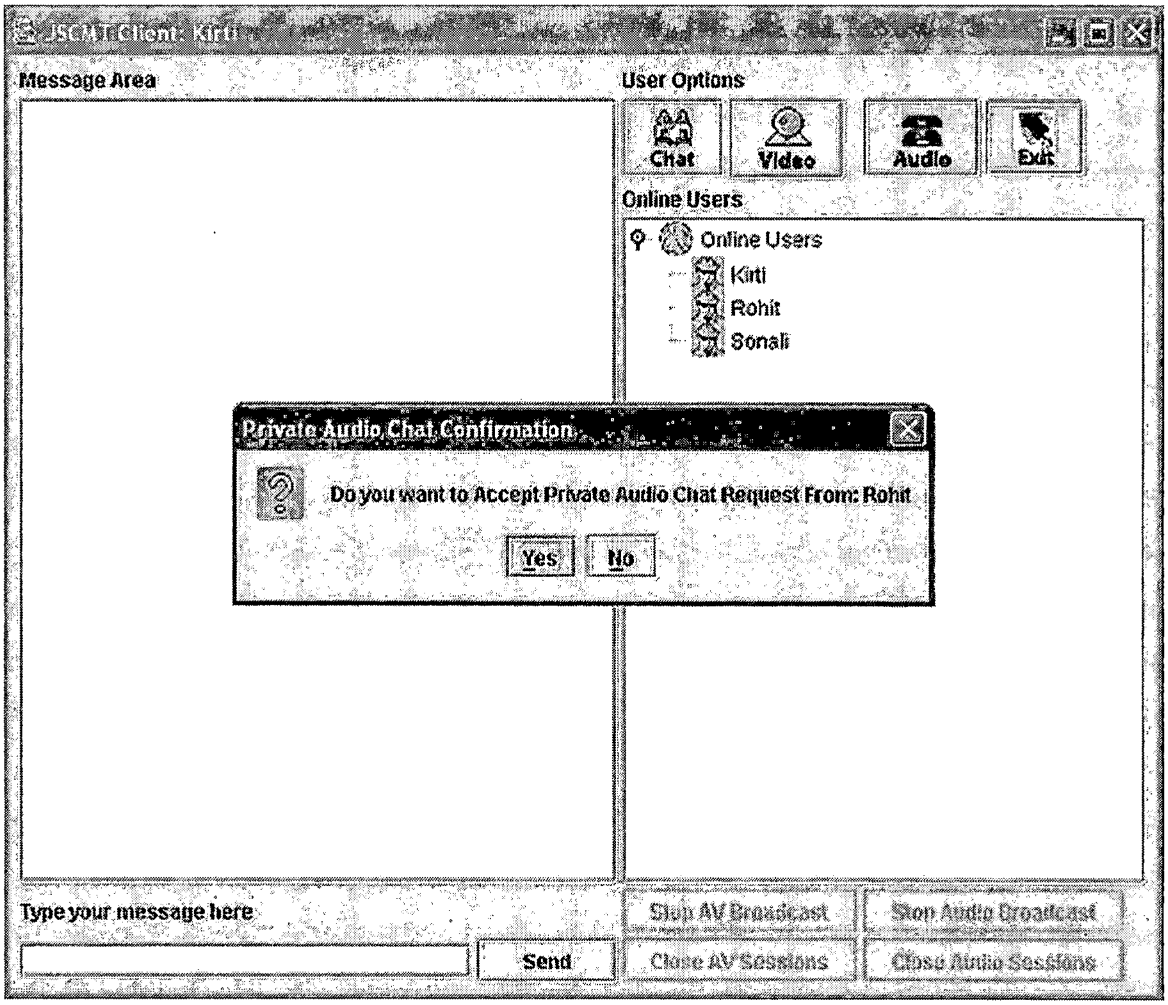


Figure 3.12 Invite for Private Audio Conference



Do you want to Accept Private Audio Chat Request From: Rohit

Figure 3.13 Receive Private Audio Conference Invitation

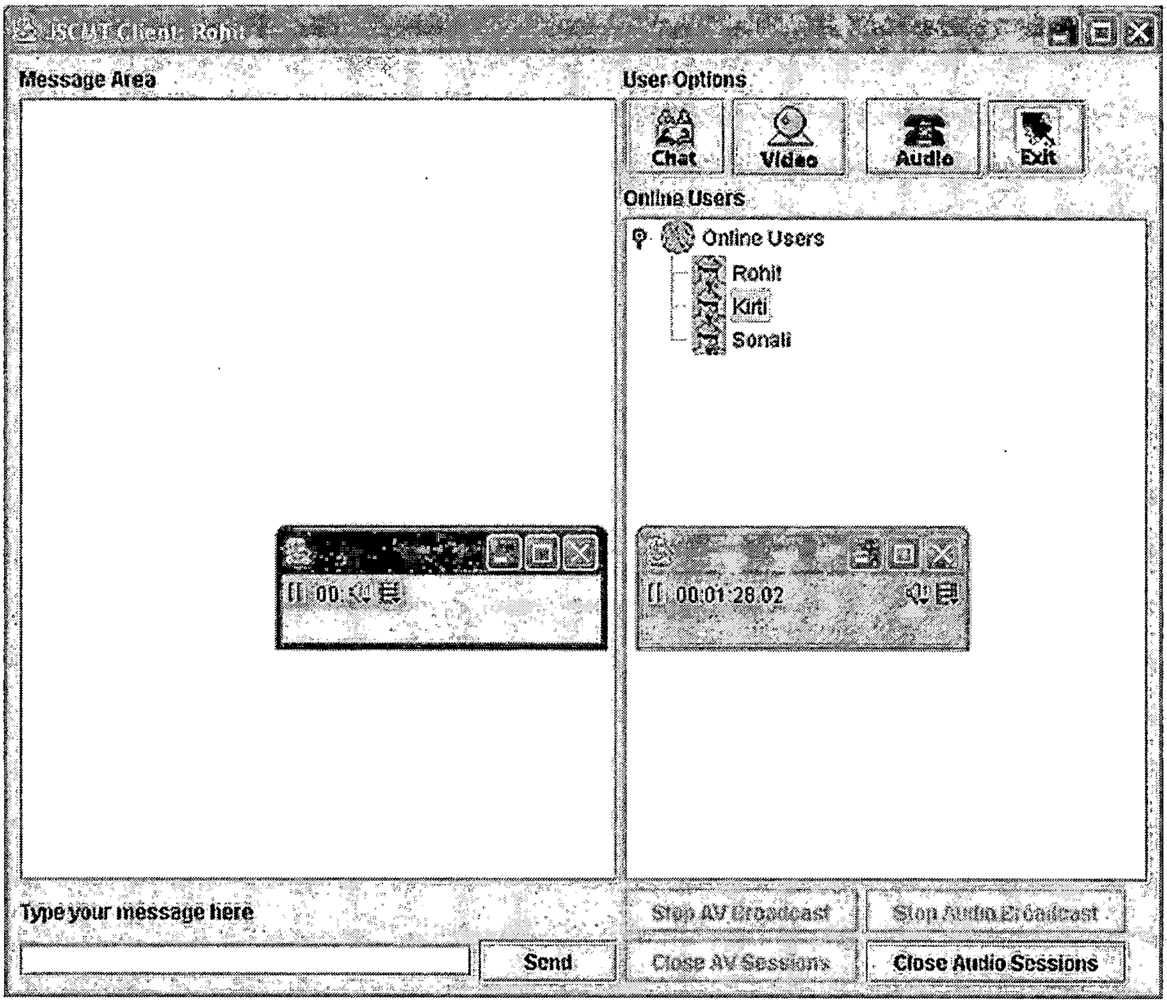


Figure 3.14 Participating in Private Audio Conference

## Close Audio Session

JSCMT client participating in private audio session can close audio session by clicking "Close Audio Sessions" button. This is shown in Figure 3.15.

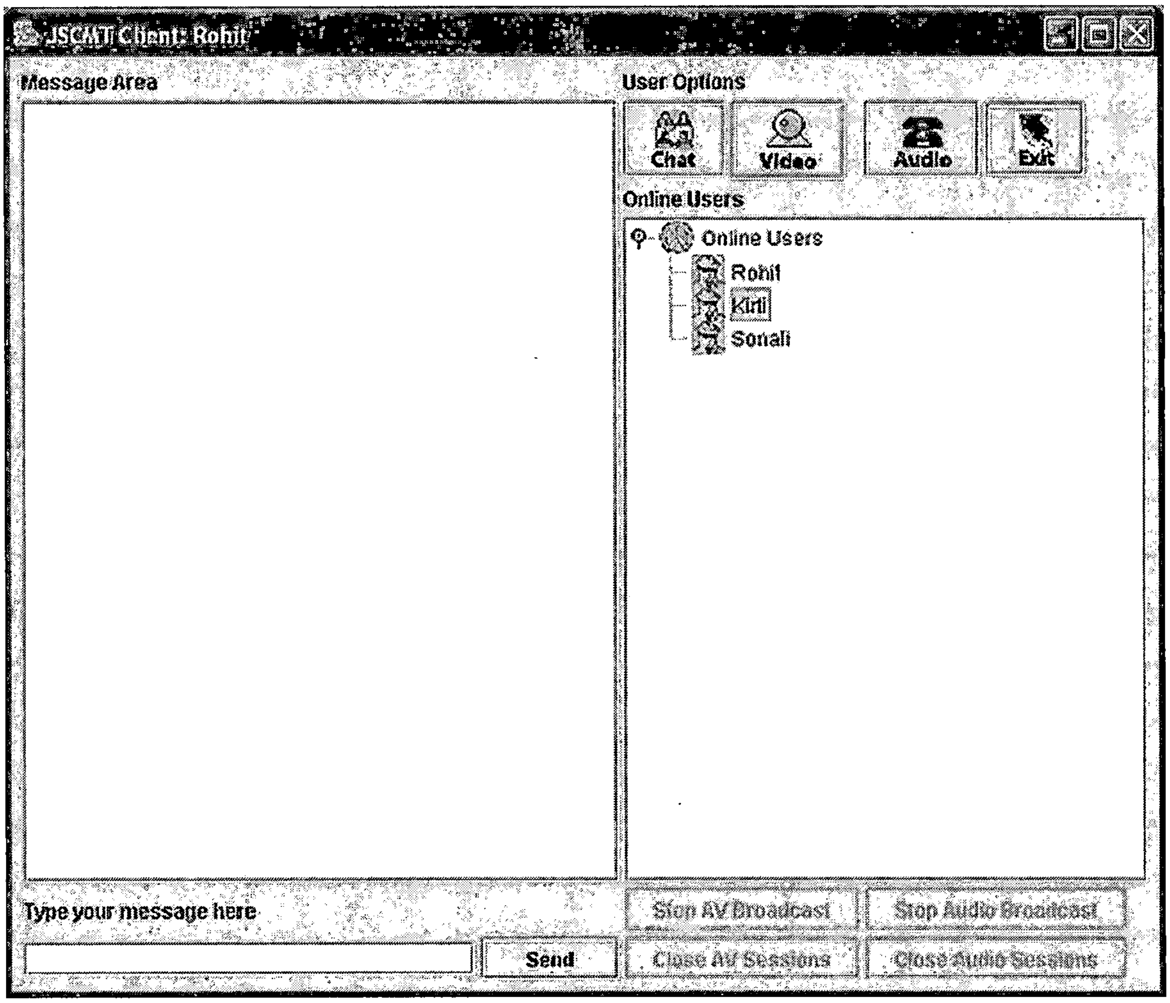


Figure 3.15 Close Audio Session

Viewing Client's Web Cam

The steps involved in requesting to view another client's web cam are:

- Select the user from the list of online users displayed in the users online display area.
- Right click to pop up the context menu.
- Select "View Web cam" option.

Client requesting to view another client's web cam is shown in Figure 3.16 and Figure 3.17 shows client receiving this request.

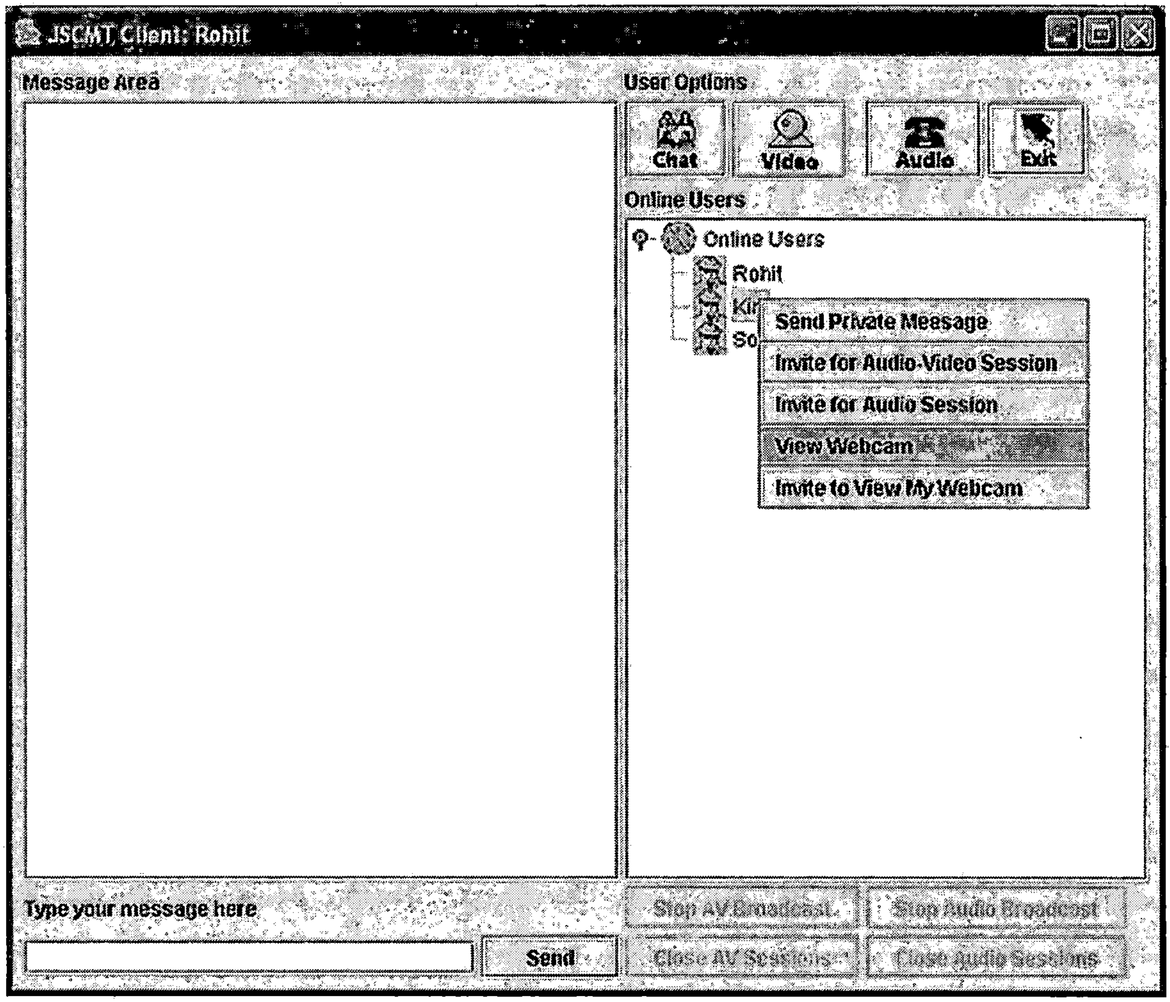
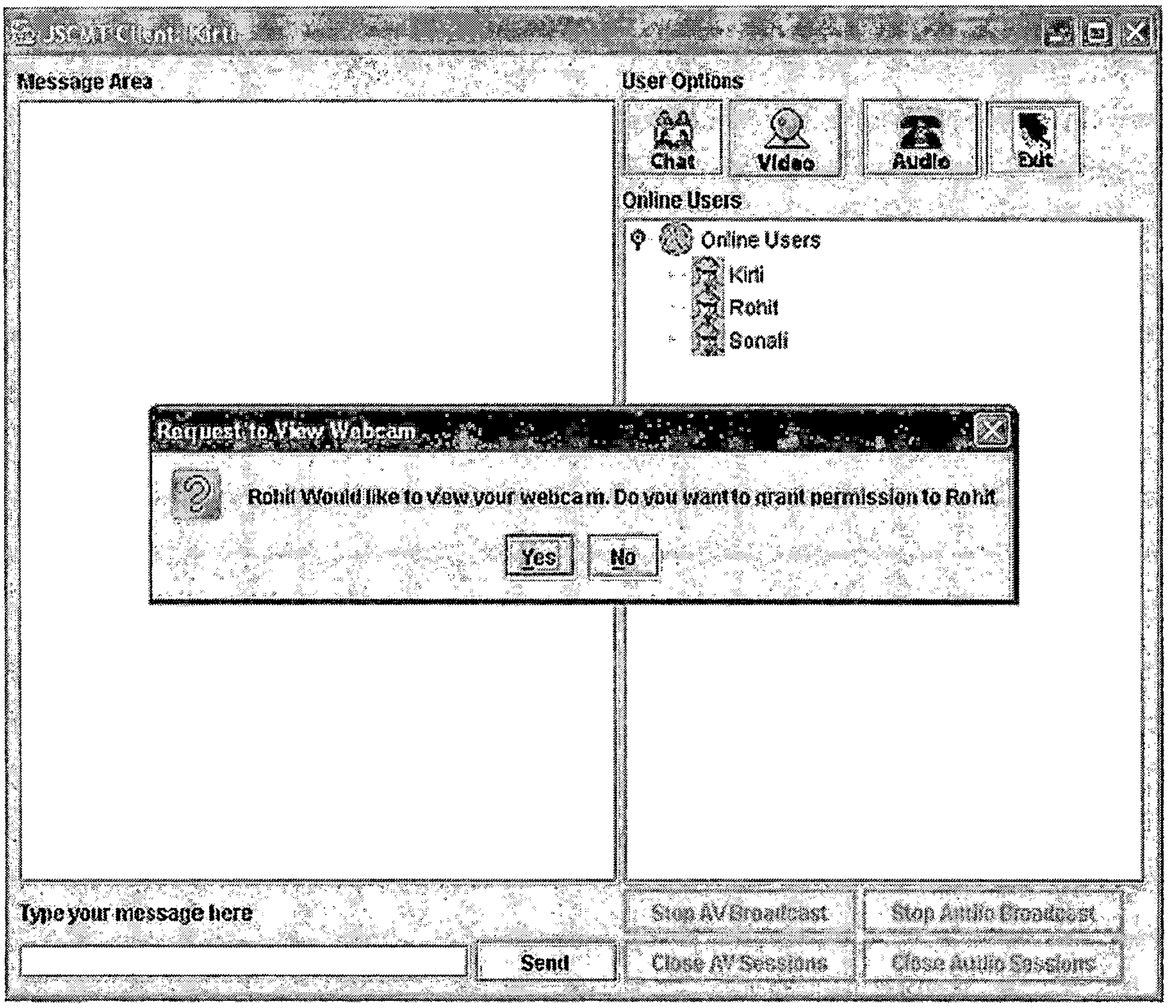


Figure 3.16 Request to View Client's Web Cam



Rohit Would like to view your webcam. Do you want to grant permission to Rohit

Figure 3.17 Receiver View Web Cam Request

## Inviting Client to View One's Web Cam

The steps involved in inviting client to view one's web cam are:

- Select the user from the list of online users displayed in the users online display area.
- Right click to pop up the context menu.
- Select "Invite to View My Web cam" option.

Client inviting another client to view one's web cam is shown in Figure 3.18 and Figure 3.19 shows invited client viewing inviting client's web cam.

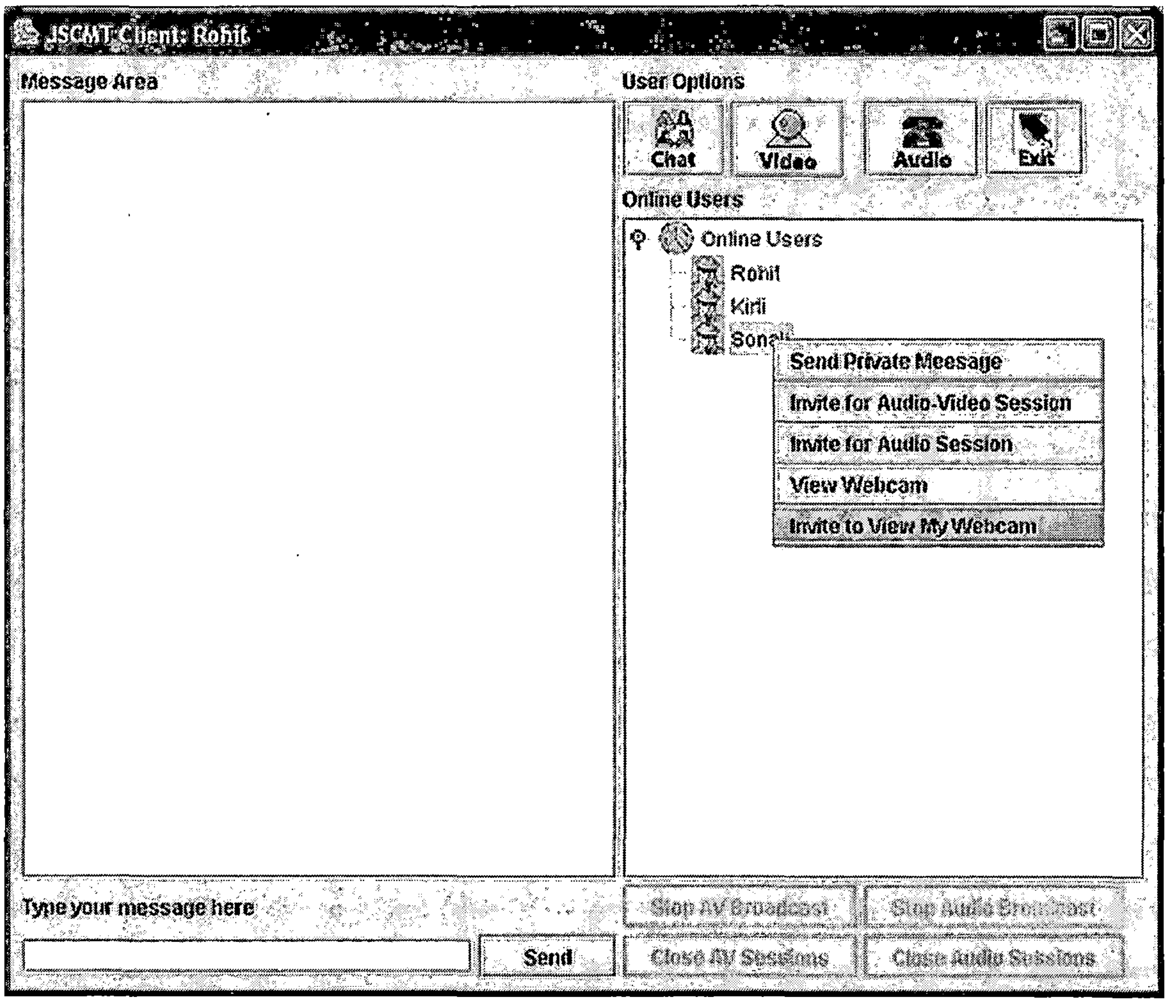


Figure 3.18 Invite to View One's Web Cam

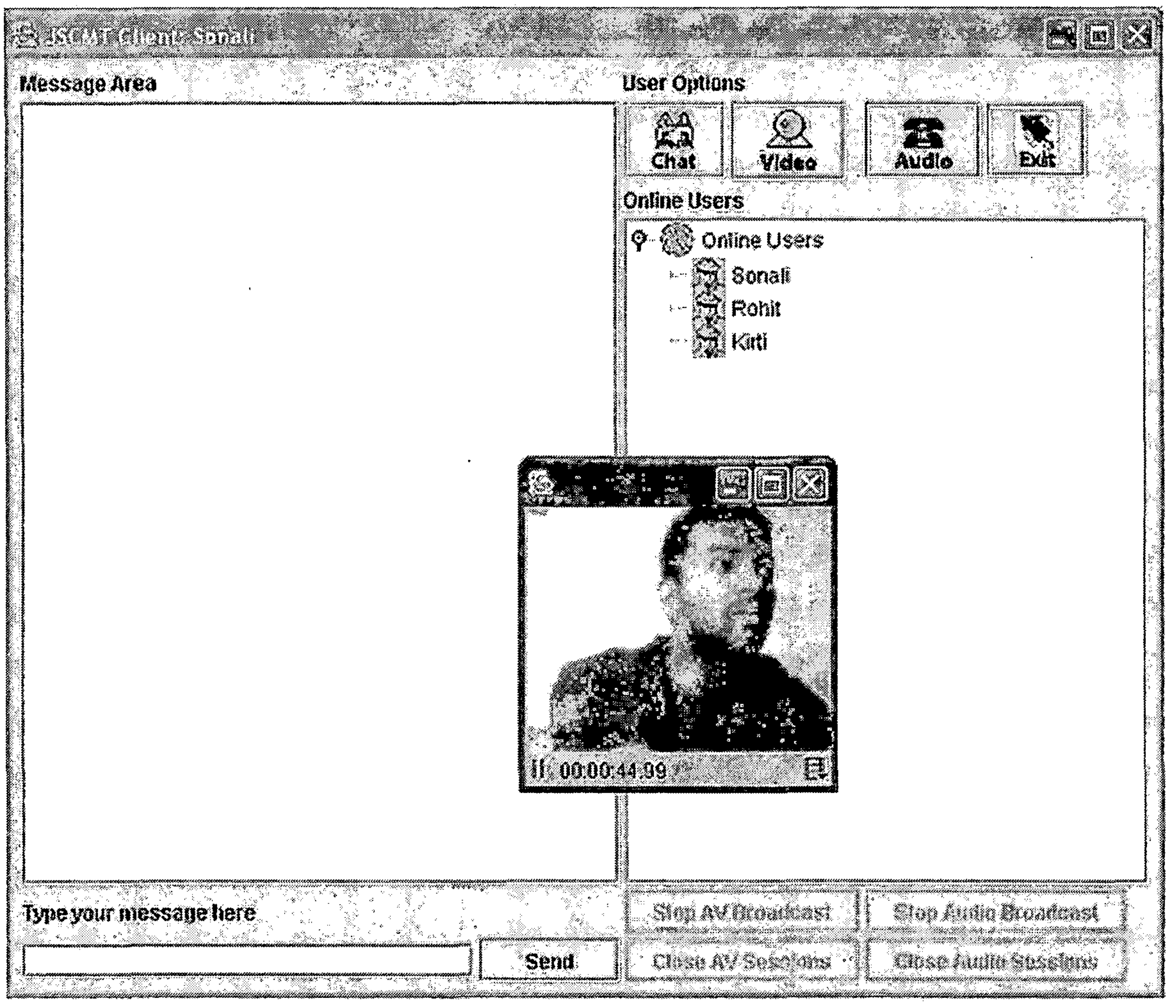


Figure 3.19 View Inviting Client's Web Cam

## Private Text Messaging

Client can exchange private text messages with another client by using private text messaging feature of JSCMT. The steps involved sending private text messages to another client are:

- Select the user from the list of online users displayed in the users online display area.

- Right click to pop up the context menu
- Select "Send Private Message" option

Client inviting another client for private text messaging is shown in Figure 3.20. Clients actively engaging in private text messaging can be seen in Figure 3.21.

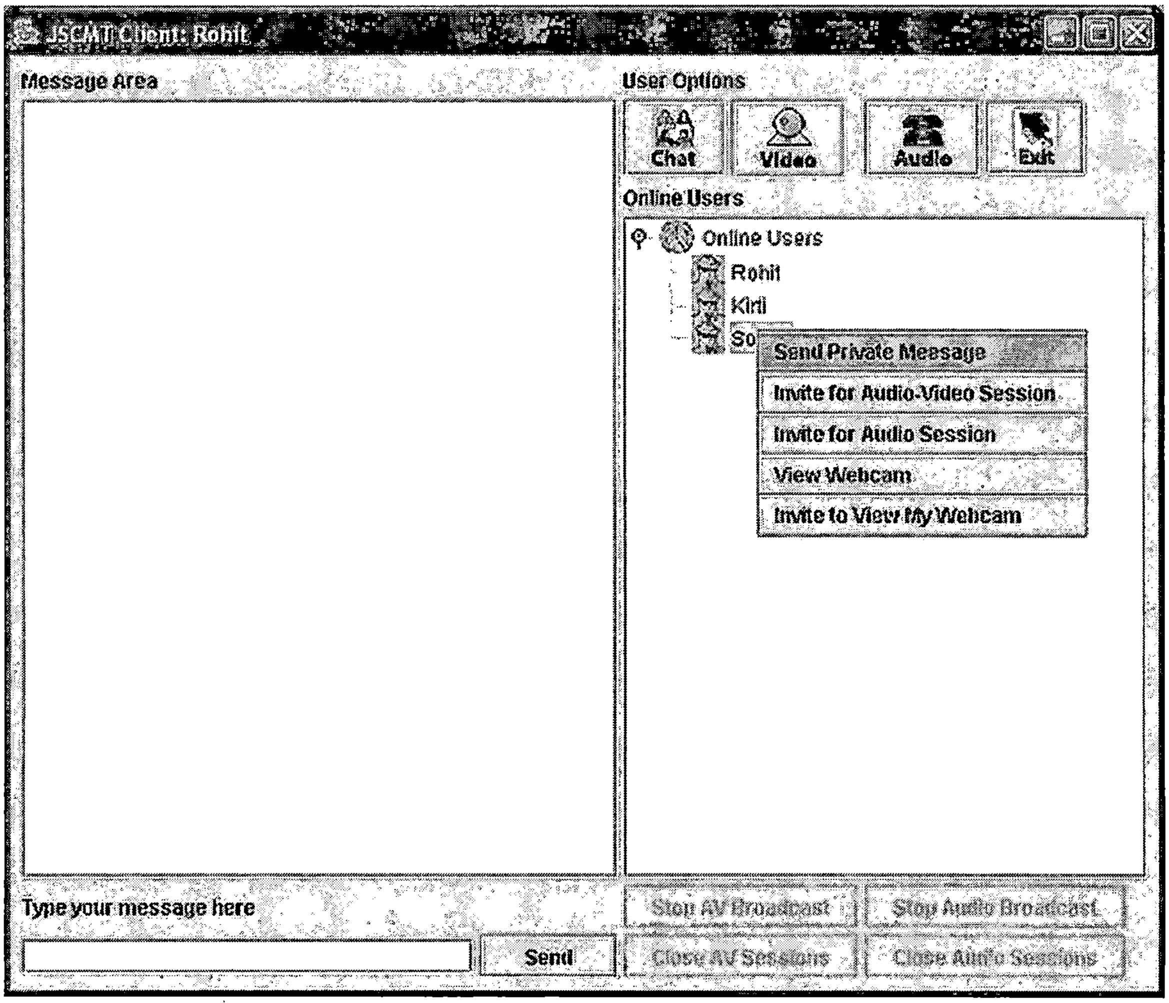
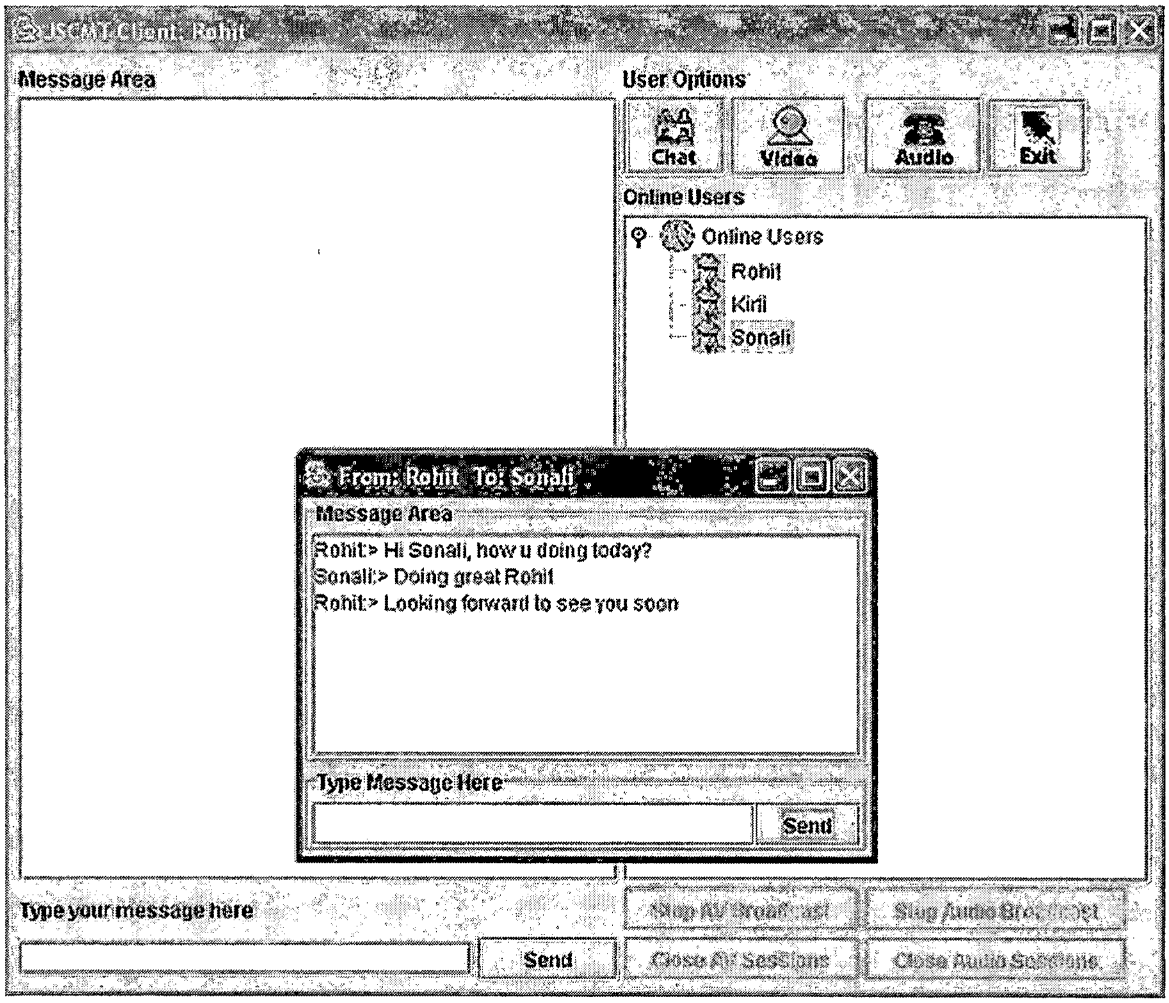


Figure 3.20 Invite for Private Text Messaging



Rohit:> Hi Sonali, how u doing today?
Sonali:> Doing great Rohit
Rohit:> Looking forward to see you soon

Figure 3.21 Participate In Private Text Messaging

## Summary

This chapter explains step by step procedures for using various features of JSCMT Application. It elaborates various functions of JSCMT with the help of user interfaces.

# CHAPTER FOUR

# CONCLUSIONS AND FUTURE DIRECTIONS

## Conclusions

Motivation to collaborate users with common interest but located in different geographical locations, using various communication tools resulted in the development of JSCMT. Streaming of real-time audio and video over the network for interactive communication between users has increased widely. JSCMT incorporates streaming of real-time audio and video into its audio-video conferencing tool.

JSCMT is developed as a Java application. JSCMT architecture is based on conventional client/server model. Server is only responsible for creating Session and Channel that are eventually used by collaborating clients. This design dictates very little processing load on the server whereas most of the request processing is done by the clients. JSCMT uses two types of protocols, TCP for text messaging and RTP for media streaming over the network. JSCMT uses JSDT API to implement communication part of the project and JMF API to implement streaming and receiving real-time audio and video.

## Future Directions

JSCMT has a good scope for future enhancements. Functionality of JSCMT application can be extended by providing support to allow the user to participate in more than one audio video session. Authentication process can be further refined. Functionality of JSCMT can be extended to support more than one Session. Various other features that can be added to JSCMT in future are discussed in this chapter.

### Refining Authentication Process

Authentication process implemented by JSCMT is at a low level. JSCMT server authenticates the client by using the information sent by the client and matching it with the information maintained by the server in the form of text file. This process can be further enhanced by implementing database on the server which stores the user specific information of all the clients.

### Support for Multiple Sessions

Currently creation of only one Session is supported by JSCMT. Support for Multiple Session will allow a new Session to be created for every new location. JSCMT client can participate in any of the available Sessions. This adds a distributed feature to the application.

Support for File Transfer

Functionality of exchanging files between the users is one of the important features of any collaborative application. Currently JSCMT does not implement this feature. File transfer support can be added as a attractive feature to JSCMT by implementing an appropriate file transfer utility.

Support for Multiple Audio-Video Session

Currently JSCMT allows user to participate in only one audio-video session at any given time. This means if user is broadcasting audio-video to all the users online and wants to invite another user for private audio-video session, client needs to stop the broadcast first. If the mechanism is implemented to create instances of data sources for audio and video upon request without requiring initializing the capture devices, client can participate in more than one session.

## Summary

In this chapter, future recommendations were discussed that can add new features to the application.

## REFERENCES

[1]. "IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1993)"

[2]. Bruce Eckel, "Thinking in Java", 3rd Edition (2002), Prentice Hall, ISBN: 0131002872

[3]. Martin Fowler and Kendall Scott, "UML Distilled", 2nd Edition, Addison-Wesley Publishing Company, ISBN: 0-201-65783-X

[4]. Rob Gordon, Stephen Talley, Robert Gordon., "Essential JMF - Java Media Framework", 1st Edition (1999), Prentice Hall, ISBN: 0130801046

[5]. Rich Burridge, "Java Shared Data Toolkit User Guide", 1999, Sun Microsystems

[6]. Java 2 Platform, Standard Edition (J2SE) 1.4.1 API Specification, Sun Microsystems, http://java.sun.com/j2se/1.4.1/docs/api

[7]. Java Media Framework (JMF) 2.1.1 API Specification, Sun Microsystems, http://java.sun.com/products/java-media/jmf/2.1.1/apidocs/

[8]. Java Shared Data Toolkit (JSDT) 2.0 API Specification, Sun Microsystems, http://java.sun.com/products/java-media/jsdt/reference/api/

[9]. O. Kim et al., "Issues in Platform-Independent Support for Multimedia Desktop Conferencing and Application Sharing," Poc. Seventh IFIP Conf. on High Performance Networking (HPN'97), Chapman & Hall London, 1997, pp. 115-139