Development of Dialogue-Based Object-Oriented Application

Md. Mehedi Masud, **Khandaker Shaidul Islam**, and **Md. Kamrul Islam**

Computer Science and Engineering Discipline
Khulna University,
Khulna-9208, Bangladesh
Email: mmehedi@bttb.net.bd

Abstract

Communication through the Internet becomes a cost-effective way to conduct a real-time voice conversation between computers and is currently an enthusiastically accepted component. In this paper, an application is developed that provides a real time conversation facility through network, that can also work on the Internet. The communication media provided are text and audio. The developed application is based on client/server architecture. UML class diagram, sequence diagram, use-case diagram are used for describing the system. Java is used as the programming language for the implementation of the system.

1. Introduction

Today, people are interested with applications like MSN - messenger, YAHOO - messenger to talk on Internet [1]. These applications are specially designed for real time communication. But main problem is that the programming ideas behind this technology are not disclosed. The application proposed in this work is based on object-oriented approach. The client part provides user interface as it shows a list of users to select and then communicate with one of them. The server part manages communication between two users. The client uses TCP connection for control transfer and UDP datagram for data transfer. The server waits on different ports for TCP connection and UDP datagram. Java's multithreading capability has been extensively used for simultaneous operations such as sending text message by a user while talking to a user.

2. Description

Although many formal and/or intuitive methods are available in the identification of the system requirements, case model is used here, as it is an ideal method for establishing the requirements for interactive application. The actors in the system are users. The users located at different places and connected to the network. They can communicate with each other by sending message and/or talking. Each user has to register into the system before communicating. Server as defined in Figure 2.1 mediates the communication between users.

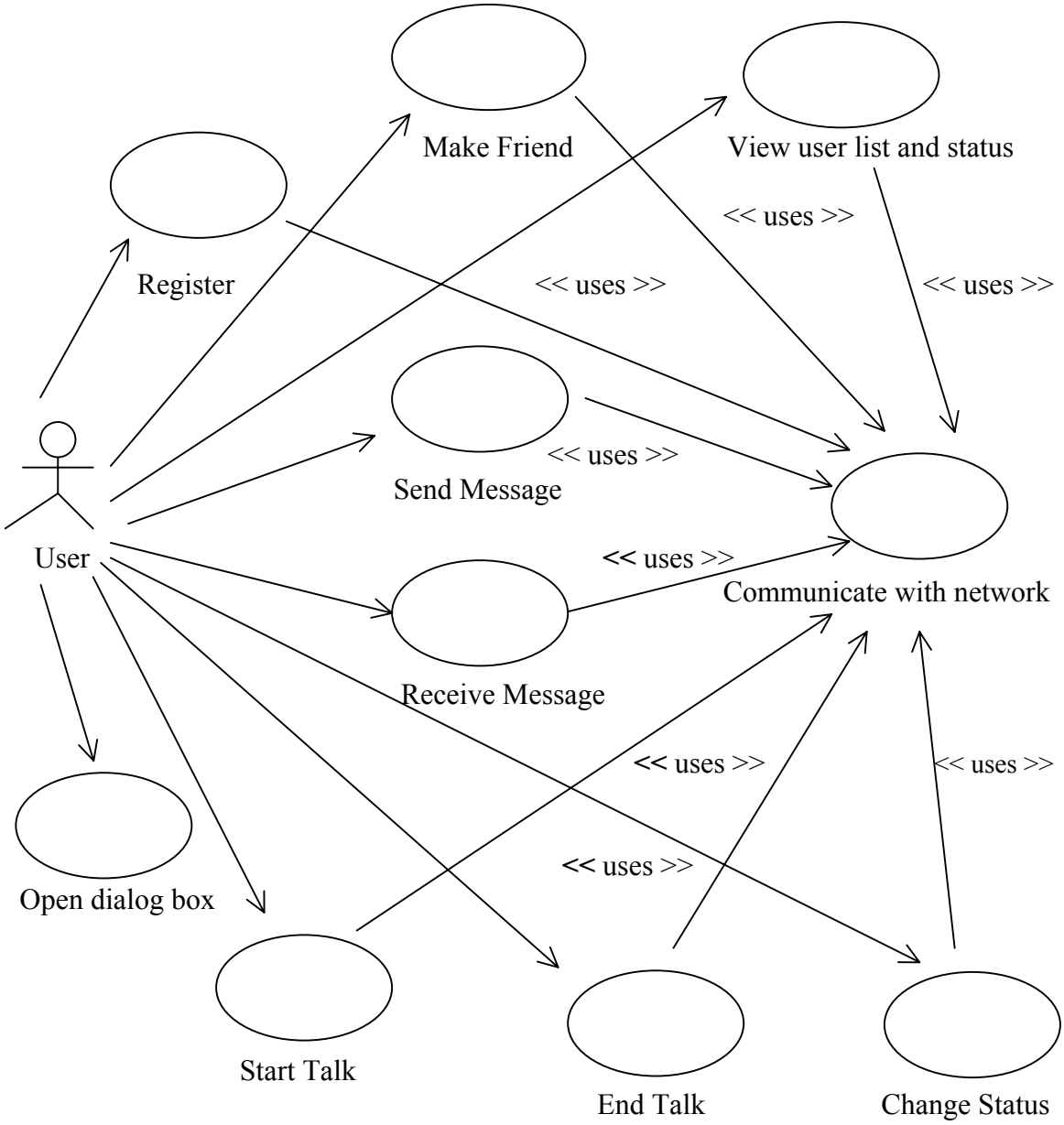

Figure 2.1 Use cases of the system

2.1 Class Descriptions

To meet some of the above requirements the classes defined for the application are discussed in the following sections in terms of their functionality and relationship with other classes. Firstly client side classes, then server side.

2.1.1 Client Side Classes

The objective of these classes is to handle the requests of a user at the client end. The classes are described as follows. The main class in the client side is OnLineClient class. The other classes are MessageReceiver, MsgSendErrorReceiver, UserListManager; RequestToTalkManager, WaitToTalk, SoundSender and SoundReceiver which are instantiated as needed.

OnlineClient Class

The OnLineClient controls all the functions at the client side. This class interacts with users and handles a user requests through different methods and instances of inner classes. The methods are as follows.

StartHandler:

This method notifies the server about the host it is running on and requests the users list.

ExitHandler:

This method informs the server that client is exited from the host it is running on

LogOnHandler:

This method manages log on of a user. It takes User Name from user and calls the following procedure. The procedure returns true if log on successful, otherwise it returns false. WFR means wait for response.

boolean procedure LogOn (UserName)
begin
Step 1.Request connection to the server, WFR
Step 2. Response comes
Step 3. Send log on request and UserName to the server, WFR
Step 4. If log on successful returns true
Step 5. Returns false
end

Procedure LogOn

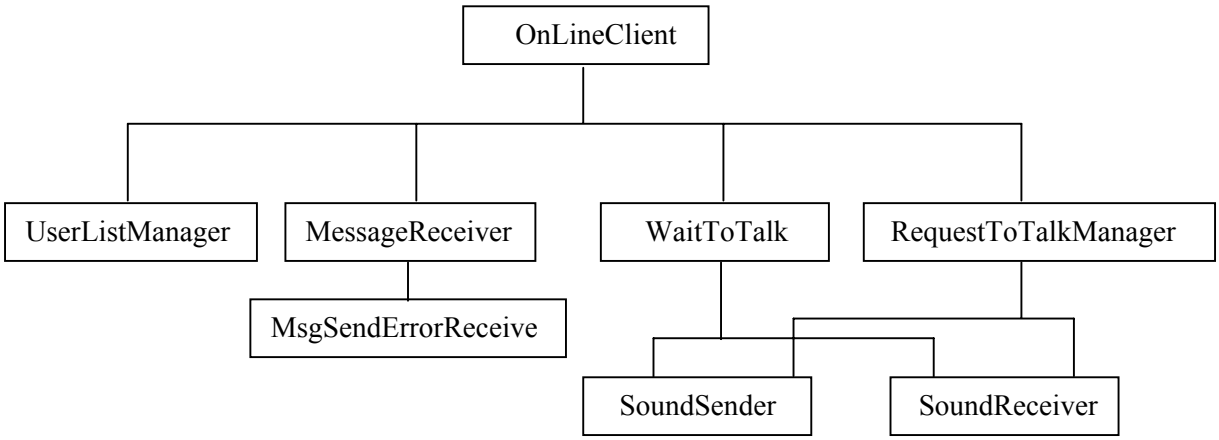

Figure 2.2 Class diagram for client

LogOutHandler:

This method sends the user name and a request to log out to the server when the user wants to log out.

MessageSender:

This method manages sending of message to a remote user. It takes message and remote user name from local user. It then makes packet from the message and sends the packet to the server to send it to the remote user. The restriction here is that the maximum size of message can be as large as the size of data a packet can contain, which is chosen here 8192 bytes. When the message size is large than the packet data size, the message is not sent and an error message is reported to the local user. When the packet is sent to the server, it instantiates MsgSendErrorReceiver class to receive message from the server, regarding confirmation of the delivery of the message.

StartTalking

This method initiates talking between two users. It takes talking request from local user to talk to a remote user and lets the local user to talk with remote user if the remote user agrees to talk. If the remote user refuses to talk or timeout occurs it informs that to the local user. It calls WaitToTalk to get message from the server for remote user response.

StopTalking:

This method sends request to the server to end talk with a remote user when the local user wants to end talk with that user. It also stops SoundSender and SoundReceiver objects.

MsgSendErrorReceiver Class

The object of this class waits on a specified port to receive a UDP packet containing confirmation about the packet delivery from the server. It sends an error message to the local user if the packet is undelivered.

MessageReceiver Class

The object of this class receives text message from the server, which is sent from a remote user, and displays it to the local user. It waits on a specified port to receive UDP packets containing the message.

UserListManager Class

It manages the user list i.e. adds user to the list or remove from the list depending on their arrival or departure. It waits on a specified port to receive add or remove request from the server along with user name to add or remove.

SoundSender Class

The object of this class is handled by sending of voice data from the local user to a remote user via the server. The procedure is as follows.

void procedure RecordAndSend()

begin
Step 1. Wait for available data in audio device input buffer
Step 2. Read a specified amount of data
Step 3. Make packet with this data and remote host ip
Step 4. Send this packet to the server
Step 5. Go to step 1
end

Proceduce RecordAndSend

This class is istantiated and started when two users agree to talk and stopped when the users finish talking. The amount of data chosen here is 4410 bytes.

SoundReceiver Class

This class is instantiated and started when two users are involved in talking and stops when they end talking. It receives packets from the server, which are sent by remote client and plays the sound data in it. The procedure is as follows.

procedure ReceiveAndPlay( )

begin
Step 1. Wait for a packet on a specified port
Step 2. When a packet comes, extracts the data from the packets
Step 3. Put the data into audio device output buffer
Step 4. Go to step 1
end

Procedure ReceiveAndPlay

RequestToTalkManager Class

The object of this class receives request to talk or end talk with remote user from the server. When it receives request to talk it informs the local user and sends local user response to the server. When it receives request to end talk it stops SoundSender and SoundReceiver objects.

WaitToTalk Class

The object of this class waits for remote user response to talk with local user from the server. After getting response it informs local user by displaying appropriate message. It instantiates SoundSender and SoundReceiver class for sending and receiving voice data if the remote user agrees to talk.

2.1.2 Server Side Classes

The objective of these classes is to service client requests. The main class here is OnLineServer. The other classes are, ConnectionHandler, TextExchanger, Sound-Exchanger, SignInHandler, SignOutHandler, ListSender, and SessionManager. They are instantiated as needed.

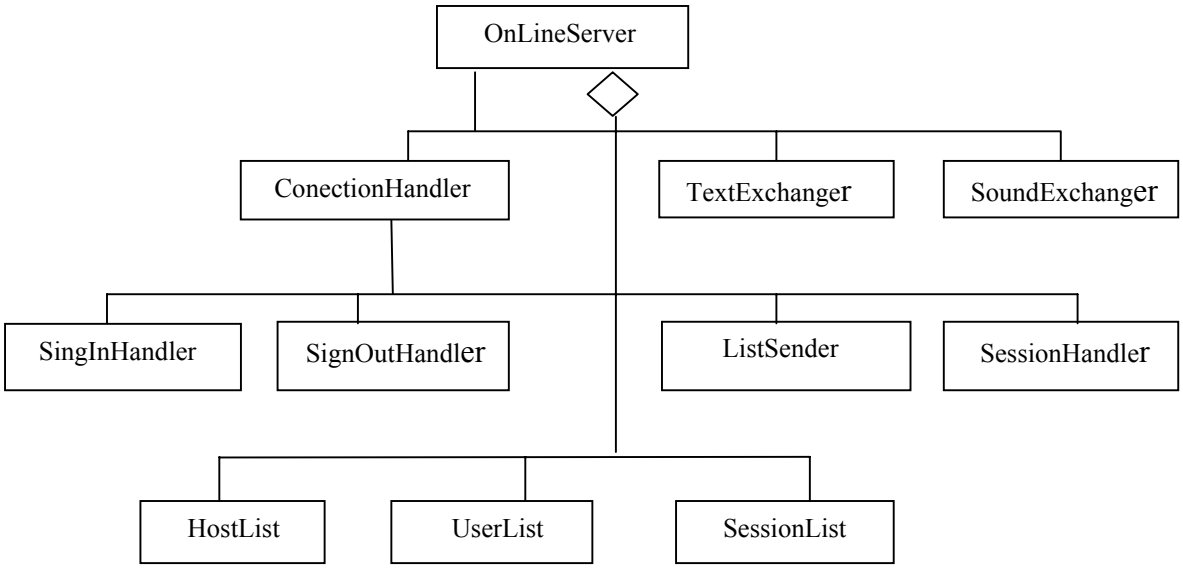

Figure 2.3 Class diagram for server

OnLineServer Class

The instance of the class initiates server process. It controls all operations of the server. It instantiates three classes -- ConnectionHandler to handle requests from clients, TextExchanger to exchange text message between two clients, and SoundExchanger to exchange voice data. It contains three objects which maintains different information, HostList for host information, UserList for user information, SessionList for user session.

ConnectionHandler Class

The instance of this class actually interprets and handles users request. It keeps waiting on a specified port for the client opening server socket. When it receives any request from a client, it serves the request immediately and/or instantiates classes – SingInHandler, SignOutHandler, ListSender, and SessionHandler. The requests it handles are:

Start:
This request is received when the client end starts its execution. It instantiates ListSender to send user list to the client.

Exit:
This request is received when the client is exiting from the host.

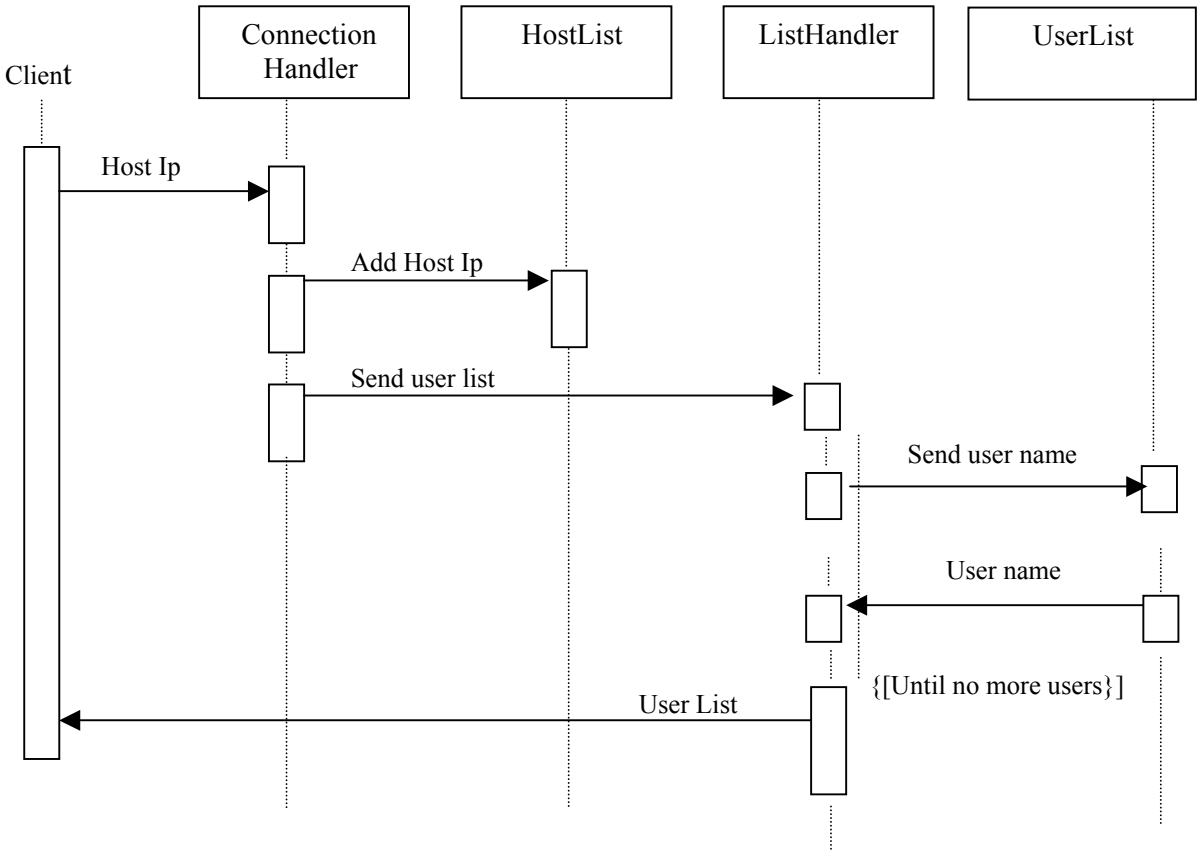

Figure 2.4 Sequence diagram for data flow in Start process

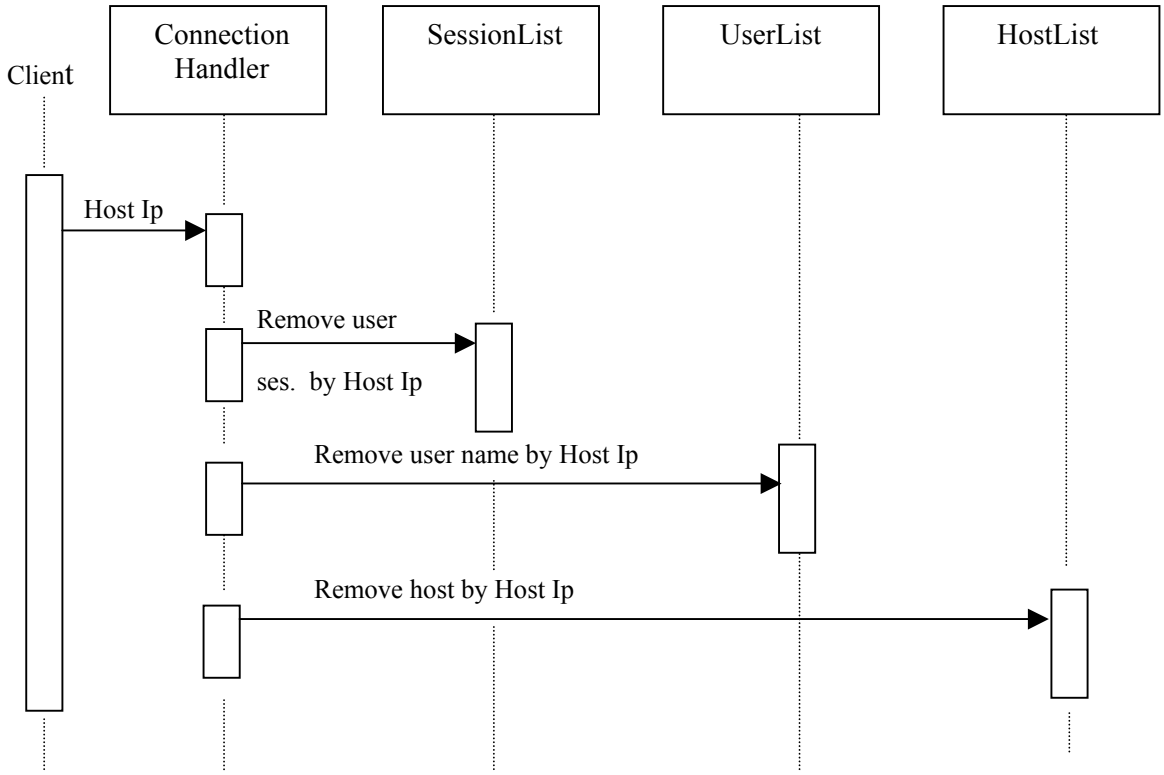

Figure 2.5 Sequence diagram for data flow in Exit process

SignIn:
When a user at a client wants to logs on the client sends this request. When the following procedure returns true, a successful sign-in message is sent to the client, otherwise sign-in failed message is sent to the client. The procedure takes user name as argument and works with UserList object.

```
boolean procedure DoSignIn(UserName)
begin
  If (UserInList (UserName))
    return false;
 else
 begin
   AddToList (UserName);
   Calll SignInHandler (UseName);
 end
   end
   Procedure  DoSignIn
```

SignOut:
When a user wants to log out at a client the client sends this request.

StartTalk:
A client sends this request when a user want's to talk to another. The ConnectionHandler instantiates SessinHandler to handle this request.

EndTalk:
A client sends request when the user at the client wants to end talk with a user at another client. The SessionHandler is instatiated by ConnectionHandler to service this request.

HostIp:
When a client sends this request, the host ip address is returned with the given user name.

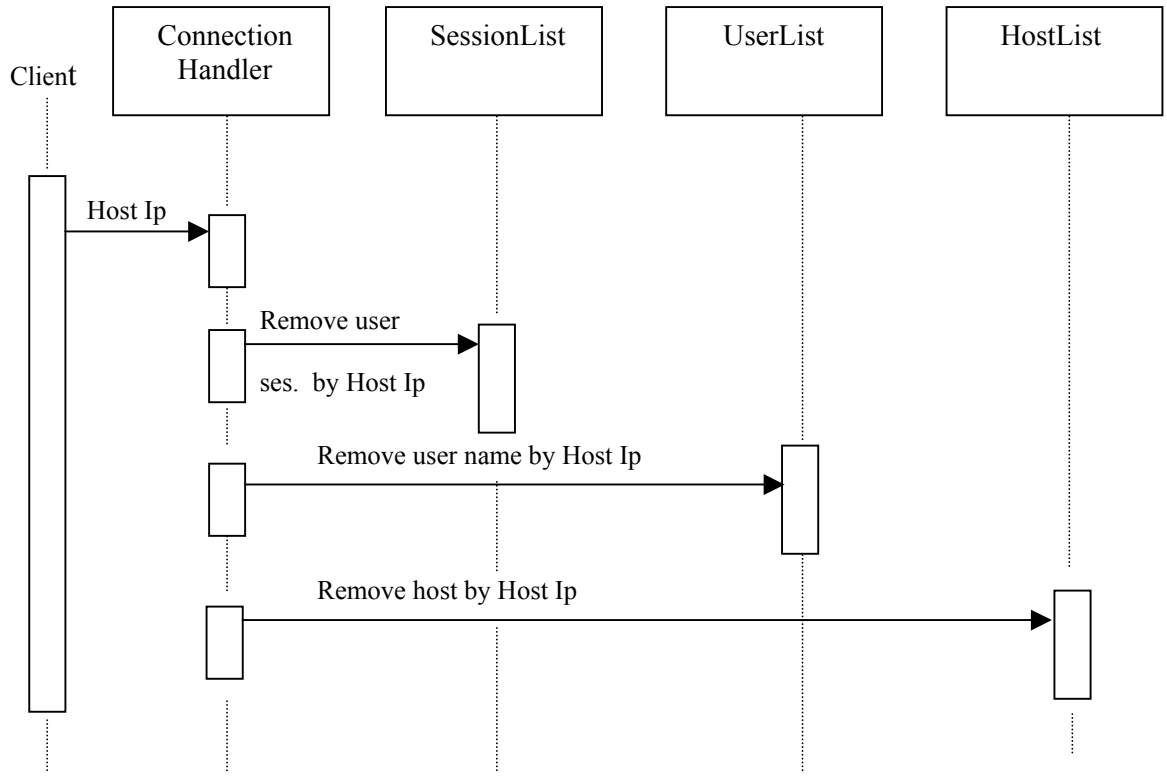

Figure 2.6 Sequence diagram for data flow in SignOut process

SignInHandler Class

This class is instantiated when a new user logs on. The objective of this class is to send this new user name to all the clients to add to their user list. It uses its UserList for this purpose. It does this by the following procedure.

void procedure AcionForSignIn(UserName)

begin
Step 1. iff(IsEmpty()) return;
Step 2. HostIp=getHostIp (UserName);
Step 3. ConnectTo(HostIp);
Step 4. Send UserName to add;
Step 5. CloseConnection ();
Step 6. Go to step 1;
end

Procedure ActionForSignIn

SignOutHandler Class

This class is instantiated when a user logs out. The objective of this class is to send the user name to other clients to remove the user name from their user list. It uses its UsreList for this purpose.

void procedure AcionForSignOut(UserName)

begin
Step 1. if(IsEmpty()) return;
Step 2. HostIp=getHostIp (UserName);
Step 3. ConnectTo(HostIp);
Step 4. Send UserName to remove;
Step 5. CloseConnection ();
Step 6. Go to step 1;
end

Procedure ActionForSignOut

ListSender Class

This class is instantiated when a client end first starts its execution. This class sends the user list to the client. If no user is present in the list, it sends a message to the client indicating this.

SessionHandler Class

The object of this class starts and closes a talking session between two users. It works by handshaking messages between the users.

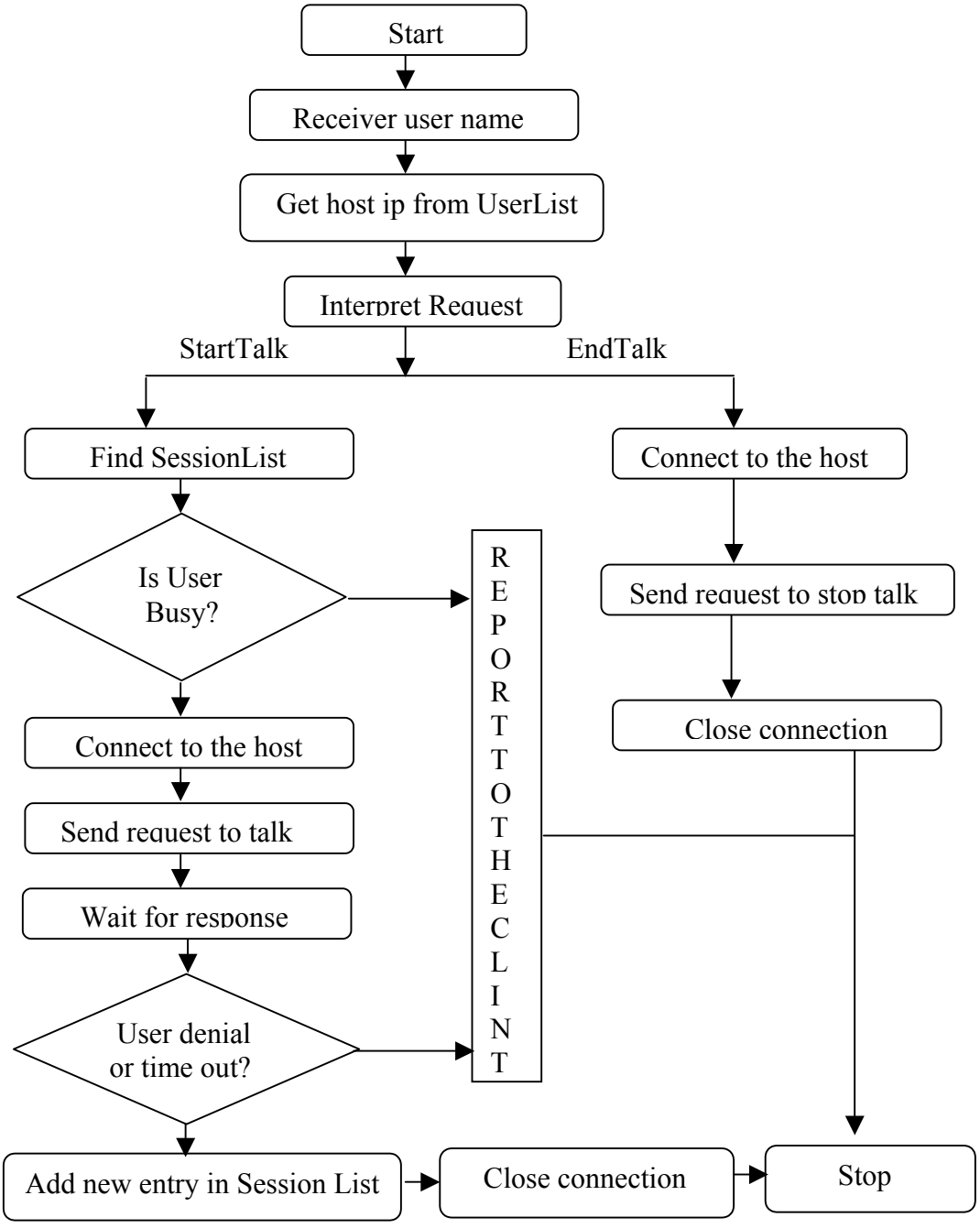

Figure 2.7 States of SessionHandler object

TextExchanger Class

This class is instantiated when the server starts. This objective of this class is to receive UDP packets containing text message from a client and sends it to the destined client. It listens on a specified port to receive packets. It uses PacketTransfer procedure to transfer packet. The procedure returns true if the packet is delivered false otherwise. Then TextExchanger object notifies these events to the sending client.

SoundExchanger Class

This class is instantiated when the server starts. The objective of this class is to receive UDP packets containing voice data from a client and sends this packet to the destined client. It listens on a specified port to receive packets. It also uses PacketTransfer procedure to transfer packet.

```
boolean procedure PacketTransfer(Packet)

begin
  Step 1. Extract data and destination host
          ip from the packet
  Step 2. Make new packet from this ip
  Step 3. Send this packet
  Step 4. If (IsSucessful ()) return true
  Step 5. return  false
end
```

Procedure PacketTransfer

3. Conclusions

It is advantageous to choose a client-server design structure that provides for fast development and requires only a small amount of effort to develop the application system. When the application system is reviewed and accepted by users, the design structure can then be modified to achieve other goals, such as enhanced capabilities and increased performance. The Internet protocol was not intended to carry real time data. Therefore special protocol should be used along with the Internet protocol for real time data transmission over Internet.

The application developed here has some limitations. When a large number of users will be in the system, the performance of the system will degrade severely, as the users and their session information is stored in the form of a list to reduce the complexity of the application. A database should be maintained at the server to reduce the access time of that information. Packets may arrive out of order. So packet sequencing can be used for reliable transmission.

References

[1] Coad, Y., "The Object–Oriented Analysis and Design Method", URL:http://www.is.cs.utwente.nl:8080/ dmrg/ OODOC/oooc/oo-12.2.html

[2] Mike W. Wang, "Client Server Application Design Structure", URL: http:// www.interex.org.

[3] Md. Nazmus Saadat, Ripon Kumar Banik, "Development of a Net Conversation Application", Undergraduate Thesis, 2001.

[4] Schulzrinne, H., "Issues in Designing a Transport Protocol for Audio and Video Conferences Group Internet-draft", *IETF*, 1993.

[5] Scott Oaks & Henry Wong, *Java Threads,* O'Reilly & Associates, Inc, Second Edition.1999

[6] Java Sound API, URL: java.sun.com.