PEER-TO-PEER VOICE COMMUNICATION NETWORK

BE CONNECTED ...

TEAM NAMES

MAHMOUD ALI EL-SAYED MOHAMED

BASANT MOHAMED ATYA IMAM

KAMELIA KAMAL RAGHEB

UNDER SUPERVISION

DR \ WALEED KHEDR

TA \ AMR ABDELLATIF

MAIN PROBLEM

- Voice applications doesn`t provide full services to the user.
- Quality of communication was depending on both client and server.
- During the conference call if one client has slow connection it will affect the conversation to all clients connected.
- So, it was important to provide a full system to solve this problems.

PROBLEM IDEA

- Providing one technique for data exchange between clients will make the quality of all services the same.
- We provide a strong server that control the operations between clients.
- The server will control the process of broadcasting of data to the clients.
- Each client is connected to other clients throw the server and there are open streams between them.
- The client will send data packets to the sever and the server will broadcast them to it`s target.
- So the quality of communication to one user is same as multiple users.

APPLICATION STRUCTURE

- This application is consists of three main classes

I. Chat Message Class (implements Serializable).

II. Server Class.

III. Client Class.

Note : There are another classes which provide services to the client class.

1- CHAT MESSAGE CLASS

- In this project we were interested in multiple types of data.
- Data types are like (Strings, Integers , Bytes …).
- But how will the server listen to multiple data types from all clients at once.
- So , it was important to provide a new date type (Serializable object) to solve this problem.
- The ChatMessage class is the solution.
- This class will solve the problem as we will use to marshal data between client and server.
- ChatMessage (int Type , byte[] message , String username , String[] destination).

SERVER CLASS

The Server is responsible for establishing connection between the clients, listen for data ,and broadcast them to its target.

Server Structure

- ServerSocket
- ClientThread
  - Socket socket
  - ObjectInputStream sInput
  - ObjectOutputStream sOutput
  - String username
- ArrayList<ClientThread> al.
- Synchronized broadcast method.

Server

CLIENT CLASS

- Client class is responsible for preparing data to be established to other clients.
- The client start connecting to the server to be able to exchange data with others.

Client Structure

- Socket socket.
- ObjectInputStream sInput.
- ObjectOutputStream sOutput.
- String username.
- String [] destination.
- Thread ListenFromServer.

DATA LIFE CYCLE

- The data to reach to it`s target pass throw some steps.
- First, the data is prepared as A ChatMessage object.
- ChatMessage cm = new ChatMessage(int Type , byte[] message, String username , String []destination);
- The first argument is the type of the message (TEXT, SOUND,CONTROL, IMAGE,…).
- The second is the message data.
  - Data must be converted to array of bytes
- The third argument is the user name of the sender of the message.
- The fourth argument is the destination of the message.
  - Array of Strings contains the user names of the clients which the sender want to send the message to them.

DATA LIFE CYCLE CONT

- After preparing the Chat message the object is sent to the server.
- The server will check the type of the message to know what to do.
- Then it will broadcast the message to its destination.
- The client in the other side will receive the message from the server.
- Then check its type and handle the message.

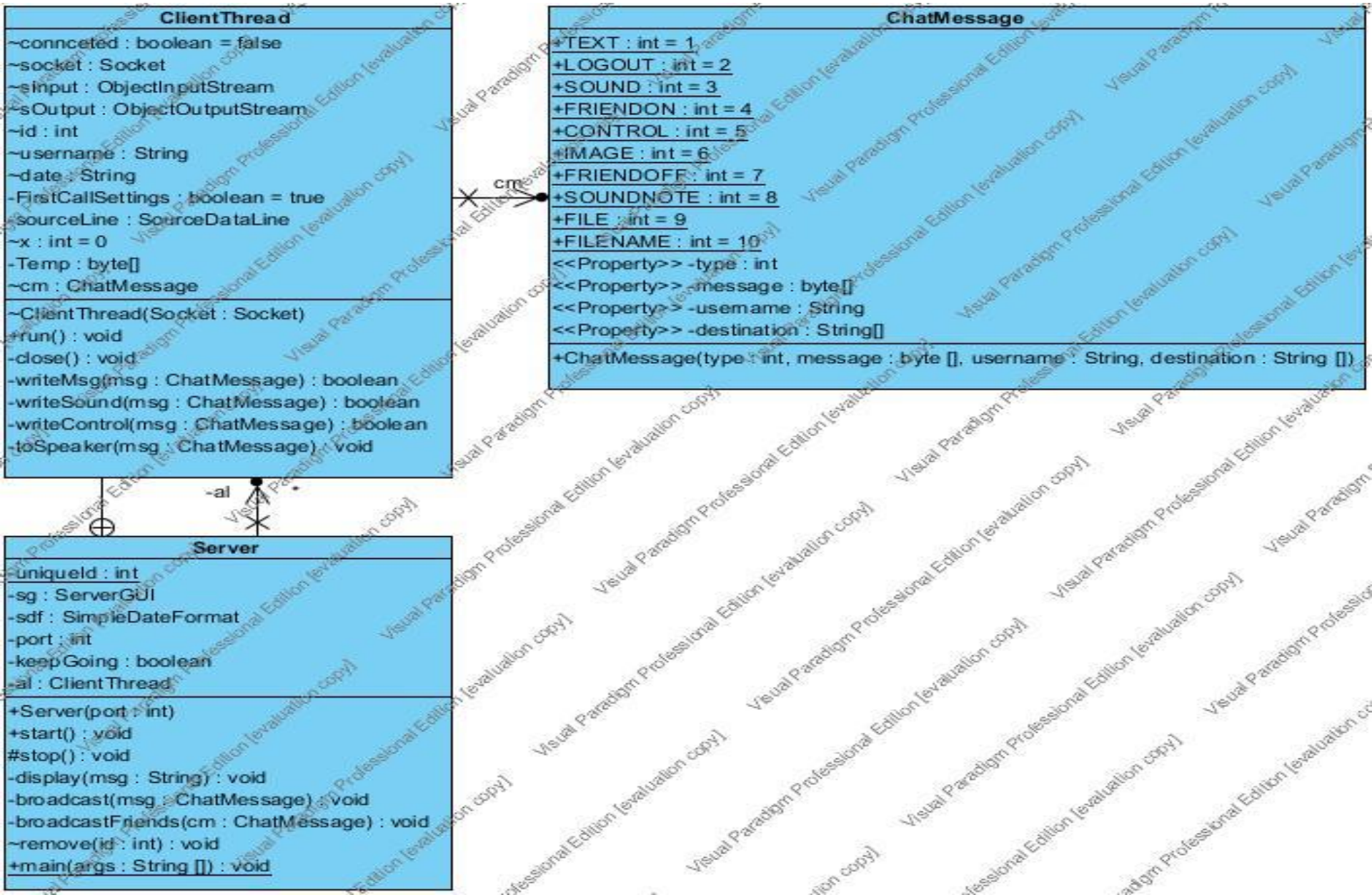
CLASSES DATAGRAMS

ChatMessage

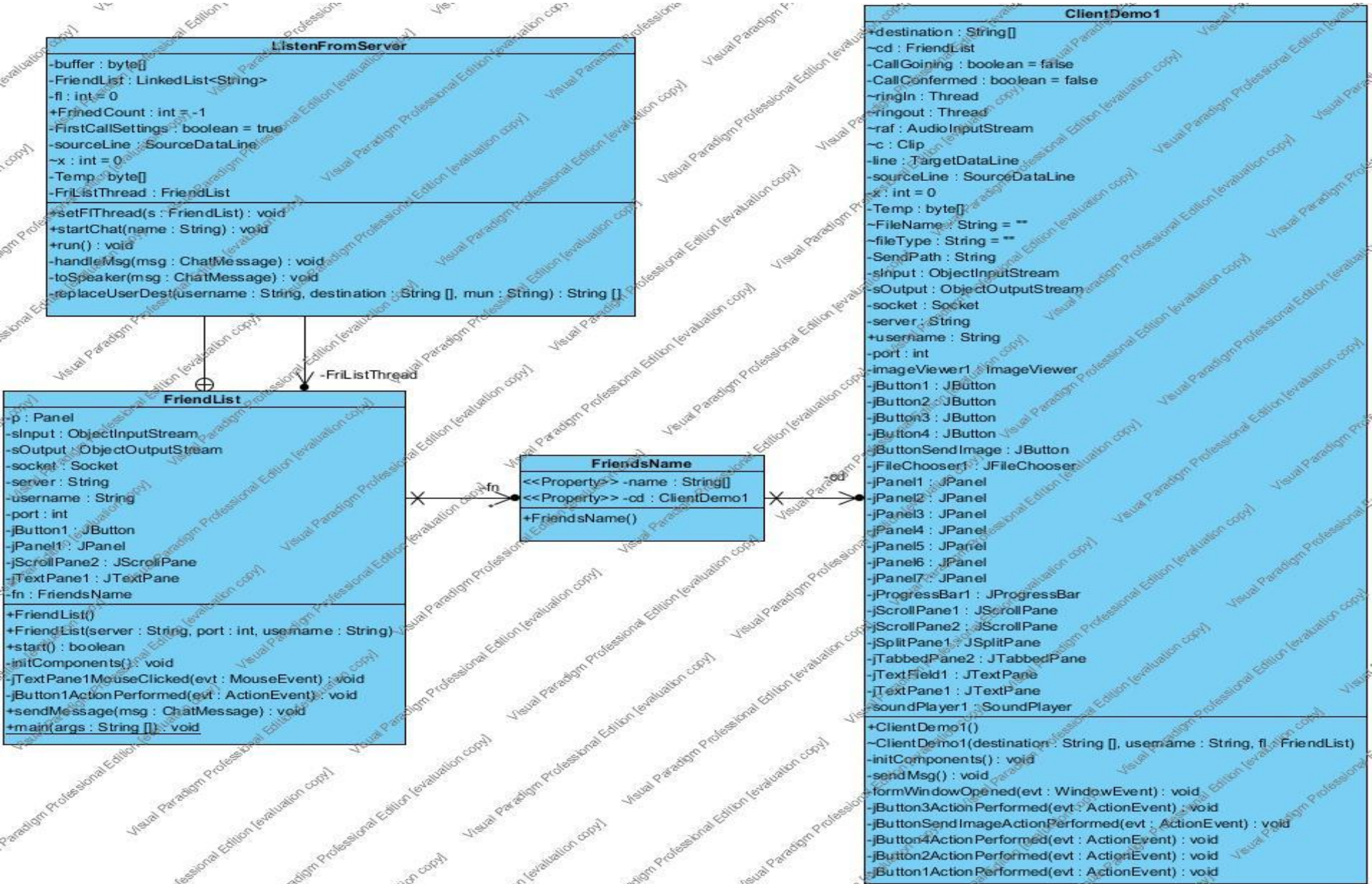
+TEXT : int = 1
+LOGOUT : int = 2
+SOUND : int = 3
+FRIENDON : int = 4
+CONTROL : int = 5
+IMAGE : int = 6
+FRIENDOFF : int = 7
+SOUNDNOTE : int = 8
+FILE : int = 9
+FILENAME : int = 10
<<Property>> -type : int
<<Property>> -message : byte[]
<<Property>> -username : String
<<Property>> -destination : String[]

+ChatMessage(type : int, message : byte [], username : String, destination : String [])

CLASSES DIAGRAMS CONT



CLASSES DIAGRAMS CONT



MESSAGE TYPES

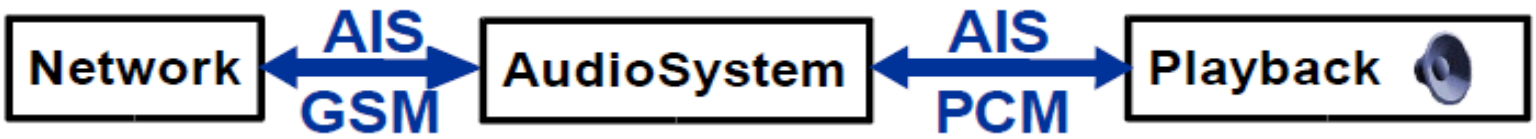
- TEXT (for text message)
- LOGOUT (to inform the server about the log off of a client)
- SOUND (for sound calls)
- FRIENDON (to inform the clients about the logging on of a friend)
- CONTROL (used in handshake of sound calls , Files ,images , notes share).
- IMAGE (for image send)
- FRIENDOFF (to inform the clients about the logging off a friend)
- SOUNDNOTE (for sound notes send)
- FILE (for file send)
- FILENAME (used to inform the client the name & size before sending file)

HOW A SOUND CALL ESTABLISHED ??

1. Control messages for handshaking .
- STARTCALL
- CALLACCEPTED
2. First configuration for data lines
- TargetDataLine.(microphone line)
- SourceDataLine.(speakers)
- AudioFormat.
3. Start thread for capturing audio from microphone and send them to the server.
4. Start thread for playback array of bytes received from other clients.

AUDIO PLAYBACK

- Use SourceDataLine for streaming playback.
- receives audio data from network in an AudioInputStream.
- converted PCM stream is read in a thread and written to the SourceDataLine.
- use SourceDataLine's buffer size for reading and writing.



AUDIO CAPTURE

- Use TargetDataLine for streaming capture
- The TargetDataLine is wrapped in an AudioInputStream so that it can be converted to the network format with AudioSystem
- A capture thread reads from the converted AudioInputStream and writes it to the network connection's OutputStream.