

A Presence-based Messaging Application ELEC 6861 Project

Masood Khosroshahy

m.kh@ieee.org - www.masoodkh.com

Instructor: Prof. Roch Glitho

December 12, 2009

Contents

1	Intro 1.1	Dduction Choices	3 3
2	Imp	lementation	3
3	Prot	tocols Design: Messages and Rules	8
	3.1	Presence	8
	3.2	Session setup and termination	10
	3.3	Messaging	12
	3.4	Exchanging files	15

1 Introduction

As per the specification, a presence-based messaging and file-exchange application has been designed and implemented. Here is the scenario:

- Clients connect to the server and immediately declare their presence.
- A connected client can initiate a session by sending the request to the server along with the preferred number of clients in the session.
- The server application checks the number of available clients for the session.
- The server application initiates a session between the clients, if preferred number of clients are available.
- When the session is underway, participants can exchange messages and files.
- Only the session initiator can terminate the session.

1.1 Choices

The implementation has been done in Java using Socket programming (NetBeans IDE v.6.7.1 has been used for development). Protocols for signaling (presence and session establishment/termination), message and file exchanges have been implemented on top of TCP (i.e., respective Socket type has been chosen) due to the reliability that it offers.

2 Implementation

The implementation is split into five classes and will be described briefly as follows:

A general Data object (see Figure 1), that is passed between the client and server, which carries signal, message and file depending on how it's been constructed on the originating side. On the receiver side, Data object is read directly from the Socket and analyzed/parsed according to its type. Using this method, Java API has been used to its fullest potential to reduce the manual delimiter placing/parsing.

The client (see Figures 2 and 3) is implemented in a single class which has numerous functions that are invoked based on the events triggered through the client GUI.

The server application has a main class which relies on two other classes for its operations (see Figures 4 and 5); one object per client which holds all client-specific references and one thread per client which interacts with the client as long as the client is connected. The server application creates two server Sockets (one socket for receiving, and responding to, client signals and one socket dedicated to sending data to the client) at startup. It then listens to incoming connections and will create a thread to interact with a new client.

	📃 Data
	Attributes
private int S	IGNAL = 0
private int N	IESSAGE = 1
private int F	<u>ILE = 2</u>
private int d	ataType = SIGNAL
private Strin	ig senderNickname
private Strin	ig textContent
private Strin	ig messageContent
private Strin	g filename = null
private byte	fileByteArray[0*] = null
private int n	umberParticipants = 0
	Operations
public void	setDataType(int dataType)
public int g	etDataType()
public void	setTextContent(String textContent)
public Strin	g getTextContent()
public void	setMessageContent(String messageContent)
public Strin	g_getMessageContent(_)
public void	setSenderNickname(String senderNickname)
public Strin	g getSenderNickname()
public void	setNumberParticipants(int numberParticipants)
public int g	etNumberParticipants()
public void	setFileByteArray(byte fileByteArray[0*])
public byte[0*] getFileByteArray()
public void	setFilename(String filename)
public Strin	g getFilename()

Figure 1: UML Class Diagram [Exchanged Data Object between Server and Client]

	🗮 MasoodConnect
	Attributes
pac	kage Socket socket = null
pac	<age objectoutputstream="null</td"></age>
pacl	<u> kage ObjectInputStream objectInputStream = null</u>
pacl	kage Socket dataSocket = null
pacl	<u> cage ObjectInputStream dataObjectInputStream = null</u>
priva	ite Data receivedData
priva	ite Data signal
priva	ite Data message = null
priva	ite String filename = null
priva	ite boolean isConnected = false
priva	ite boolean isListening = false
priva	te JButton Connect
priva	te JButton Disconnect
nriva	ite JButton Exit
nriva	ite TextArea Messare
prive	ita TavtAras PacaivadMassara
prive	ite IButten SendMessage
prive	te TextField ServerAddrees
prive	te TextField ServerPad
priva	ite Jiextriela Selvelfon
priva	ite JrextArea Statuswessage
priva	ite JButton cancelSession
priva	Ite JiextField connected Clients
priva	ite JlextField fileNamePath
priva	ite JLabel jLabel1
priva	ite JLabel jLabel2
priva	ite JLabel jLabel3
priva	ite JLabel jLabel4
priva	ite JLabel jLabel5
priva	ite JLabel jLabel6
priva	ite JPanel jPanel1
priva	ite JPanel jPanel2
priva	ite JPanel jPanel3
priva	ite JPanel jPanel4
priva	ite JPanel jPanel5
priva	ite JPanel jPanel6
priva	ite JScrollPane jScrollPane1
priva	ite JScrollPane jScrollPane2
priva	ite JScrollPane jScrollPane3
priva	ite JTabbedPane jTabbedPane1
priva	ite JButton localAddresses
priva	ite JTextField locallPaddress
priva	ite JTextField nickname
priva	ite JTextField numberParticipants
priva	ite JButton reset
priva	ite JButton selectFile
priva	te JButton sendFile
priva	te JTextField serverDataPort
prive	ite JButton sessionRequest
nrive	ite (Panel tabl ocalAddresses
prive	ite IDanal tabSandMassana
prive	ite IDenal tehSattinge
priva	te i Dutten undete Connected Olicete
huas	ne opurion updateoonnectedoilents

Figure 2: UML Class Diagram [Client-side: Main App(Part 1)]

evt)

Figure 3: UML Class Diagram [Client-side: Main App(Part 2)]



Figure 4: UML Class Diagram [Server-side: Main App]



Figure 5: UML Class Diagram [Server-side: One thread per client]

3 Protocols Design: Messages and Rules

In describing the sequence of messages for different aspects of the application, it might be clearer to show the screenshots of the actual developed application, rather than using abstract sequence diagrams. That is why in what follows, the designs of simple protocols for presence, session establishment/termination, message and file exchanges are demonstrated using screenshots.

3.1 Presence

Client needs to set the server address along with signal and data ports (see Figure 6).

Settings	Local Session	Message File
Client Nicl	kname: Bob	
Address:	localhost	Signal Port: 8888
Conne	oct Disconnect	Data Port: 8889
Received M	lessage	•
Received M Connected	lessage clients	↓ ▼ Update
Received M Connected Status Mes	lessage dients	↓ ↓ Update
Received M Connected Status Mes	lessage clients ssage erver opened.	Update Reset

Figure 6: Presence: Client Connects to Server

Client sets its Nickname which should be unique; otherwise, it will not be accepted by the server. When two sockets are created, client immediately sends its nickname. After receiving the nickname, server creates the Output Streams, starts a thread, passes the relevant information to the thread and keeps all the references in an instance of "MasoodConnect-ServerClientInfo" Class (see Figures 7, 8 and 9).

```
private void ConnectActionPerformed(java.awt.event.ActionEvent evt) {
    trv {
        if (!isConnected) {
             if (ServerAddress.getText().isEmpty() || ServerPort.getText().isEmpty()) {
                 StatusMessage.setText("Server address or port not set correctly.");
             } else {
                isConnected = true;
                 socket = new Socket(ServerAddress.getText(), Integer.parseInt(ServerPort
                 objectOutputStream = new ObjectOutputStream(socket.getOutputStream());
                 objectInputStream = new ObjectInputStream(socket.getInputStream());
                 dataSocket = new Socket (ServerAddress.getText(), Integer.parseInt(server
                 dataObjectInputStream = new ObjectInputStream(dataSocket.getInputStream(
                 StatusMessage.setText("Sockets to server opened.");
                 signal = new Data();
                 signal.setTextContent(nickname.getText());
                 objectOutputStream.writeObject(signal);
```

Figure 7: Presence: Client Creates the Sockets and Sends its Nickname



Figure 8: Presence: Server Receives Client's Nickname Upon Connection - Output



Figure 9: Presence: Server Receives Client's Nickname Upon Connection - Code

3.2 Session setup and termination

After connecting to the server, a client can request a session. The client sends to the server the number of desired participants. If the server is not busy and that number of clients are available, server picks randomly from the connected clients to form a session of the desired number (see Figures 10, 11 and 12).

ettings	Local	Session	Message	File
Session R	equest			
			-	
Num	ber of pa	rticipants:	3	
	Cancel S	ession	Request	Cossion
	cuncer J	COSION	nequest.	10331011

Figure 10: Session: Bob requests a session of 3



Figure 11: Session: Server receives the request and forms the session.

on grante	Reset		
4	SHI S	•	Evit

Figure 12: Session: Server confirms the session formation to Bob and mentions the participants.

A session can be canceled only by its initiator. If another session participant tries to cancel the session, the server sends a signal informing the client that its request has been ignored (see Figures 13, 14, 15 and 16).



Figure 13: Session: Alice sends a session cancellation signal to server

Reque	Reset			
4	III			Exit

Figure 14: Session: Alice is notified that her request has been ignored since she is not the session initiator.

Administrator: Command Prompt - java - jar MasoodConnectServer.jar 8888 8889 C:\MasoodConnectApp\Server>java - jar MasoodConnectServer.jar 8888 8889 Bob Connected. Alice Connected. John Connected. A session of 3 is requested by Bob Session cancellation request received from Alice Session cancellation request received from Bob

Figure 15: Session: Bob sends a session cancellation signal to server

Session Cancelled.	_	Reset
		Exit

Figure 16: Session: Server cancels the session created by Bob.

3.3 Messaging

When a session is underway, participants can send message to all session participants. To send a message, a client sends a signal to the server. The type of signal is "DistributeMessage" and its content is the message itself. When the server receives such a signal, it first checks whether the sender is part of the session. If so, the server distributes the message to all session participants (see Figures 17, 18, 19 and 20).

Masoou	Connect C	lient		
Settings	Local	Session	Message	File
Client Me	ssage to	Session		
Hello fol	ks!			
				-
				Send
Received	Message			
Bob: Hello	folks!			^
				•
	d clients			
Connecte	050 - COSD - 5690		1	Update
Connecte			1.1	
Connecte				opusio
Connecte Status Me	essage			
Connecte Status Me	essage I. Participa	nts: Bob Joh	n Alice	Reset

Figure 17: Messaging: Bob sends a message to the session (he receives his own message as well).

	Lane 2				
client Nickname:	Allce				
Server		Signal	Dort-	8888	
Address. 1000	D:	Data Dr	Data Port:		
Connect	Disconnect	Data Pt	Data Port: 8889		
nessand Messan	.				

Figure 18: Messaging: Alice receives Bob's message

Settings Loca	I Session A	Aessage File	
Client Nickname	e: John		
Server			
Address: 10	calhost	Signal Port:	8888
Connect	Disconnect	Data Port:	8889
eceived Messag	je		
ob: Hello folks!			
ob. Heno tono:			

Figure 19: Messaging: John also receives Bob's message

```
private void SendMessageActionPerformed(java.awt.event.ActionEvent evt) {
   String fromUser = null;
   try {
     fromUser = Message.getText();
     if (fromUser != null) {
        signal = new Data();
        signal.setSenderNickname(nickname.getText());
        signal.setTextContent("DistributeMessage");
        signal.setMessageContent(fromUser);
        objectOutputStream.writeObject(signal);
   }
}
```

Figure 20: Messaging: Client sends a signal to the server which contains the message.

3.4 Exchanging files

In much the same way as messaging, session participants can exchange files. A client selects a file and creates a Data object containing a Byte Array which in turn contains the Bytes read from the file. The client then sends a signal to the server. The type of signal (Data object) is "DistributeFile" and its content is the aforementioned Byte Array. When the server receives the signal, it checks whether the sender is part of the session. If so, it distributes the Data object to all session participants. Each of the session participants saves the file to disk from the Byte Array as soon as they receive the Data object from the server.

🛃 Masood(Connect C	lient			
Settings	Local	Session	Message	File	
Send File File Name	to Sessio e/Path:	n			
C:\Maso	odConnec	:tApp\Client2	\pic1.jpg		
			Select File	Send	
Received I	Message				
				Ê	
Connected	d clients				
				Update	
	ssage				
Status Me		A file received from Alice and saved.			
Status Me A file receive	ed from Al	ice and save	ed. 🔺	Reset	
Status Me A file receive	ed from Al	ice and save	ed.	Reset Exit	

Figure 21: Exchanging files: Alice sends a file to the session (she receives her own file as well).



Figure 22: Exchanging files: Bob receives the file sent by Alice.



Figure 23: Exchanging files: Alice receives her own file as well.



Figure 24: Exchanging files: John receives the file sent by Alice as well.