2004

Mercury Instant Messaging System: A collaborative instant messaging tool

Tejaswi Srinivas

MERCURY INSTANT MESSAGING SYSTEM,

A COLLABORATIVE INSTANT MESSAGING 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

Tejaswi Srinivas

September 2004

MERCURY INSTANT MESSAGING SYSTEM,

A COLLABORATIVE INSTANT MESSAGING TOOL

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Tejaswi Srinivas

September 2004

Approved by:

[redacted]

Dr. Ernesto Gomez, Chair, Computer Science

August 26, 2004

Date

[redacted]

Dr. Richard Botting

[redacted]

Dr. George Georgiou

ABSTRACT

Instant messaging is a big boon to users who want to communicate in real-time. Instant Messaging applications are beginning to play an important role in various aspects of our lives, including education, business, healthcare, publishing and entertainment. Recent advances in computing and networking technologies have made these applications requiring real-time processing and high bandwidth feasible. 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.

Mercury Instant Messenger (MIM) is a Java Application, intended to collaborate a group of people not located in the same geographical location but working on the same project. MIM allows users to collaborate by using communications tools like, text based chat and whiteboard. MIM is developed using a platform independent programming language, Java AWT and socket programming. Java was chosen as programming language to implement this project with the assumption that users might be using different operating systems on their computers.

ACKNOWLEDGMENTS

I thank the faculty of Computer Science department for giving me an opportunity to pursue my Masters in Computer Science at the California State University, San Bernardino. I express my sincere appreciation to my graduate advisor, Dr. Ernesto Gomez who offered me this project and directed me through this entire effort. I also thank my other committee members, Dr. Richard Botting and Dr. George Georgiou for their valuable support.

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGMENTS iv

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER ONE: INTRODUCTION

1.1 Purpose of the Project 1

1.2 Scope of the Project 1

1.3 Limitations of the Project 4

1.4 Definitions, Acronyms, and Abbreviations ... 4

1.5 Overview 6

1.6 Overall Description 6

1.6.1 Product Perspective 6

1.6.2 System Interfaces 7

1.6.3 Server Interface 7

1.6.4 New User Registration Interface 8

1.6.5 Chat as Administrator Interface 9

1.6.6 User Client Interface 11

1.6.7 Chatrooms Interface 19

1.6.8 Chatrooms Toolbar Interface 20

1.6.9 Private Messaging Interface 21

1.6.10 Login/Settings Interface 22

1.7 Hardware Interfaces 23

1.8 Software Interfaces 23

1.9 Communications Interfaces 24

1.10 User Characteristics 24

1.11 Logical Database Requirements 24

CHAPTER TWO: SOFTWARE DESIGN

2.1 Overview 26

2.2 Mercury Instant Messaging System Methodology 26

2.2.1 Central Server 26

2.2.2 Remote Client 28

2.3 Mercury Instant Messaging System Architecture 30

CHAPTER THREE: SYSTEM VALIDATION

3.1 Unit Test 31

3.2 Subsystem Testing 33

3.3 System Testing 34

CHAPTER FOUR: CONCLUSION AND FUTURE DIRECTIONS

4.1 Conclusion 36

4.2 Future Directions 36

APPENDIX A: SOURCE CODE MERCURYSERVER.JAVA 38

APPENDIX B: SOURCE CODE MERCURYCLIENT.JAVA 51

REFERENCES 79

LIST OF TABLES

Table 1. Mercury Instant Messaging System User Functions 4

Table 2. Unit Test Results (Interfaces) 32

Table 3. Subsystem Test Results 34

Table 4. System Test Results 35

LIST OF FIGURES

Figure 1. Mercury Instant Messenger Use Case Diagram 3

Figure 2. Mercury Instant Messaging System Deployment Diagram 7

Figure 3. Mercury Server Interface 8

Figure 4. New User Registration Interface 9

Figure 5. Chat as Administrator Interface 10

Figure 6. Administrator Toolbar Interface 11

Figure 7. User Client Interface (a) 15

Figure 8. User Client Interface (b) 16

Figure 9. User Client Interface (c) 17

Figure 10. User Client Interface (d) 18

Figure 11. User Client Interface (e) 19

Figure 12. Chatrooms Interface 20

Figure 13. Chatrooms Toolbar Interface 21

Figure 14. Private Messaging Interface 22

Figure 15. Login/Settings Interface 23

Figure 16. Mercury Instant Messaging System Flat Files Structure 25

Figure 17. Mercury Server Class Diagram 27

Figure 18. Mercury Client Class Diagram 29

CHAPTER ONE

INTRODUCTION

1.1 Purpose of the Project

The purpose of the project is to use Java technology to create an instant messenger application that could be used by any person who has the basic knowledge of working with a graphical user interface. There are many operating system specific instant messenger applications available today. But the goal here is to develop an application that provides communication to users running different operating systems. Mercury Instant Messenger (MIM) gives users the same functionalities and look and feel on any platform.

MIM is a client-server system where the application can be downloaded from a website and account creation/sign-up is done by the administrator. The administrator will be able to create a profile for the user and set his/her preferences to use the application.

1.2 Scope of the Project

MIM has the following functionality for its users:

- Cross platform compatibility, so that users running different operating systems will get to use the program.

- The instant messaging server handles new user registration and current user logins.
- Online notification to current users about the status of friends and similarly online notification of current user's status to members of the buddy list.
- Keep alive signal processing by the instant messaging server to check client-server connection.
- An attractive and clean interface implemented using GUI.
- Buddy list/friends list.
- Ability to send messages to users who are currently offline.
- Use of sounds to "Buzz" users.
- Ability to create, enter and manage both public and private chat rooms.
- Collaborative whiteboard that allows users to draw in various formats.

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

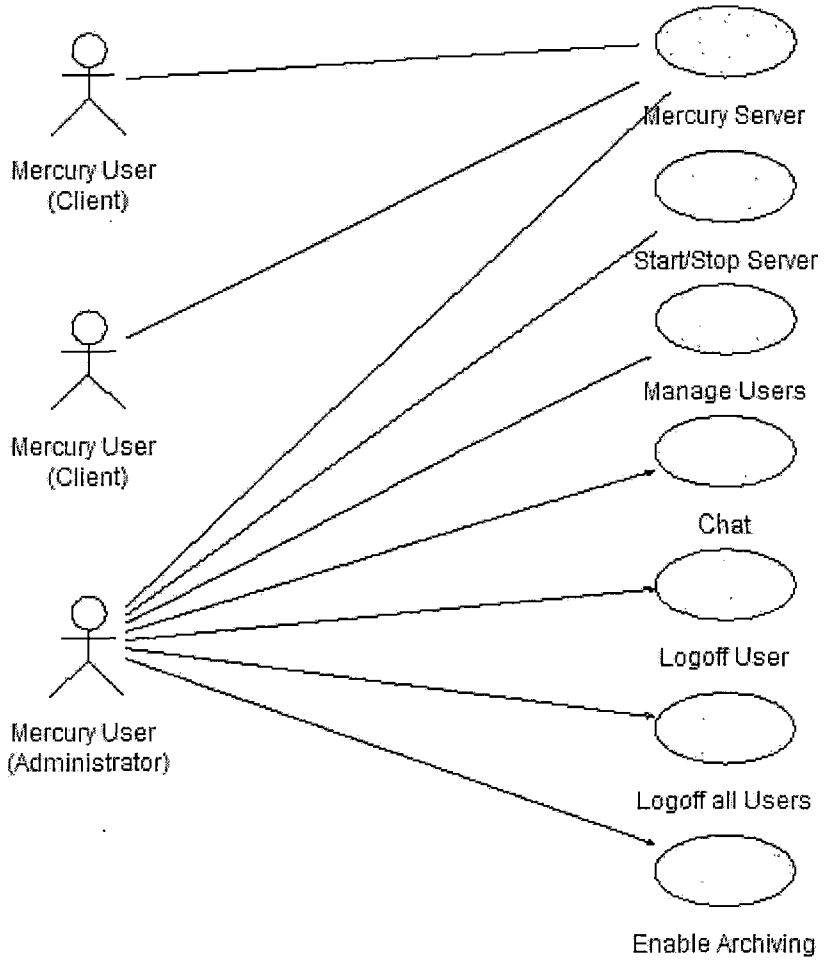

Figure 1. Mercury Instant Messenger Use Case Diagram

The various functions available to the MIMS users are listed in the table below:

Table 1. Mercury Instant Messaging System User Functions

Objects	Functions Available
Chatrooms	Create, ignore, allow, ban, view room information
Group Chat	Send, receive, broadcast, chat with a particular user, view profile, ignore, buzz user(s), save conversation
Whiteboard	Send, receive, save, select drawing tools, insert image, save whiteboard
Private Messaging	Send messages, receive messages, leave offline messages, check offline messages

1.3 Limitations of the Project

Since both the server and client are written in Java on the machines in which they are executed java virtual machine should be installed. For peer-to-peer chatting also server should be active but the messages are sent directly to the other client. New users can be registered by the administrator only.

1.4 Definitions, Acronyms, and Abbreviations

The definitions, acronyms, and abbreviations used in the document are described below:

MIMS - Mercury Instant Messaging System, a client-server program written in Java, that enables users on the WWW/LAN, to send instant messages in real-time to multiple users online.

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.

External User Interface - This refers to the external look and feel of the frames, panels, buttons, and other GUI that allows the user to interact with the software package.

Java - An object oriented language developed by Sun Microsystems. Java programs are capable of running on most popular computer platforms without the need for recompilation.

API - Application Programming Interface.

UML - Unified Modeling Language used for Object Oriented design.

JVM - Java Virtual Machine which provides a virtual environment for Java programs to execute.

Current User - User who runs the client and logs onto MIMS.

Offline Users - Users of the MIMS who are not logged onto the system.

Online Users - Users of the MIMS who are logged onto the system.

IP Address - Internet Protocol Address, a unique ID that identifies a computer on the internet.

1.5 Overview

The remainder of this MIMS SRS document defines the function and specific requirements of MIMS in a format consistent with the IEEE Std 830-1993 SRS format [1] and IEEE Std 830-1998 SRS format [2].

1.6 Overall Description

1.6.1 Product Perspective

The MIMS is a client-server based instant messaging system. This system serves users on the Internet and on Intranets. People can logon to the system from homes, offices, schools, universities, etc. MIMS is a GUI program with the interface design done using Java AWT technology. Users can send instant messages in real-time to other users logged onto the system. The system also provides users with features like offline messaging, collaborative whiteboard and chat rooms.

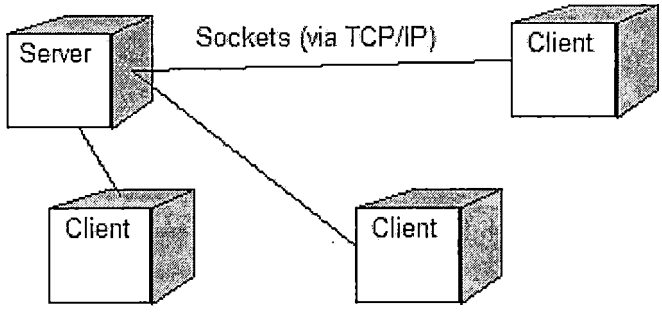

Figure 2. Mercury Instant Messaging System Deployment Diagram

1.6.2 System Interfaces

The server and client of MIMS is implemented using socket programming. The application is written using Java 1.3, therefore, the JVM 1.3 at least is necessary to run it. The GUI for the client application is written using AWT, therefore, the classes in the package would be needed to compile and run the client application.

1.6.3 Server Interface

This is the interface used by the server administrator when he runs the server application. The options available in this interface are; the port number on which the server is running, enable/disable chat archiving, display showing which users are connected to the server, buttons with options for "Manage users", "Chat as administrator", "Logoff a user", "Logoff all users",

and "Server Shutdown", displays showing server traffic, activity information and the mercury logo.

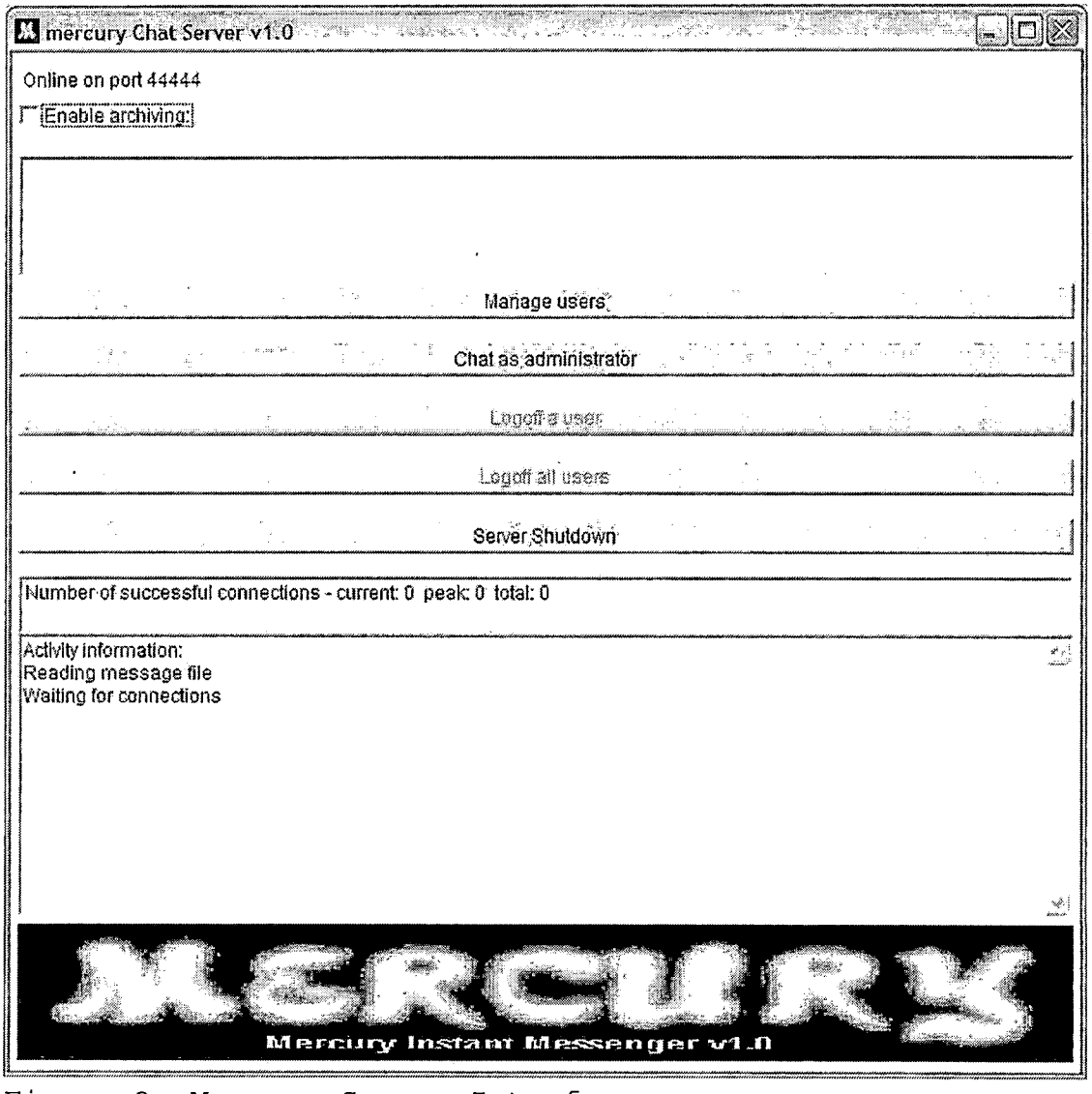

Figure 3. Mercury Server Interface

1.6.4 New User Registration Interface

This interface is the manage users/user tool screen. The administrator would have to enter the user's desired username, password, and additional optional information to

register and open a new account on the system. Then the user presses the "Finished" button to submit the data to the system. This interface provides registration data to the program to be stored in a file. As the client contacts the server it sends this data to the server at the time of user validation.

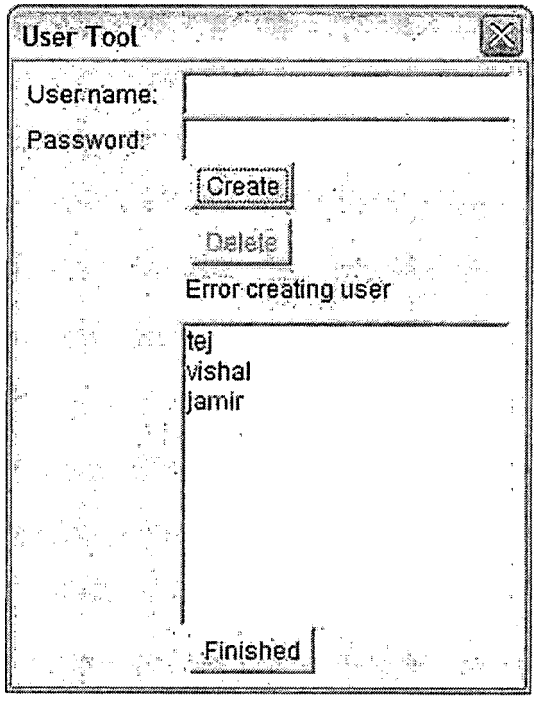

Figure 4. New User Registration Interface

1.6.5 Chat as Administrator Interface

To get to this interface the administrator must click on the chat as administrator button on the server interface. This interface is more or less similar to the normal user client interface with the exception that the chat toolbar always pops-up. This enables the

administrator to see which users are logged in into the various chat rooms and he can then choose to boot, invite, ban, or allow users in whatever he pleases.

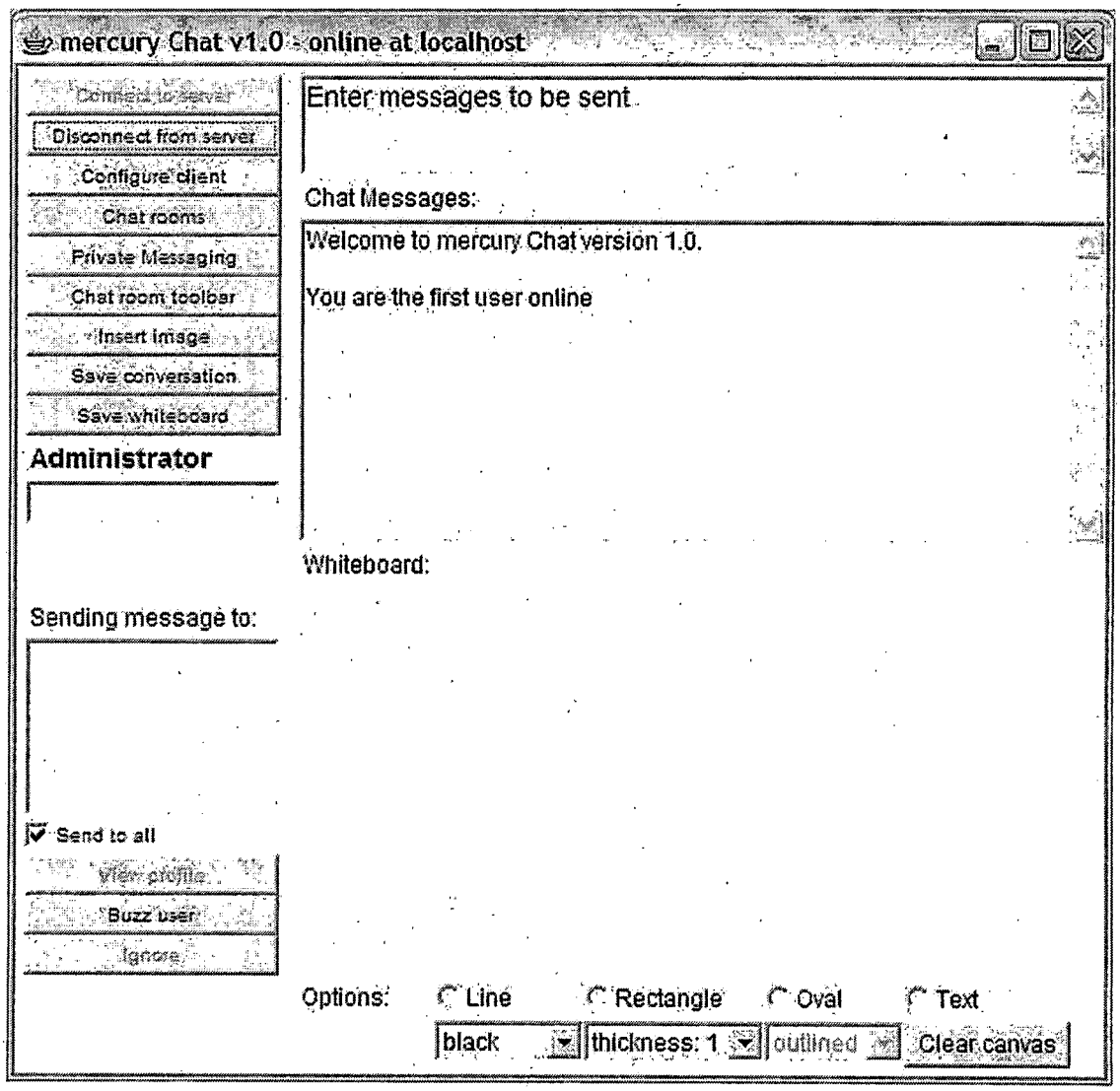

Figure 5. Chat as Administrator Interface

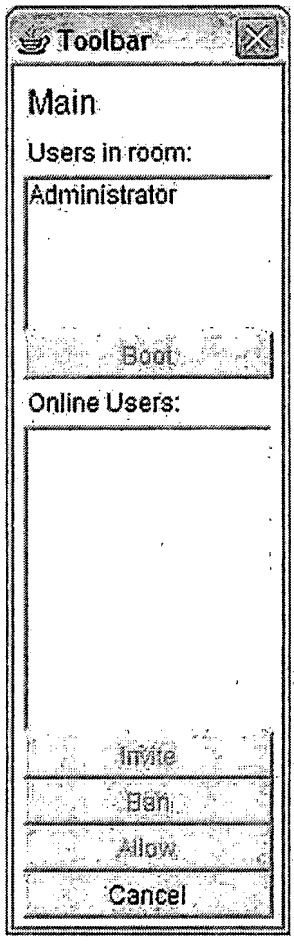

Figure 6. Administrator Toolbar Interface

1.6.6 User Client Interface

This is the interface that the user gets when he runs the chat client. The interface contains the following:

- Connect to server button – This pops-up a window where the user can enter his username, password, IP/domain name of the server, port number the server is running on and optional additional information.

- Disconnect from server button - By pressing this button the user can disconnect from the server.
- Configure client button - Collects same information as the connect to server button but only stores this information and does not connect to the server when user presses the ok button.
- Chat rooms button - By pressing this button the user can see a list of existing chat rooms and can also create/manage chat rooms.
- Private messaging button - By pressing this button a user can send private messages to other users and can also read saved offline messages.
- Chatroom toolbar button - By pressing this button the user can see the chatroom toolbar.
- Insert image button - By pressing this button the user can select an image file and paste it onto the whiteboard.
- Save conversation button - By pressing this button the user can save all chat conversations to a text file for future reference.

- Save whiteboard button - By pressing this button the user can save the whiteboard drawings to an image file on his computer for future reference.
- User activity monitor - This display tells the user what part of the chat he is using.
- User list display - This display shows the user which other users are logged into the chat.
- Send to all checkbox - This checkbox when enabled sends messages to all online users in the broadcast mode.
- View profile button - On clicking this button the user can view the profile of a user.
- Buzz user button - On clicking this button the user can buzz another online user with a sound.
- Ignore button - The user can select any online user from the user list and choose to ignore him by pressing this button.
- Text messaging box - Here the user types his conference messages.
- Chat messages box - Here the user can see his typed messages and the messages that the other users are sending him.

- Whiteboard and whiteboard options - The user can draw on the whiteboard and send the image to other online users using the various options. The options available are various shapes, brush sizes, colors, and text. The user can also insert an image from his computer onto the whiteboard.

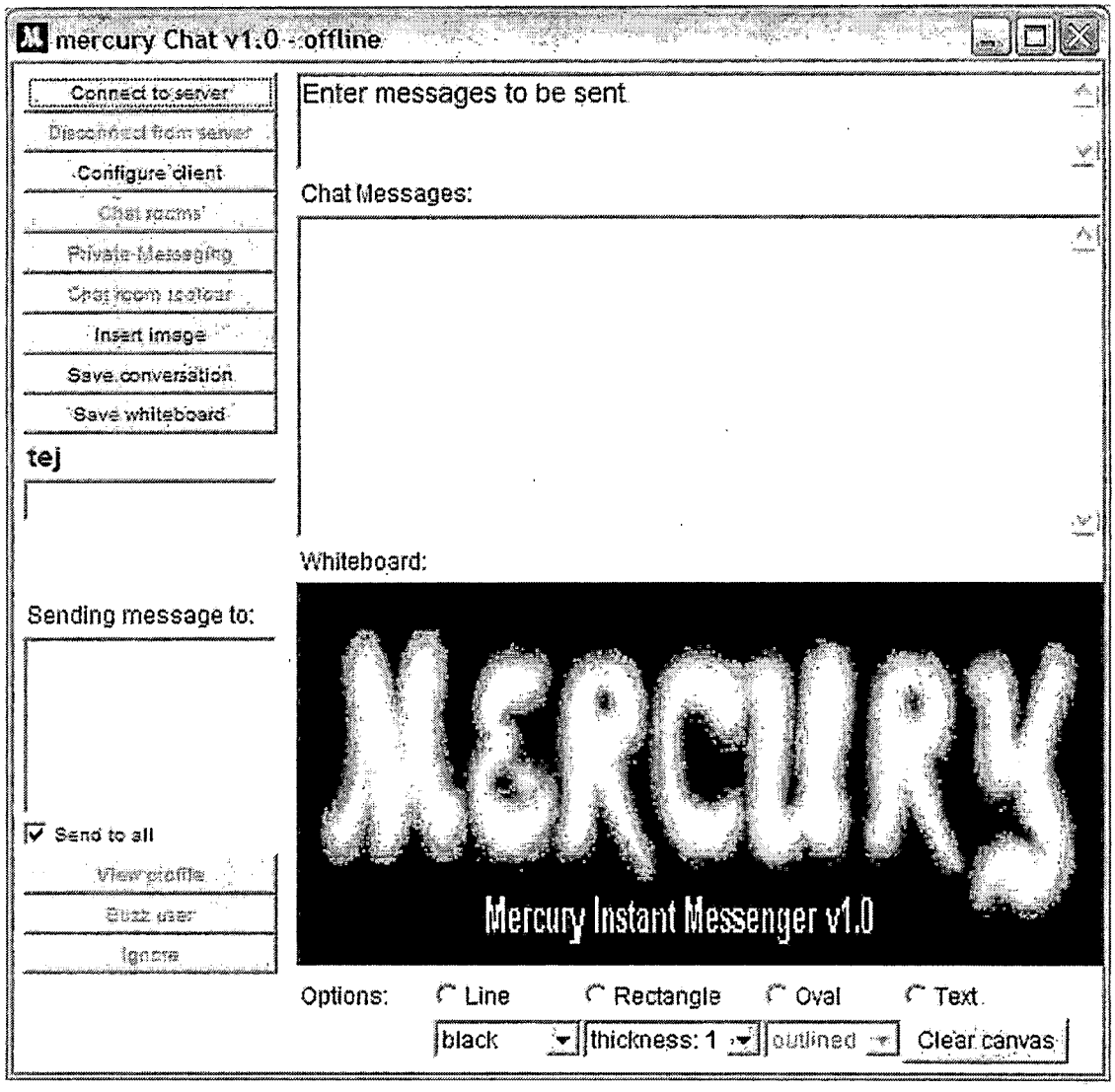

Figure 7. User Client Interface (a)

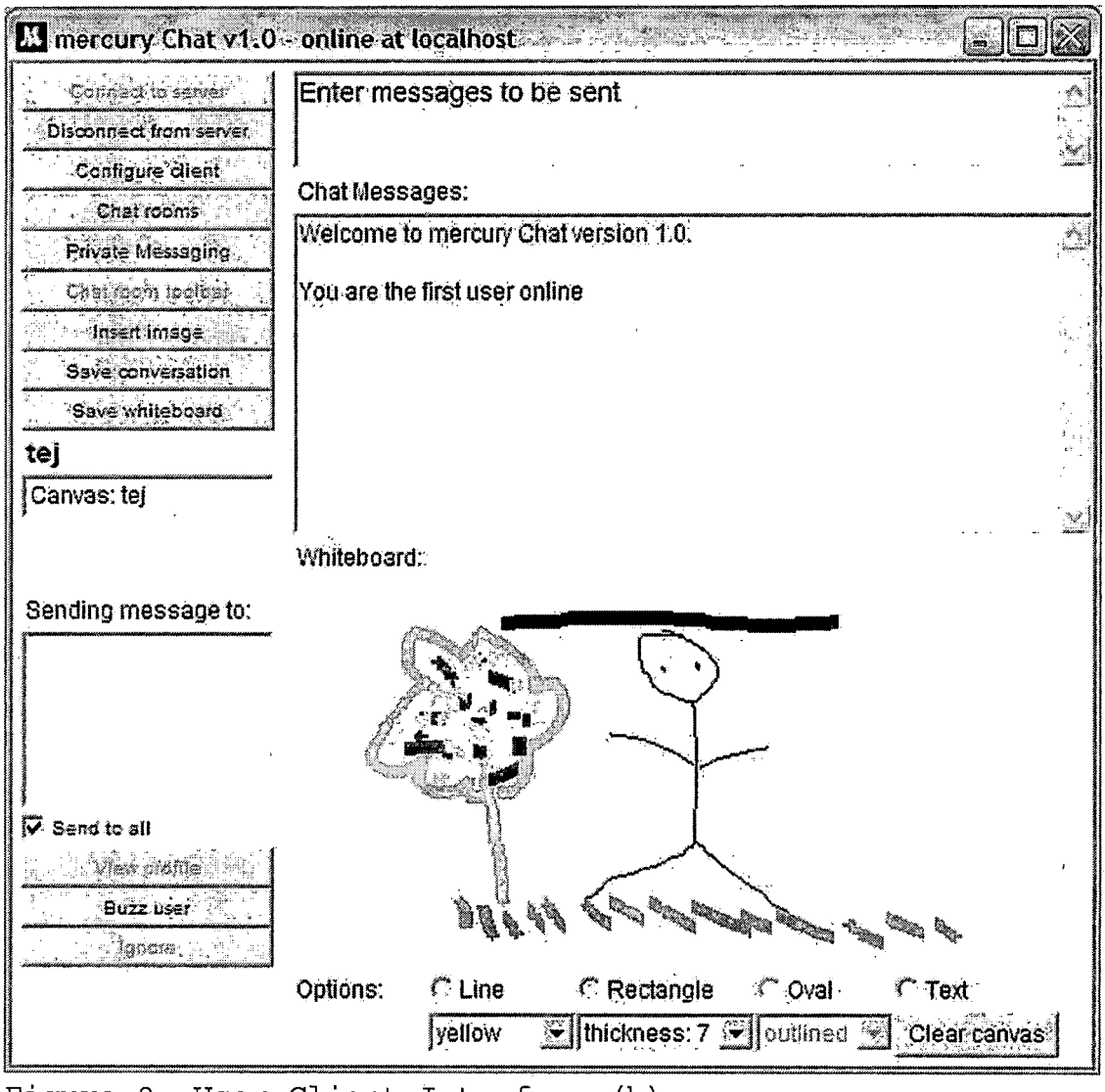

Figure 8. User Client Interface (b)

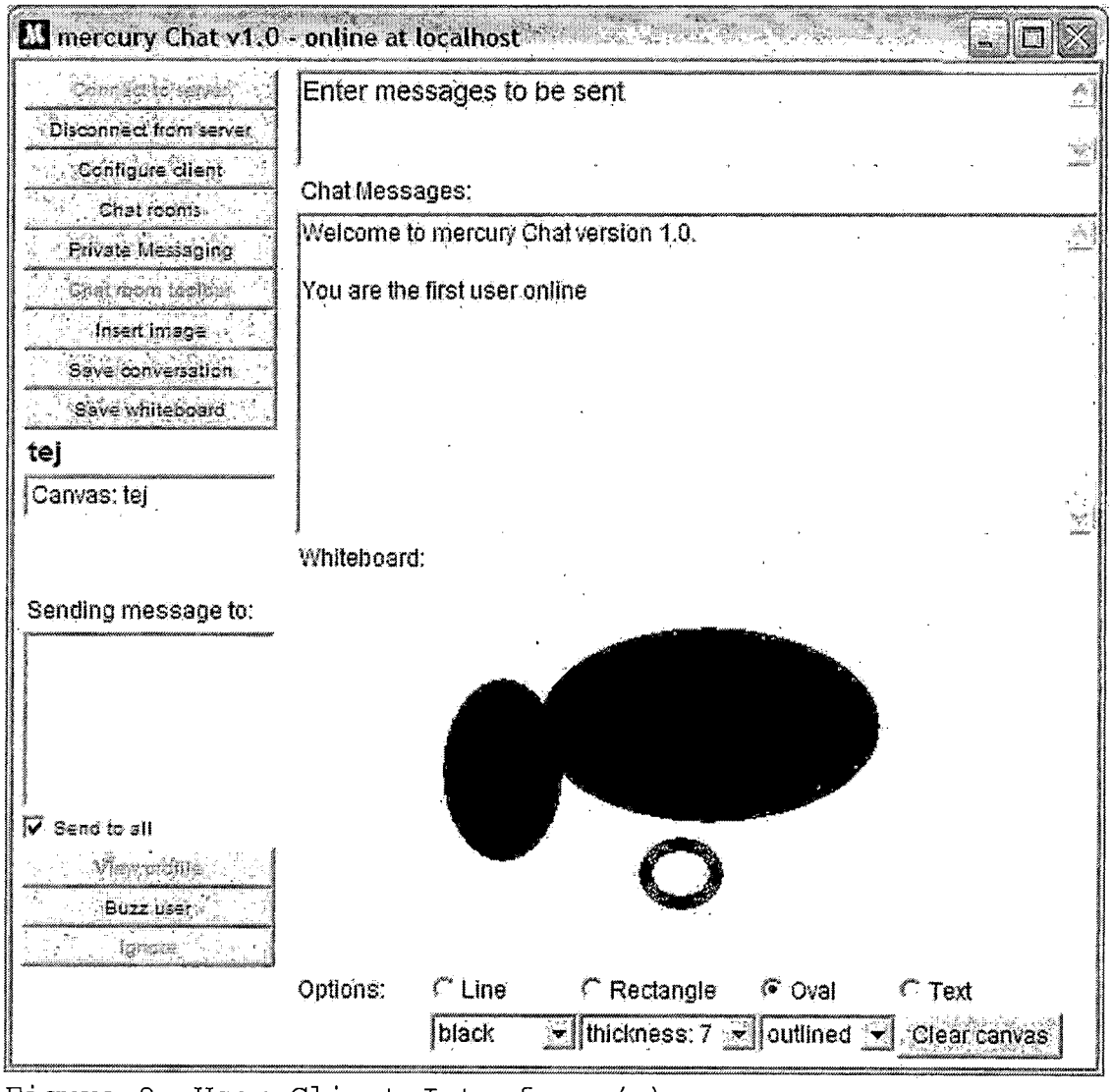

Figure 9. User Client Interface (c)

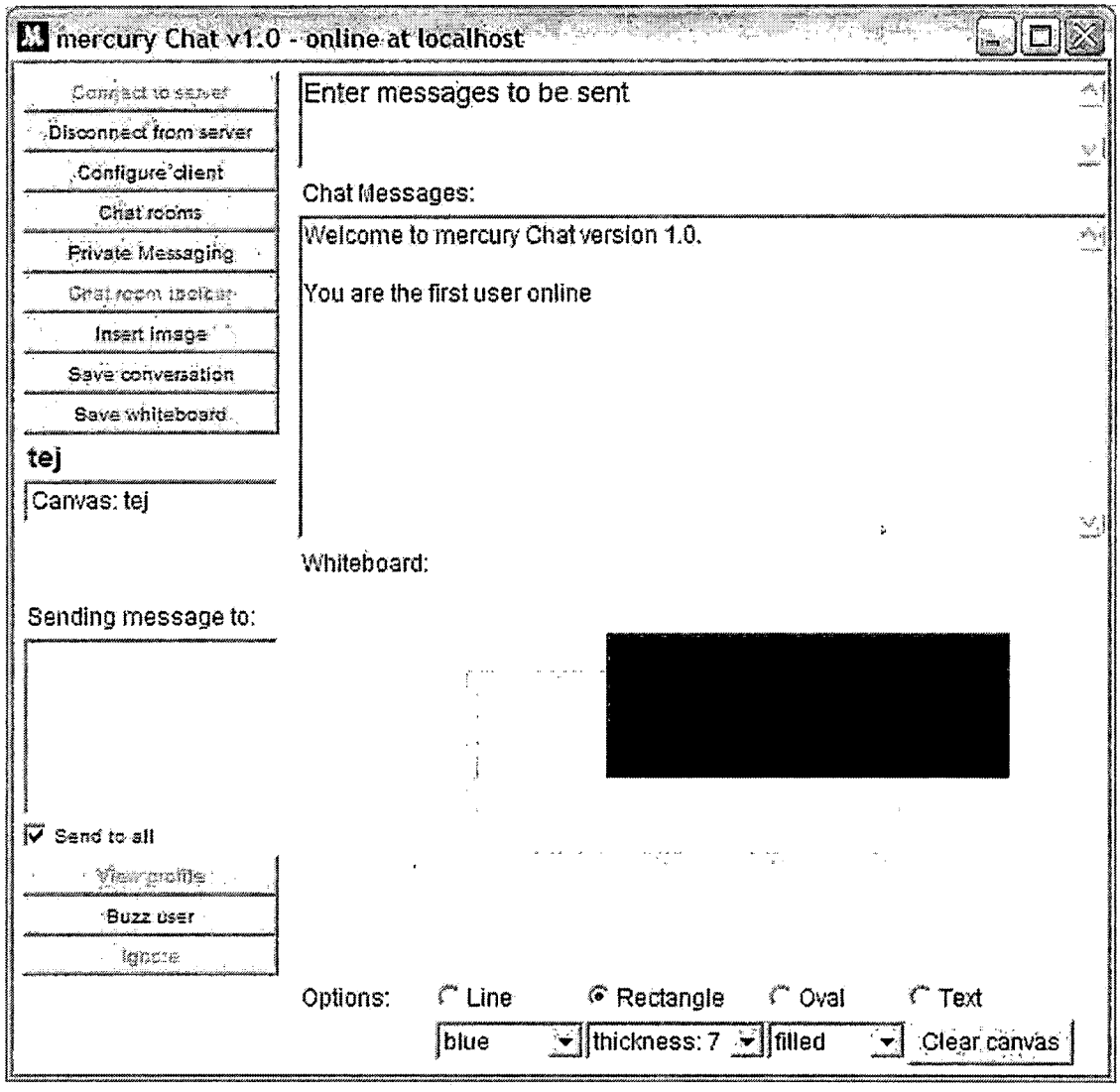

Figure 10. User Client Interface (d)

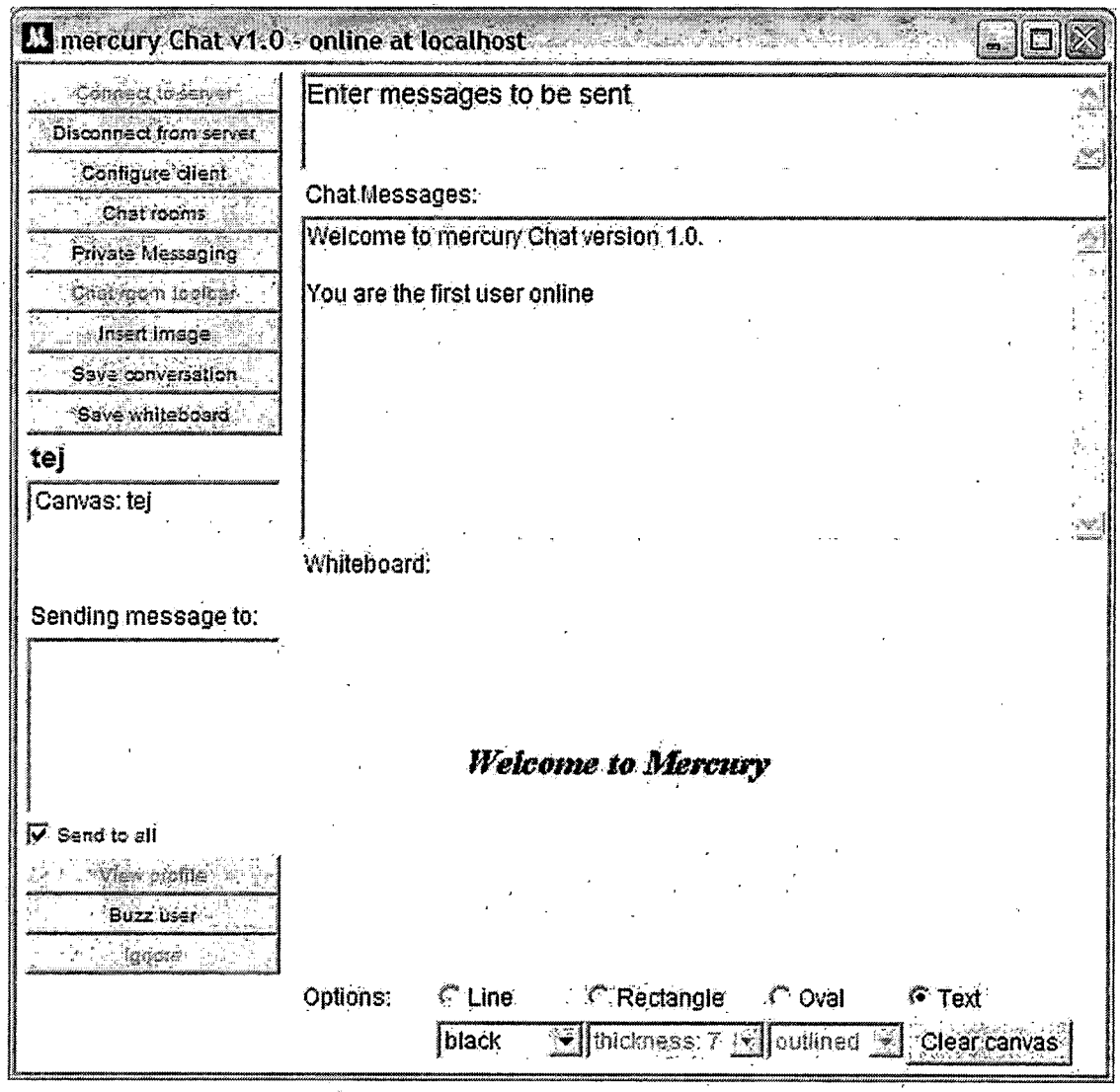

Figure 11. User Client Interface (e)

1.6.7 Chatrooms Interface

In this interface the user can see which chat rooms are available, enter chat rooms, view chat room information, create and update chat room lists.

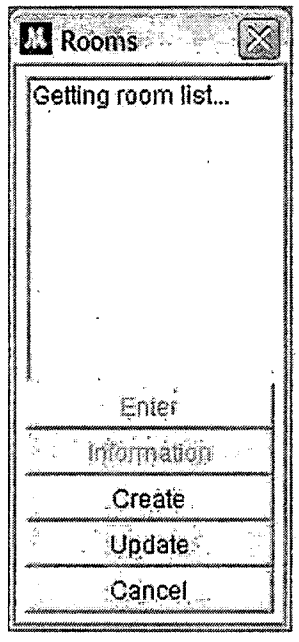

Figure 12. Chatrooms Interface

1.6.8 Chatrooms Toolbar Interface

This toolbar pops-up when the user enters a chat room other than the main chat room. This is a floating toolbar and allows the user to see which users are in the chat room, which users are online, ability to boot users, and the ability to invite/ban/allow users into the chat room.

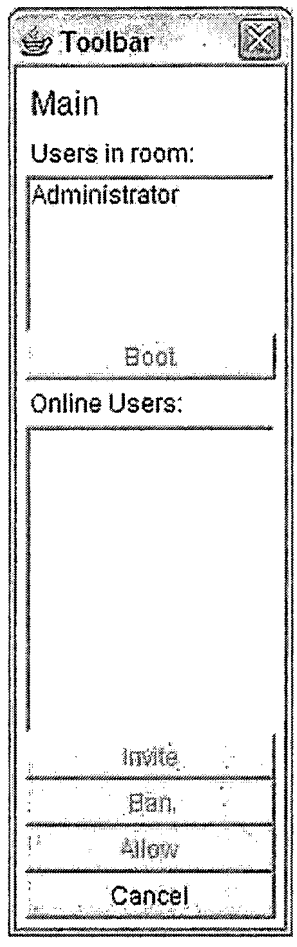

Figure 13. Chatrooms Toolbar Interface

1.6.9 Private Messaging Interface

This interface allows users to send private instant messages to online users, send offline messages to users, and the ability to check saved offline messages sent by other users.

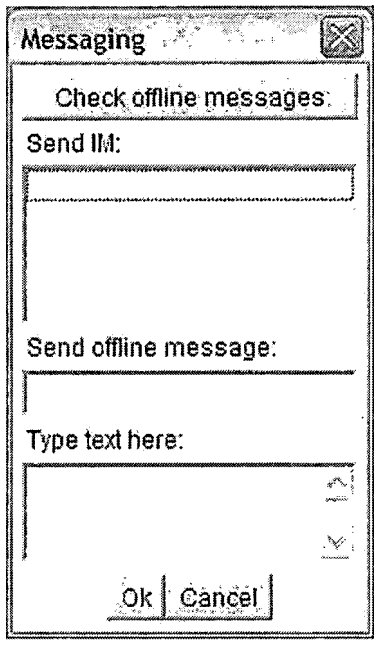

Figure 14. Private Messaging Interface

1.6.10 Login/Settings Interface

This interface enables the user to log into the system. The user enters his username, password, IP address/domain name of the server and the server port number and presses on OK to submit the data to the system.

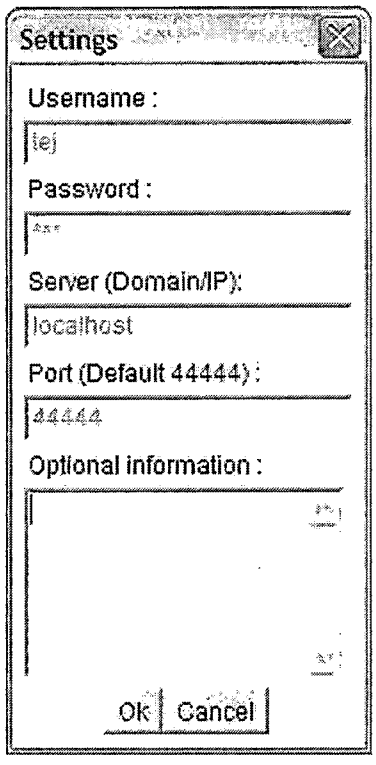

Figure 15. Login/Settings Interface

1.7 Hardware Interfaces

MIMS will not directly implement any hardware interfaces. All interfaces to I/O devices will be provided by the operating system.

1.8 Software Interfaces

Since the MIMS client/server application needs to be hosted on the internet, a web server is required to serve the application, JDK 1.3 for software development, and a web browser to access the downloads page.

The following browsers/platforms are supported at the minimum:

- Operating Systems Supported:
  - Windows 98/NT.
  - LINUX 5.2+.
  - Solaris
- Browsers:
  - Netscape 4.0+
  - Internet Explorer 4.0+
- Servers:
  - Any web server.

1.9 Communications Interfaces

Socket programming and JDK "Net" package handles all required communications.

1.10 User Characteristics

The primary users of MIMS are home, office, school and university users who would be looking for a cost-effective, efficient tool to send messages and have real-time conversations. The users are expected to have the ability to work with a GUI/Multimedia based application.

1.11 Logical Database Requirements

Mercury server is not a database-driven application. It stores all the data pertaining to the user in flat

files. It also contains various other information like the friends list of a user and the status of the users. No database is required at the client side. All the information is obtained by a request to the server. The flat files being used are:

User.passwords - This file contains the username and password strings of the user.

Messages.saved - This file contains saved messages in java UTF format.

<chatroom name>.log - This file contains archived chatroom messages.

The following figure shows the data structure of the flat files being used for MIMS:

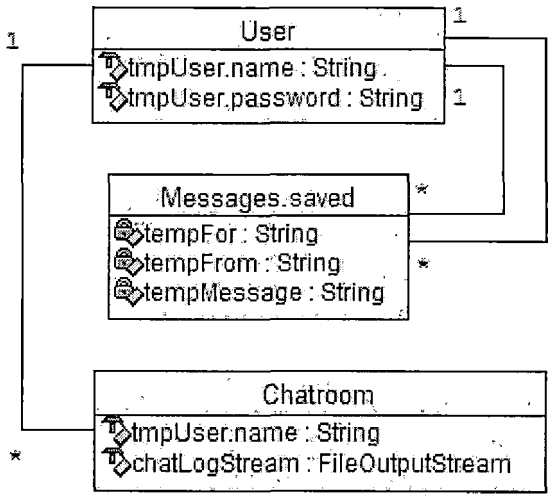

Figure 16. Mercury Instant Messaging System Flat Files Structure

CHAPTER TWO

SOFTWARE DESIGN

2.1 Overview

In this chapter MIMS Methodology, MIMS Architecture, and MIMS Detailed Design are explained.

2.2 Mercury Instant Messaging System Methodology

MIMS is developed as a Java application to make use of its cross-platform compatibility advantages. This makes MIMS easily available to all users.

The framework is based on client (remote clients)/server (central server) model. The remote clients would send requests to the central server and the central server would process the requests and respond. The central server and the clients use sockets over TCP/IP to communicate with each other.

2.2.1 Central Server

The central server has an application server, user details and all other required data. The following is the class diagram for the Mercury Server class:

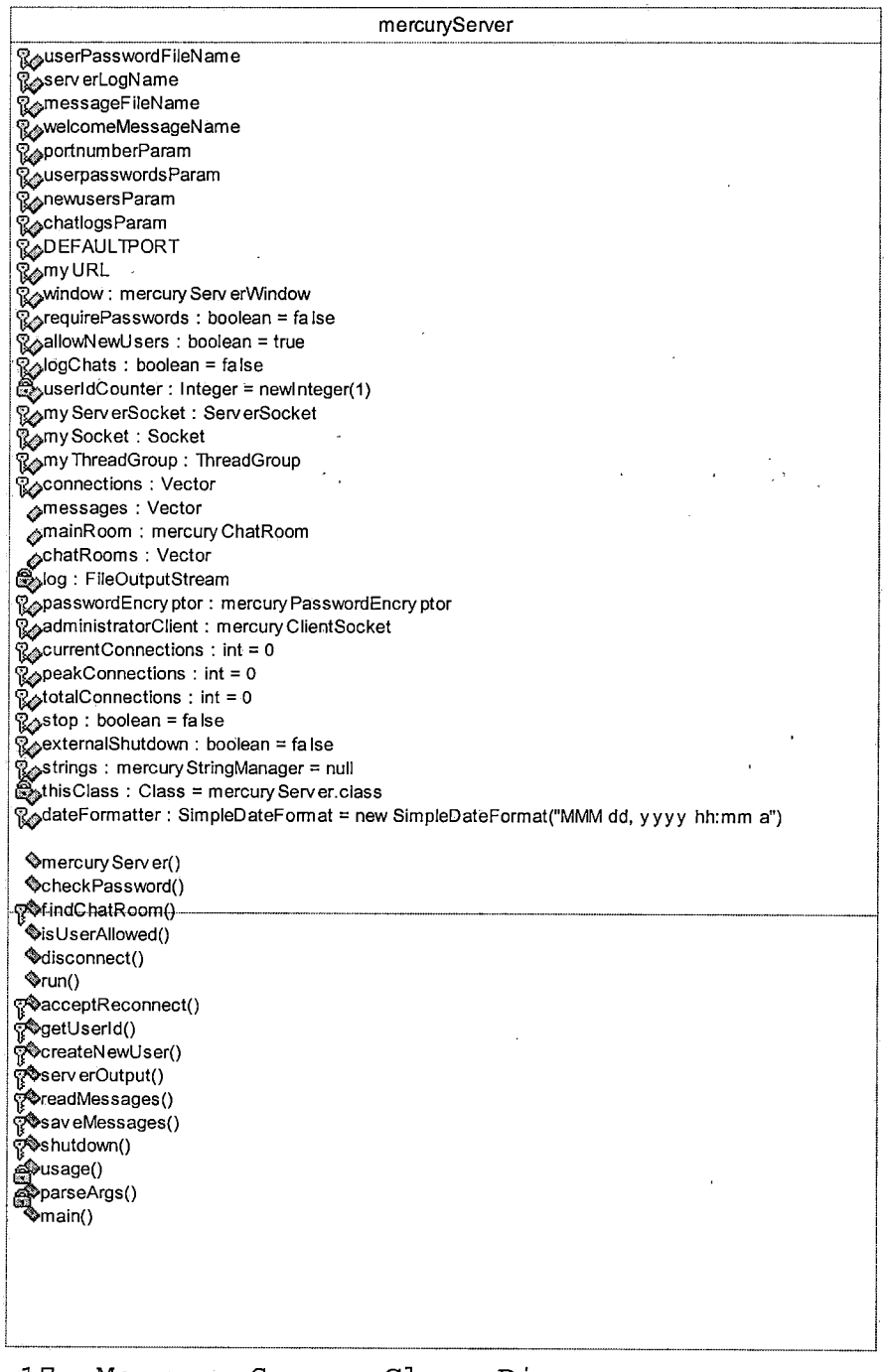

Figure 17. Mercury Server Class Diagram

2.2.2 Remote Client

The client could be run from any computer, which has a JVM installed. The following is the class diagram for the mercury client class:

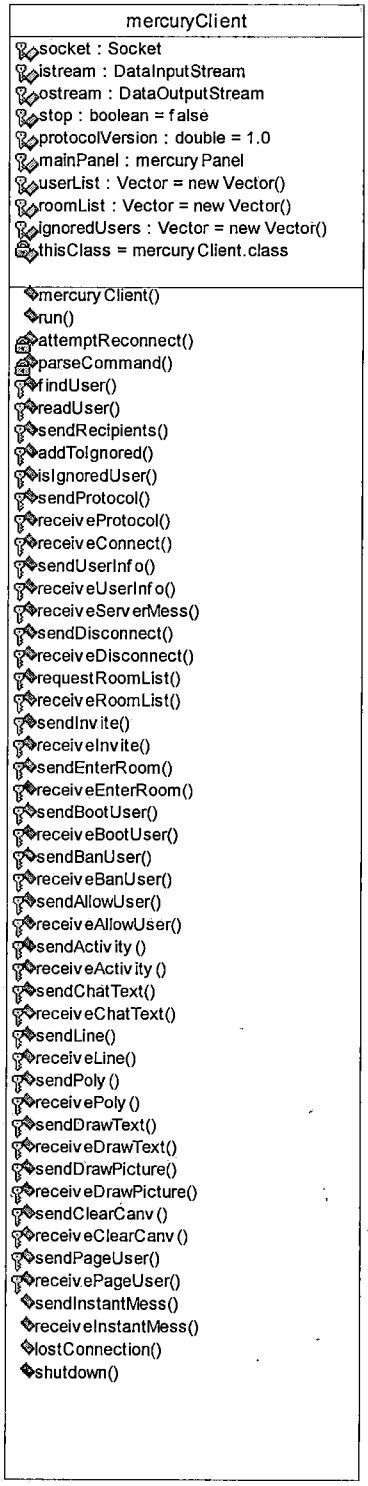

Figure 18. Mercury Client Class Diagram

2.3 Mercury Instant Messaging System Architecture

MIM is implemented as a client-server architecture. Server runs on port 44444 (by default) waiting for the clients to connect and it also maintains a continuous connection with the user data files which contain all the information regarding the client (profile, password, etc.). After a successful connection to the server the client can send messages. Registration of the users is done by the administrator using tools available in the server application. A web server is needed to run to serve the page from which the client can be downloaded. All image processing in the application is done by converting bitmap images to integer values and vice-versa. All these integers are sent between clients and server through sockets as vectors.

CHAPTER THREE

SYSTEM VALIDATION

The system validation test is a kind of test process that ensures that the software program meets the expectation of the user. The purpose of system validation is to provide the highest degree of assurance that a specific process will produce the same result consistently and meet predetermined specifications and quality attributes. This can also guarantee the system performance and reliability. This validation process helps in eliminating lot of bugs from the code base and tries to make the software bug free. Meeting security standards is not possible without having a validation process and protecting sensitive user data from malicious users is the primary goal of testing.

3.1 Unit Test

Unit test is the basic level of testing where individual components are tested to ensure that they operate correctly. These individual components can be object, class, program, etc. The unit testing results of MIMS are shown in Table 1.

Table 2. Unit Test Results (Interfaces)

Interfaces	Tests Performed	Results
Mercury Chat Server Interface	• Verify archiving function is working properly. • Check all the buttons work properly. • Verify all server statistics displays are working properly. • Check server load bearing capabilites.	Pass
Manage Users Interface	• Check all the forms are shown properly to the user. • Check all buttons are working properly. • Check password field masks the entered password.	Pass
Chat as Administrator Interface	• Verify that appropriate interface is loaded depending on the role of the user. • Check user already connected status before loading interface. • Display an error message if two administrators are logged in. • Verify that the chat toolbar interface pops-up.	Pass
User Client Interface	• Check all interfaces are displayed correctly. • Check all the buttons work properly.	Pass
Settings Interface	• Check all the buttons work properly. • Verify the password field masks the password. • Verify all error messages are working.	Pass
Chatrooms Interface	• Verify all displays are working properly. • Check all buttons are working properly.	Pass

Interfaces	Tests Performed	Results
Private Messaging Interface	• Verify all displays are working properly. • Check all buttons are working properly. • Check whether saved messages can be sent and read.	Pass
Chatroom Toolbar Interface	• Check all the buttons work properly. • Verify the page gets the correct user account information. • Check all displays work correctly.	Pass
Activity Monitor Interface	• Verify the correct activity is being shown in the display.	Pass
Whiteboard	• Confirm that all the brushes and drawing tools are working properly. • Verify clear canvas button is working.	Pass

3.2 Subsystem Testing

Subsystem testing is the next step up in the testing process where all related units from a subsystem do a certain task. Thus, the subsystem test process is useful for detecting interface errors from a front-end perspective and specific functions from the back end point of view. Table 9 shows subsystem test results in detail.

Table 3. Subsystem Test Results

Subsystem	Tests Performed	Results
Authorize subsystem	• Test if error message is displayed on incorrect login. • Verify the login user information is stored in the chat session properly.	Pass
Accounts management subsystem	• Make sure all the existing users are listed in the user list. • Check if the subsystem can detect the error of creating of the user that already exists in the subsystem. • Verify the newly created user information is the same as the information provided.	Pass

3.3 System Testing

System testing is the testing process that uses real data, which the system is intended to manipulate, to test the system. First all subsystem will be integrated into one system. Then test the system by using a variety of data to see the overall result. This testing is usually done when the system is about to be deployed in a production environment. The deployment is a mere replication of the tested system into production but tests on production environment have to be done to make sure everything is running.

System testing of MIMS system begins with the following steps:

Table 4. System Test Results

System Testing	Results
1. Install MIMS system into OS.	Pass
2. Start up MIMS server.	Pass
3. Run tests by using real data on all interfaces.	Pass

CHAPTER FOUR

CONCLUSION AND FUTURE DIRECTIONS

4.1 Conclusion

Motivation to collaborate users with common interest but located in different geographical locations, using various conference tools resulted in the development of MIMS. Instant messaging over the network for interactive communication between users has increased widely. MIMS incorporates both text based instant messaging and a collaborative whiteboard.

MIMS is developed as a Java application. MIMS 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. MIMS uses TCP/IP for text messaging for text and whiteboard conferencing over the network. MIMS uses Java AWT and socket programming to implement communication.

4.2 Future Directions

MIMS has a good scope for future enhancements. Functionality of MIMS application can be extended by

providing support to allow the user to participate in voice chat, webcam video chat, and encrypted file transfer. Functionality of MIMS application whiteboard can be improved by allowing return to the free form drawing tool, and providing a restore to saved whiteboards option whereby users can choose to restore to a previous whiteboard from a directory of saved whiteboards.

APPENDIX A

SOURCE CODE MERCURYSERVER.JAVA

```
//
// mercury Chat
// mercuryserver.java
//

import java.net.*;
import java.util.*;
import java.text.*;
import java.io.*;


class mercuryServerShutdown
   extends Thread
{

   private mercuryServer server;

   public mercuryServerShutdown(mercuryServer s)
   {
     server = s;
   }

   public void run()
   {
     if (server.stop)
        return;

     server.externalShutdown = true;
     server.shutdown();
   }
}

public class mercuryServer
   extends Thread
{
   protected static String userPasswordFileName = "User.passwords";
   protected static String serverLogName = "Server.log";
   protected static String messageFileName = "Messages.saved";

   protected static int DEFAULTPORT = 44444;

   protected URL myURL;
   protected int port = DEFAULTPORT;
   protected mercuryServerWindow window;
   protected boolean graphics = true;
   protected boolean requirePasswords = false;
   protected boolean allowNewUsers = true;
   protected boolean logChats = false;
   private Integer userIdCounter = new Integer(1);

   protected ServerSocket myServerSocket;
   protected Socket mySocket;
   protected ThreadGroup myThreadGroup;

   protected Vector connections;
   public Vector messages;
   public mercuryChatRoom mainRoom;
   public Vector chatRooms;

   private FileOutputStream log;
```

```
protected mercuryPasswordEncryptor passwordEncryptor;

protected mercuryClientSocket administratorClient;

protected int currentConnections = 0;
protected int peakConnections = 0;
protected int totalConnections = 0;

protected boolean stop = false;
protected boolean externalShutdown = false;

protected mercuryStringManager strings = null;
private Class thisClass = mercuryServer.class;

protected SimpleDateFormat dateFormatter =
  new SimpleDateFormat("MMM dd, yyyy hh:mm a");


public mercuryServer(String[] args)
{
  super("mercury Chat server");

  try {
     myURL = new URL("file", "localhost", "./");
  }
  catch (Exception E) {
     System.out.println(E);
     System.exit(1);
  }

  strings = new mercuryStringManager(myURL,
                        Locale.getDefault().getLanguage());

  if (!parseArgs(args))
     System.exit(1);

  try {
     File logFile = new File(serverLogName);
     log = new FileOutputStream(logFile);
  }
  catch (IOException e) {
     serverOutput(strings.get(thisClass, "noopen") + " " +
           serverLogName + "\n");
  }


  if (graphics)
     {
       window =
          new mercuryServerWindow(this,
                        strings.get(thisClass,
                              "mercuryversion") +
                        mercury.VERSION);
       window.setSize(700, 700);
       window.setVisible(true);
     }

  else
     {
       System.out.println("\n" + strings.get(thisClass,
```

```
                                    "serverstatus"));
            System.out.println(strings.get(thisClass, "listenport") +
                               " " + port);
            System.out.println(strings.get(thisClass, "connections"));
        }

    try {
        myServerSocket = new ServerSocket(port);
    }
    catch (IOException e) {
        System.out.println("\n" + strings.get(thisClass,
                                    "noserversocket"));
        System.exit(1);
    }

    connections = new Vector();
    messages = new Vector();
    chatRooms = new Vector();
    myThreadGroup = new ThreadGroup("Clients");

    passwordEncryptor = new mercuryPasswordEncryptor();

    if (requirePasswords)
        try {
            new FileOutputStream(userPasswordFileName, true).close();
        }
        catch (IOException f) {}

    mainRoom = new mercuryChatRoom("Main", "Administrator", false, null);
    try {
        mainRoom.setLogging(logChats);
    }
    catch (IOException e) {
        serverOutput(strings.get(thisClass, "noroomlog") + " " +
                mainRoom.name + "\n");
    }

    serverOutput(strings.get(thisClass, "readingmessages") + "\n");
    readMessages();

    serverOutput(strings.get(thisClass, "waiting") + "\n");


    double javaVersion = 0.0;
    try {
        javaVersion = new Double(System.getProperty("java.version")
                          .substring(0, 3)).doubleValue();
    }
    catch (NumberFormatException e) {}

    if (javaVersion >= 1.3)
        Runtime.getRuntime()
            .addShutdownHook(new mercuryServerShutdown(this));

    start();
}

public boolean checkPassword(String fileName, String userName,
                             String password)
    throws Exception
```

```
{
  DataInputStream passwordStream = null;

  try {

    passwordStream =
      new DataInputStream(new FileInputStream(fileName));

    while(true)
      {
        String tempUserName = "";
        String tempPassword = "";
        try {
          tempUserName = passwordStream.readUTF();
          tempPassword = passwordStream.readUTF();
        }
        catch (EOFException e) {
          throw new Exception(strings.get(thisClass,
                                "nosuchuser"));
        }

        if (tempUserName.equals(userName))
          {
            if (tempPassword.equals(password))
              return (true);
            else
              return (false);
          }
      }
  }
  catch (IOException e) {
    serverOutput(strings.get(thisClass, "errorpasswordfile") +
          " " + e.toString() + "\n");
    return (false);
  }
}

protected mercuryChatRoom findChatRoom(String roomName)
{

  mercuryChatRoom tempRoom = null;
  mercuryChatRoom returnRoom = null;

  if (mainRoom.name.equals(roomName))
    returnRoom = mainRoom;

  else
    {
      for (int count = 0; count < chatRooms.size(); count ++)
        {
          tempRoom = (mercuryChatRoom)
            chatRooms.elementAt(count);

          if (tempRoom.name.equals(roomName))
            {
              returnRoom = tempRoom;
              break;
            }
        }
```

```
    }

  return (returnRoom);
}

public boolean isUserAllowed(mercuryChatRoom room,
                mercuryClientSocket client, String password)
{
  if (room.priv)
    {
      boolean invited = false;


      if (!room.password.equals(password))
        {

          for (int count2 = 0;
              count2 < room.invitedUsers.size(); count2 ++)
            {
              Integer Id = (Integer) room.invitedUsers
                .elementAt(count2);

              if (Id.intValue() == client.user.id)
                {
                  invited = true;
                  break;
                }
            }

          if (!invited)
            {
              try {
                client.sendServerMessage(
                 client.strings.get(thisClass,
                            "notallowed") +
                 " " + room.name);
              }
              catch (IOException e) {
                disconnect(client, false);
                return (false);
              }
              return (false);
            }
        }
    }

  for (int count1 = 0; count1 < room.bannedUserNames.size(); count1 ++)
    {
      if (room.bannedUserNames.elementAt(count1)
        .equals(client.user.name))
        {
          try {
            client.sendBanUser(client.user.id, room.name);
          }
          catch (IOException e) {
            disconnect(client, false);
            return (false);
          }
          return (false);
        }
```

```
    }

  return (true);
}

public synchronized void disconnect(mercuryClientSocket who,
                    boolean notify)
{
  int count;
  mercuryChatRoom chatRoom;

  if (notify)
    {
      try {
        who.sendDisconnect(who.user.id, strings
                .get(thisClass, "disconnected"));
      }
      catch (IOException e) {}
    }

  who.shutdown();

  try {
    who.leaveChatRoom();
  }
  catch (IOException e) {}

  synchronized (connections)
    {
      connections.removeElement(who);
      connections.trimToSize();
      currentConnections = connections.size();

      for (count = 0; count < currentConnections; count ++)
        {
          mercuryClientSocket other = (mercuryClientSocket)
            connections.elementAt(count);
          try {
            other.sendDisconnect(who.user.id, "");
          }
          catch (IOException e) {}
        }
    }

  serverOutput(strings.get(thisClass, "user") + " " + who.user.name +
          " " + strings.get(thisClass, "disconnectedat") + " " +
          dateFormatter.format(new Date()) + "\n");

  serverOutput(strings.get(thisClass, "thereare") + " " +
          currentConnections + " " +
          strings.get(thisClass, "usersconnected") + "\n");

  try {
    sleep(250);
  }
  catch (InterruptedException I) {}

  if (graphics)
    {
      synchronized (window.userList) {
```

```
            for (count = 0; count < window.userList.getItemCount();
               count ++)
              {
                 if (window.userList.getItem(
                          count).equals(who.user.name))
                   {
                      window.userList.remove(who.user.name);
                      break;
                   }
              }
        }

        window.updateStats();

        window.disconnectAll.setEnabled(currentConnections > 0);
        synchronized (window.userList)
          {
            if ((currentConnections <= 0) ||
               window.userList.getSelectedItem() == null)
               window.disconnect.setEnabled(false);
          }
      }
    return;
}

public synchronized void disconnectAll(boolean notify)
{
  int count;

  synchronized (connections)
    {
      for(count = (currentConnections - 1); count >= 0; count --)
        {
          mercuryClientSocket temp = (mercuryClientSocket)
             connections.elementAt(count);

          if (temp == null)
             continue;

          disconnect(temp, notify);
        }
    }

  return;
}


public void run()
{
  while (!stop)
    {
      try {
        mySocket = myServerSocket.accept();
      }
      catch (IOException e) {
        serverOutput(strings.get(thisClass, "socketerror") +
                "\n");
        try {
          myServerSocket.close();
```

```
            }
            catch (IOException f) {
              serverOutput(strings.get(thisClass,
                           "closesocketerror") + "\n");
            }
            System.exit(1);
          }

          if (mySocket == null)
            {
              serverOutput(strings.get(thisClass, "nullsocket") +
                     "\n");
              try {
                 myServerSocket.close();
              }
              catch (IOException g) {
                 serverOutput(strings.get(thisClass,
                              "closesocketerror") +
                        "\n");
              }
              System.exit(1);
            }

          boolean reconnection = false;
          String tmpAddr = mySocket.getInetAddress().getHostAddress();
          synchronized (connections)
            {
              for (int count = 0; count < connections.size();
                   count ++)
                {
                  mercuryClientSocket tmpClient =
                     (mercuryClientSocket)
                     connections.elementAt(count);

                  if (!tmpClient.online &&
                     tmpAddr.equals(tmpClient.mySocket
                              .getInetAddress()
                              .getHostAddress()))
                     {
                       acceptReconnect(tmpClient);
                       reconnection = true;
                       break;
                     }
                }
            }

          if (!reconnection)
            new mercuryClientSocket(this, mySocket, myThreadGroup);
        }
    }

    protected void acceptReconnect(mercuryClientSocket client)
    {
      try {
         client.mySocket.close();
      }
      catch (IOException e) {}

      client.mySocket = mySocket;
```

```
    try {
      client.istream =
        new DataInputStream(client.mySocket.getInputStream());
      client.ostream =
        new DataOutputStream(client.mySocket.getOutputStream());
      client.interrupt();
    }
    catch (IOException a) {}
}

protected int getUserId()
{

    int tmp;

    synchronized (userIdCounter)
      {
        tmp = userIdCounter.intValue();
        userIdCounter = new Integer(tmp + 1);
      }

    return (tmp);
}

protected void createNewUser(String userName, String encryptedPassword)
    throws Exception
{
    new mercuryUserTool().createUser(userName, encryptedPassword);
}

protected void serverOutput(String message)
{
    if (graphics)
      window.logWindow.append(message);
    else
      System.out.print(message);

    if (log != null)
      {
        try {
          byte[] messagebytes = message.getBytes();
          log.write(messagebytes);
        }
        catch (IOException F) {
          if (graphics)
            window.logWindow
              .append(strings.get(thisClass, "writelogerror") +
                  "\n");
          else
            System.out.print(strings.get(thisClass,
                          "writelogerror") + "\n");
        }
      }

    return;
}

protected void readMessages()
{
    String tempFor = "";
```

```
  String tempFrom = "";
  String tempMessage = "";

  DataInputStream messageStream = null;

  try {
     messageStream =
       new DataInputStream(new FileInputStream(messageFileName));

     while(true)
       {
         try {
           tempFor = messageStream.readUTF();
           tempFrom = messageStream.readUTF();
           tempMessage = messageStream.readUTF();
         }
         catch (EOFException e) {
           break;
         }

         messages.addElement(new mercuryMessage(tempFor, tempFrom,
                                    tempMessage));
       }

     messageStream.close();
  }
  catch (IOException E) {}

  return;
}

protected void saveMessages()
{
  DataOutputStream messageStream = null;

  try {
     messageStream =
       new DataOutputStream(new FileOutputStream(messageFileName));

     for (int count = 0; count < messages.size(); count ++)
       {
         mercuryMessage tempMessage =
           (mercuryMessage) messages.elementAt(count);

         messageStream.writeUTF(tempMessage.messageFor);
          messageStream.writeUTF(tempMessage.messageFrom);
         messageStream.writeUTF(tempMessage.text);
       }

     messageStream.close();
  }
  catch (IOException E) {
     serverOutput(strings.get(thisClass, "writemsgerror") + "\n");
  }

  return;
}

protected void shutdown()
{
```

```
    serverOutput(strings.get(thisClass, "shutting") + "\n");

    if (currentConnections > 0)
      {
        serverOutput(strings.get(thisClass, "disconnecting") + "\n");

        synchronized (connections)
          {
            for (int count = 0; count < currentConnections;
                 count ++)
              {
                mercuryClientSocket who =
                  (mercuryClientSocket)
                  connections.elementAt(count);

                try {
                  who.sendDisconnect(who.user.id,
                        who.strings.get(thisClass,
                              "shuttinggoodbye"));
                }
                catch (IOException e) {}
              }
          }

        disconnectAll(true);
      }
    else
      serverOutput(strings.get(thisClass, "nousers") + "\n");

    serverOutput(strings.get(thisClass, "peakconn") + " " +
            peakConnections + "\n");
    serverOutput(strings.get(thisClass, "totalconn") + " " +
            totalConnections + "\n");

    serverOutput(strings.get(thisClass, "savingmsg") + "\n");
    saveMessages();

    serverOutput(strings.get(thisClass, "closinglog") + "\n");
    try {
      log.close();
    }
    catch (IOException F) {
      serverOutput(strings.get(thisClass, "closelogerror") + "\n");
    }

    if (graphics)
      window.dispose();

    stop = true;

    if (!graphics)
      {
        System.out.println("");
        System.out.println(strings.get(thisClass,
                           "shutdowncomplete"));
      }

    if (!externalShutdown)
      System.exit(0);
  }
```

```
    public static void main(String[] args)
    {
      new mercuryServer(args);
      return;
    }
}
```

APPENDIX B

SOURCE CODE MERCURYCLIENT.JAVA

```
//
// mercury Chat
// mercuryClient.java
//

import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;


public class mercuryClient
   extends Thread
{
   protected Socket socket;
   protected DataInputStream istream;
   protected DataOutputStream ostream;
   protected boolean stop = false;

   protected mercuryPanel mainPanel;
   protected Vector userList = new Vector();
   protected Vector roomList = new Vector();
   protected Vector ignoredUsers = new Vector();

   private Class thisClass = mercuryClient.class;


   public mercuryClient(String host, String name, int portnumber,
                        mercuryPanel panel)
     throws UnknownHostException, IOException, Exception
   {
     super("mercury Chat client thread");

     mainPanel = panel;


     socket = new Socket(host, portnumber);


     ostream = new DataOutputStream(socket.getOutputStream());


     istream = new DataInputStream(socket.getInputStream());

     synchronized (istream) {
        synchronized (ostream) {

          sendProtocol(protocolVersion);

          if (istream.readShort() == mercuryCommand.SETPROTO)
             receiveProtocol();
          else
             {
               istream.readFully(new byte[istream.available()]);

               new mercuryTextDialog(mainPanel.parentWindow,
                    mainPanel.strings.get(thisClass, "warning"),
                    mainPanel.strings.get(thisClass, "lostdata"),
                    60, 15, TextArea.SCROLLBARS_VERTICAL_ONLY,
```

```
                false, mainPanel.strings.get("dismiss"));
            }
        }
    }
    start();
    sendUserInfo();
}

public void run()
{
    while (!stop)
        try {
            parseCommand();
            ostream.flush();
        }
        catch (EOFException a) {
            if (!stop)
                {
                    try {
                        istream.reset();
                        continue;
                    }
                    catch (IOException b) {
                        if (attemptReconnect())
                            continue;
                        else
                            {
                                b.printStackTrace();
                                lostConnection();
                            }
                    }
                }
            return;
        }
        catch (IOException c) {
            if (!stop)
                {
                    if (attemptReconnect())
                        continue;
                    else
                        {
                            c.printStackTrace();
                            lostConnection();
                        }
                }
            return;
        }
}

private boolean attemptReconnect()
{
    System.out.println("Connection lost; attempting silent reconnect");

    try {
synchronized (istream) {
            synchronized (ostream) {
                socket = new Socket(mainPanel.host,
                                    mainPanel.portNumber);
                ostream = new DataOutputStream(socket
                                               .getOutputStream());
```

```
            istream = new DataInputStream(socket
                                .getInputStream());
          }
        }
      }
      catch (Exception ee) {
        return (false);
      }
      return (true);
    }

    private void parseCommand()
      throws IOException
    {
      short commandType = 0;

      synchronized (istream) {

        istream.mark(1024);
        commandType = istream.readShort();
        if (stop)
          return;
            switch (commandType) {

            case mercuryCommand.NOOP:
              {
                break;
              }

            case mercuryCommand.PING:
              {
                break;
              }

            case mercuryCommand.CONNECT:
              {
                receiveConnect();
                break;
              }

            case mercuryCommand.USERINFO:
              {
                receiveUserInfo();
                break;
              }

            case mercuryCommand.SERVERMESS:
              {
                receiveServerMess();
                break;
              }

            case mercuryCommand.DISCONNECT:
              {
                receiveDisconnect();
                break;
              }

            case mercuryCommand.ROOMLIST:
              {
```

```
        receiveRoomList();
        break;
    }

  case mercuryCommand.INVITE:
    {
        receiveInvite();
        break;
    }

  case mercuryCommand.ENTERROOM:
    {
        receiveEnterRoom();
        break;
    }

  case mercuryCommand.BOOTUSER:
    {
        receiveBootUser();
        break;
    }

  case mercuryCommand.BANUSER:
    {
        receiveBanUser();
        break;
    }

  case mercuryCommand.ALLOWUSER:
    {
        receiveAllowUser();
        break;
    }

  case mercuryCommand.ACTIVITY:
    {
        receiveActivity();
        break;
    }

  case mercuryCommand.CHATTEXT:
    {
        receiveChatText();
        break;
    }

  case mercuryCommand.LINE:
    {
        receiveLine();
        break;
    }

  case mercuryCommand.RECT:
    {
        receivePoly(mercuryCommand.RECT);
        break;
    }

  case mercuryCommand.OVAL:
    {
```

```
        receivePoly(mercuryCommand.OVAL);
        break;
    }

case mercuryCommand.DRAWTEXT:
    {
        receiveDrawText();
        break;
    }

case mercuryCommand.DRAWPICTURE:
    {
        receiveDrawPicture();
        break;
    }

case mercuryCommand.CLEARCANV:
    {
        receiveClearCanv();
        break;
    }

case mercuryCommand.PAGEUSER:
    {
        receivePageUser();
        break;
    }

case mercuryCommand.INSTANTMESS:
    {
        receiveInstantMess();
        break;
    }

case mercuryCommand.STOREDMESS:
    {
        receiveStoredMess();
        break;
    }

case mercuryCommand.ERROR:
    {
        receiveError();
        break;
    }

default:
    {
        byte[] foo = new byte[istream.available()];
        istream.readFully(foo);

        String stringFoo = new String(foo);
        if (stringFoo.startsWith("welcome to mercury"))
            {
                notifyObsoleteV1();
                shutdown(false);
                mainPanel.offline();
                return;
            }
```

```
                    System.out.println(mainPanel.strings
                                .get(thisClass,
                                   "unknowncommand") +
                                " " + commandType);
                    break;
                }
            }
        }
    }
}

protected mercuryUser findUser(int userId)
{
    mercuryUser tmpUser = null;
    mercuryUser returnUser = null;

    if (userId == 0)
        return (null);

    for (int count = 0; count < userList.size(); count ++)
        {
            tmpUser = (mercuryUser) userList.elementAt(count);

            if (tmpUser.id == userId)
                {
                    returnUser = tmpUser;
                    break;
                }
        }

    return (returnUser);
}

protected mercuryUser readUser()
    throws IOException
{
    int userId = 0;
    userId = istream.readInt();
    return (findUser(userId));
}

protected void sendRecipients()
    throws IOException
{
    String[] selectedUsers;
    int numberUsers = 0;
    if (mainPanel.sendToAllCheckbox.getState())
        {
            ostream.writeInt(0);
        }
    else
        {
            selectedUsers = mainPanel.sendToList.getSelectedItems();

            numberUsers = selectedUsers.length;

            ostream.writeInt(numberUsers);

            for (int count1 = 0; count1 < numberUsers; count1 ++)
                {
```

```
            for (int count2 = 0; count2 < userList.size();
                count2++)
              {
                mercuryUser tmp = (mercuryUser)
                  userList.elementAt(count2);

                if (selectedUsers[count1].equals(tmp.name))
                  ostream.writeInt(tmp.id);
              }
          }
      }
  }

  protected void addToIgnored(String name)
  {
    if (!isIgnoredUser(name))
      ignoredUsers.addElement(name);

    mainPanel.messagesArea.append("<<" + mainPanel.strings
                  .get(thisClass, "ignoreusers") + " ");
    for (int count = 0; count < ignoredUsers.size(); count ++)
      {
        mainPanel.messagesArea
          .append((String) ignoredUsers.elementAt(count));
        if (count < (ignoredUsers.size() - 1))
          mainPanel.messagesArea.append(", ");
      }
    mainPanel.messagesArea.append(">>\n");
  }

  protected void addToIgnored()
  {
    String[] selectedUsers;
    int numberUsers = 0;

    selectedUsers = mainPanel.sendToList.getSelectedItems();

    numberUsers = selectedUsers.length;

    for (int count1 = 0; count1 < numberUsers; count1 ++)
      {
        for (int count2 = 0; count2 < userList.size(); count2++)
          {
0           mercuryUser tmp = (mercuryUser)
              userList.elementAt(count2);

            if (selectedUsers[count1].equals(tmp.name))
              addToIgnored(tmp.name);
          }
      }
  }

  protected boolean isIgnoredUser(String who)
  {
    for (int count = 0; count < ignoredUsers.size(); count ++)
      if (((String) ignoredUsers.elementAt(count)).equals(who))
        return (true);
    return (false);
  }
```

```
protected void receiveConnect()
  throws IOException
{
  String userName = istream.readUTF();

  mainPanel.messagesArea
    .append("<<" + mainPanel.strings.get(thisClass, "newuser") +
        " \"" + userName + "\" " +
        mainPanel.strings.get(thisClass, "connected") + ">>\n");
}

protected void sendUserInfo()
  throws IOException
{
  synchronized (ostream)
    {
      ostream.writeShort(mercuryCommand.USERINFO);
      ostream.writeInt(0);
      ostream.writeUTF(mainPanel.name);
      ostream.writeUTF(mainPanel.encryptedPassword);
      ostream.writeBoolean(mainPanel.passwordEncryptor
                .canEncrypt);
      ostream.writeUTF(mainPanel.additional);

    }
}

protected void receiveUserInfo()
  throws IOException
{
  int tmpId = 0;
  String tmpName = "";
  String tmpAdditional = "";
  String tmpChatroomName = "";
  mercuryUser newUser;

  tmpId = istream.readInt();
  tmpName = istream.readUTF();
  istream.readUTF();
  istream.readBoolean();
  tmpAdditional = istream.readUTF();

  if (tmpName.equals(mainPanel.name))
    {
      mainPanel.clientId = tmpId;
    }
  else
    {
      newUser = new mercuryUser(tmpId, tmpName, "", tmpAdditional);

      userList.addElement(newUser);
    }
}

protected void receiveServerMess()
  throws IOException
{
  String message = "";

  message = istream.readUTF();
```

```
    new mercuryInfoDialog(mainPanel.parentWindow,
                  mainPanel.strings.get(thisClass,
                              "servermessage"),
                  true, message,
                  mainPanel.strings.get("ok"));
  }

  protected void sendDisconnect()
    throws IOException
  {
    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.DISCONNECT);
        ostream.writeInt(mainPanel.clientId);
        ostream.writeUTF("");
      }
  }

  protected void receiveDisconnect()
    throws IOException
  {
    int tmpId = 0;
    mercuryUser tmpUser;
    String disconnectMess = "";
    java.awt.List list = mainPanel.sendToList;

    tmpId = istream.readInt();
    disconnectMess = istream.readUTF();
    istream.readFully(new byte[istream.available()]);

    if ((tmpId == mainPanel.clientId) || (tmpId == 0))
      {
        if (disconnectMess.equals(""))
          disconnectMess =
            mainPanel.strings.get(thisClass, "noreason");

        new mercuryInfoDialog(mainPanel.parentWindow,
                      mainPanel.strings.get(thisClass,
                                  "disconnected"),
                      true, disconnectMess,
                      mainPanel.strings.get("ok"));

        shutdown(false);
        mainPanel.offline();
      }
    else
      {
        tmpUser = findUser(tmpId);

        if (tmpUser == null)
          return;

        mainPanel.messagesArea
          .append("<<" + tmpUser.name + " " +
              mainPanel.strings.get(thisClass,
                          "isdisconnecting")
              + ">>\n");

        synchronized (list)
```

```
        {
          for (int count2 = 0; count2 < list.getItemCount();
              count2 ++)
            {
              if (list.getItem(count2).equals(tmpUser.name))
                {
                  if (list.isIndexSelected(count2))
                    list.select(0);
                  list.remove(count2);
                  break;
                }
            }

          if (list.getSelectedItems().length == 0)
            mainPanel.sendToAllCheckbox.setState(true);
        }

      mercuryRoomInfo roomInfo = null;

      if (mainPanel.roomInfoArray != null)
        {
          for (int count2 = 0;
              count2 < mainPanel.roomInfoArray.length;
              count2 ++)
            if (mainPanel.roomInfoArray[count2]
              .name.equals(tmpUser.chatroomName))
              {
                roomInfo = mainPanel
                  .roomInfoArray[count2];
                break;
              }
          if (roomInfo != null)
            {
              roomInfo.userNames
                .removeElement(tmpUser.name);
              roomInfo.userNames.trimToSize();
            }
        }

      userList.removeElement((Object) tmpUser);
      userList.trimToSize();

      if (mainPanel.roomControlDialog != null)
        mainPanel.roomControlDialog.updateLists();
    }
}

protected void requestRoomList()
  throws IOException
{
  synchronized (ostream)
    {
      ostream.writeShort(mercuryCommand.ROOMLIST);
      ostream.writeShort(0);
    }
}

protected void receiveRoomList()
  throws IOException
{
```

```
    int howManyRooms = 0;
    mercuryRoomInfo[] roomList;
    mercuryRoomInfo tmp = null;

    howManyRooms = istream.readShort();

    roomList = new mercuryRoomInfo[howManyRooms];

    for (int count1 = 0; count1 < howManyRooms; count1 ++)
      {
        tmp = new mercuryRoomInfo();

        tmp.name = istream.readUTF();
        tmp.creatorName = istream.readUTF();
        tmp.priv = istream.readBoolean();
        tmp.invited = istream.readBoolean();
        int numUsers = istream.readInt();

        for (int count2 = 0; count2 < numUsers; count2 ++)
          tmp.userNames.addElement(istream.readUTF());

        roomList[count1] = tmp;

        if (tmp.name.equals(mainPanel.currentRoom.name))
          mainPanel.currentRoom = tmp;
      }

    mainPanel.roomInfoArray = roomList;

    if (mainPanel.roomsDialog != null)
      mainPanel.roomsDialog.receivedList();

    if (mainPanel.roomControlDialog != null)
      mainPanel.roomControlDialog.updateLists();
  }

  protected void sendInvite(int userId, String roomName)
    throws IOException
  {
    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.INVITE);
        ostream.writeInt(mainPanel.clientId);
        ostream.writeUTF(roomName);
        ostream.writeInt(userId);
      }
  }

  protected void receiveInvite()
    throws IOException
  {
    mercuryUser fromUser = null;
    String roomName = null;

    fromUser = readUser();

    roomName = istream.readUTF();
    istream.readInt(); // Discard our user id

    if (fromUser == null)
```

```
      return;

    if (isIgnoredUser(fromUser.name))
      return;

    new mercuryInfoDialog(mainPanel.parentWindow,
                  mainPanel.strings.get(thisClass, "invitation"),
                  false, mainPanel.strings.get(thisClass,
                                  "invitedtoroom") +
                  " \"" + roomName + "\" " +
                  mainPanel.strings.get(thisClass, "byuser") +
                  " " + fromUser.name,
                  mainPanel.strings.get("ok"));
}

protected void sendEnterRoom(String roomName, boolean priv,
                  String password)
    throws IOException
{
    if (roomName.equals(""))
      return;

    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.ENTERROOM);
        ostream.writeInt(mainPanel.clientId);
        ostream.writeUTF(roomName);
        ostream.writeBoolean(priv);
        ostream.writeUTF(password);
        ostream.writeBoolean(mainPanel.passwordEncryptor
                  .canEncrypt);
      }
}

protected void receiveEnterRoom()
    throws IOException
{
    int userId = 0;
    String newRoomName = "";
    mercuryUser tmpUser;
    java.awt.List list = mainPanel.sendToList;

    userId = istream.readInt();
    newRoomName = istream.readUTF();
    istream.readBoolean();
    istream.readUTF();
    istream.readBoolean();

    if (userId == mainPanel.clientId)
      {
        if (!mainPanel.currentRoom.name.equals(""))
          mainPanel.messagesArea
            .append("<<" +
                  mainPanel.strings.get(thisClass, "entering") +
                  " \"" + newRoomName + "\">>\n");
        if (mainPanel.sendToList.getItemCount() > 0)
          mainPanel.sendToList.removeAll();
        mainPanel.sendToAllCheckbox.setState(true);
        mainPanel.canvas.clear();
```

```
        for (int count1 = 0; count1 < userList.size(); count1 ++)
          {
            tmpUser = (mercuryUser) userList.elementAt(count1);

            if (tmpUser.chatroomName.equals(newRoomName))
              mainPanel.sendToList.add(tmpUser.name);
          }

        for (int count1 = 0;
            count1 < mainPanel.roomInfoArray.length; count1 ++)
          {
            mercuryRoomInfo roomInfo =
              mainPanel.roomInfoArray[count1];

            if (roomInfo.name.equals(newRoomName))
              {
                mainPanel.currentRoom = roomInfo;
                break;
              }
          }

        if (mainPanel.roomControlDialog != null)
          {
            mainPanel.roomControlDialog.dispose();
            mainPanel.roomControlDialog = null;
          }
        if (mainPanel.currentRoom.creatorName
          .equals(mainPanel.name))
          mainPanel.roomControlDialog =
            new mercuryRoomControlDialog(mainPanel);

        return;
      }

    for (int count1 = 0; count1 < userList.size(); count1 ++)
      {
        tmpUser = (mercuryUser) userList.elementAt(count1);

        if (tmpUser.id != userId)
          continue;

        String oldRoomName = tmpUser.chatroomName;
        if (oldRoomName.equals(mainPanel.currentRoom.name))
          {
            if (!oldRoomName.equals(""))
              mainPanel.messagesArea
                .append("<<" + tmpUser.name + " " +
                    mainPanel.strings.get(thisClass,
                                "movedto") +
                    " \"" + newRoomName + "\">>\n");

            synchronized (list) {
              for (int count2 = 0; count2 < list.getItemCount();
                count2 ++)
                {
                  if (list.getItem(count2)
                    .equals(tmpUser.name))
                    {
                      if (list.isIndexSelected(count2))
                        list.select(0);
```

```
                        list.remove(count2);
                        break;
                    }
                }
            }
        }
    else if (newRoomName.equals(mainPanel.currentRoom.name))
        {
        if (!oldRoomName.equals(""))
            mainPanel.messagesArea
              .append("<<" + tmpUser.name + " " +
                  mainPanel.strings.get(thisClass,
                                        "entering") +
                  ">>\n");

        synchronized (list) {
            list.add(tmpUser.name);
        }
        }

    tmpUser.chatroomName = newRoomName;

    break;
    }

  if (mainPanel.roomControlDialog != null)
    requestRoomList();
}

protected void sendBootUser(int userId, String roomName)
  throws IOException
{
  synchronized (ostream)
    {
      ostream.writeShort(mercuryCommand.BOOTUSER);
      ostream.writeInt(mainPanel.clientId);
      ostream.writeUTF(roomName);
      ostream.writeInt(userId);
    }
}

protected void receiveBootUser()
  throws IOException
{
  mercuryUser fromUser = null;
  String roomName = null;

  fromUser = readUser();

  roomName = istream.readUTF();
  istream.readInt(); // Discard our user id

  if (fromUser == null)
    return;

  new mercuryInfoDialog(mainPanel.parentWindow,
              mainPanel.strings.get(thisClass, "booted"), true,
              mainPanel.strings.get(thisClass, "bootedfrom") +
              " \"" + roomName + "\" " +
              mainPanel.strings.get(thisClass, "byuser") +
```

```
                " " + fromUser.name,
                mainPanel.strings.get("ok"));
  }

  protected void sendBanUser(int userId, String roomName)
    throws IOException
  {
    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.BANUSER);
        ostream.writeInt(mainPanel.clientId);
        ostream.writeUTF(roomName);
        ostream.writeInt(userId);
      }
  }

  protected void receiveBanUser()
    throws IOException
  {
    mercuryUser fromUser = null;
    String roomName = null;

    fromUser = readUser();

    roomName = istream.readUTF();
    istream.readInt(); // Discard our user id

    new mercuryInfoDialog(mainPanel.parentWindow,
                mainPanel.strings.get(thisClass, "banned"), true,
                mainPanel.strings.get(thisClass, "bannedfrom") +
                " \"" + roomName + "\"",
                mainPanel.strings.get("ok"));
  }

  protected void sendAllowUser(int userId, String roomName)
    throws IOException
  {
    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.ALLOWUSER);
        ostream.writeInt(mainPanel.clientId);
        ostream.writeUTF(roomName);
        ostream.writeInt(userId);
      }
  }

  protected void receiveAllowUser()
    throws IOException
  {
    mercuryUser fromUser = null;
    String roomName = null;

    fromUser = readUser();

    roomName = istream.readUTF();
    istream.readInt(); // Discard our user id

    if (fromUser == null)
      return;
```

```
  if (isIgnoredUser(fromUser.name))
    return;

  new mercuryInfoDialog(mainPanel.parentWindow,
                mainPanel.strings.get(thisClass, "allowed"),
                true,
                mainPanel.strings.get(thisClass, "allowedto") +
                " \"" + roomName + "\"",
                mainPanel.strings.get("ok"));
}

protected void sendActivity(short activity)
  throws IOException
{
  synchronized (ostream)
    {
      ostream.writeShort(mercuryCommand.ACTIVITY);
      ostream.writeInt(mainPanel.clientId);
      ostream.writeShort(activity);
      sendRecipients();
    }
}

protected void receiveActivity()
  throws IOException
{

  mercuryUser fromUser = null;
  short activity = 0;
  int numForUsers = 0;

  fromUser = readUser();

  activity = istream.readShort();
  numForUsers = istream.readInt();
  for (int count = 0; count < numForUsers; count ++)
    istream.readInt();

  if (fromUser == null)
    return;

  if (isIgnoredUser(fromUser.name))
    return;

  String tmpString = "";
  if (activity == mercuryCommand.ACTIVITY_DRAWING)
    tmpString = mainPanel.strings.get(thisClass, "drawing") + " " +
      fromUser.name;
  else if (activity == mercuryCommand.ACTIVITY_TYPING)
    tmpString = mainPanel.strings.get(thisClass, "typing") + " " +
      fromUser.name;
  if (!mainPanel.activityField.getText().equals(tmpString))
    mainPanel.activityField.setText(tmpString);
}

protected void sendChatText(String data)
  throws IOException
{
  synchronized (ostream)
    {
```

```
            ostream.writeShort(mercuryCommand.CHATTEXT);
            ostream.writeInt(mainPanel.clientId);
            if (mainPanel.sendToAllCheckbox.getState())
               ostream.writeBoolean(false);
            else
               ostream.writeBoolean(true);
            ostream.writeShort(0); // No colour
            ostream.writeUTF(data);

            sendRecipients();
         }
      }

      protected void receiveChatText()
         throws IOException
      {
         mercuryUser fromUser = null;
         boolean priv = false;
         short colour = 0;
         String data = "";
         String output = "";
         int numForUsers = 0;

         fromUser = readUser();

         priv = istream.readBoolean();
         colour = istream.readShort();
         data = istream.readUTF();

         numForUsers = istream.readInt();
         for (int count = 0; count < numForUsers; count ++)
            istream.readInt();

         if (fromUser != null)
            {
               if (isIgnoredUser(fromUser.name))
                  return;

               if (fromUser != null)
                  {
                     if (priv)
                        output += "*" + mainPanel.strings
                           .get(thisClass, "privatefrom") + " " +
                           fromUser.name + "*> ";
                     else
                        output += fromUser.name + "> ";
                  }
            }

         output += data;

         mainPanel.activityField.setText("");

         mainPanel.messagesArea.append(output);
      }

      protected void sendLine(short colour, short startx, short starty,
                              short endx, short endy, short thick)
         throws IOException
      {
```

```
    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.LINE);
        ostream.writeInt(mainPanel.clientId);
        ostream.writeShort(colour);
        ostream.writeShort(startx);
        ostream.writeShort(starty);
        ostream.writeShort(endx);
        ostream.writeShort(endy);
        ostream.writeShort(thick);

        sendRecipients();
      }
  }

  protected void receiveLine()
    throws IOException
  {

    mercuryUser fromUser = null;
    short colournum = 0;
    short startx = 0;
    short starty = 0;
    short endx = 0;
    short endy = 0;
    short thick = 0;
    int numForUsers = 0;
    Color colour;

    fromUser = readUser();

    colournum = istream.readShort();
    startx = istream.readShort();
    starty = istream.readShort();
    endx = istream.readShort();
    endy = istream.readShort();
    thick = istream.readShort();

    numForUsers = istream.readInt();
    for (int count = 0; count < numForUsers; count ++)
      istream.readInt();

    if (fromUser != null)
      if (isIgnoredUser(fromUser.name))
        return;

    colour = mainPanel.canvas.colourArray[colournum];
    mainPanel.canvas.drawLine(colour, startx, starty, endx, endy,
                              thick, mercuryCanvas.MODE_PAINT);
  }

  protected void sendPoly(short colour, short x, short y, short width,
                          short height, short thick, boolean fill, int kind)
    throws IOException
  {

    synchronized (ostream)
      {
        ostream.writeShort(kind);
        ostream.writeInt(mainPanel.clientId);
```

```
        ostream.writeShort(colour);
        ostream.writeShort(x);
        ostream.writeShort(y);
        ostream.writeShort(width);
        ostream.writeShort(height);
        ostream.writeShort(thick);
        ostream.writeBoolean(fill);

        sendRecipients();
      }
}

protected void receivePoly(short kind)
  throws IOException
{
  mercuryUser fromUser = null;
  short colournum = 0;
  short x = 0;
  short y = 0;
  short height = 0;
  short width = 0;
  short thick = 0;
  boolean fill = false;
  int numForUsers = 0;
  Color colour;

  fromUser = readUser();

  colournum = istream.readShort();
  x = istream.readShort();
  y = istream.readShort();
  width = istream.readShort();
  height = istream.readShort();
  thick = istream.readShort();
  fill = istream.readBoolean();

  numForUsers = istream.readInt();
  for (int count = 0; count < numForUsers; count ++)
    istream.readInt();

  if (fromUser != null)
    if (isIgnoredUser(fromUser.name))
      return;

  colour = mainPanel.canvas.colourArray[colournum];

  if (kind == mercuryCommand.RECT)
    mainPanel.canvas.drawRect(colour, x, y, width, height, fill,
                  thick, mercuryCanvas.MODE_PAINT);
  else if (kind == mercuryCommand.OVAL)
    mainPanel.canvas.drawOval(colour, x, y, width, height, fill,
                  thick, mercuryCanvas.MODE_PAINT);
}

protected void sendDrawText(short colour, short x, short y, short type,
            short attribs, short size, String text)
  throws IOException
{
  synchronized (ostream)
    {
```

```
        ostream.writeShort(mercuryCommand.DRAWTEXT);
        ostream.writeInt(mainPanel.clientId);
        ostream.writeShort(colour);
        ostream.writeShort(x);
        ostream.writeShort(y);
        ostream.writeShort(type);
        ostream.writeShort(attribs);
        ostream.writeShort(size);
        ostream.writeUTF(text);

        sendRecipients();
      }
}

protected void receiveDrawText()
  throws IOException
{

  mercuryUser fromUser = null;
  short colournum = 0;
  short x = 0;
  short y = 0;
  short type = 0;
  short size = 0;
  short attribs = 0;
  String text = "";
  int numForUsers = 0;
  Color colour;

  fromUser = readUser();

  colournum = istream.readShort();
  x = istream.readShort();
  y = istream.readShort();
  type = istream.readShort();
  attribs = istream.readShort();
  size = istream.readShort();
  text = istream.readUTF();

  numForUsers = istream.readInt();
  for (int count = 0; count < numForUsers; count ++)
    istream.readInt();

  if (fromUser != null)
    if (isIgnoredUser(fromUser.name))
      return;

  colour = mainPanel.canvas.colourArray[colournum];

  mainPanel.canvas.drawText(colour, x, y, type, attribs, size, text,
                    mainPanel.canvas.MODE_PAINT);
}

protected void sendDrawPicture(short x, short y, File pictureFile)
  throws IOException
{
  int fileLength = (int) pictureFile.length();;

  byte[] byteArray = new byte[fileLength];
  try {
```

```
      FileInputStream fileStream = new FileInputStream(pictureFile);

      if (fileStream.read(byteArray) < fileLength)
        {
          new mercuryInfoDialog(mainPanel.parentWindow,
                        mainPanel.strings
                        .get(thisClass, "error"),
                        true,
                        mainPanel.strings
                        .get(thisClass, "readpictureerror"),
                        mainPanel.strings.get("ok"));
          return;
        }
    }
    catch (Exception e) {
      e.printStackTrace();
      return;
    }

    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.DRAWPICTURE);
        ostream.writeInt(mainPanel.clientId);
        ostream.writeShort(x);
        ostream.writeShort(y);
        ostream.writeInt(fileLength);
        ostream.write(byteArray, 0, fileLength);

        sendRecipients();
      }
  }

  protected void receiveDrawPicture()
    throws IOException
  {
    mercuryUser fromUser = null;
    short x = 0;
    short y = 0;
    int length = 0;
    byte[] data = null;
    int numForUsers = 0;

    fromUser = readUser();

    x = istream.readShort();
    y = istream.readShort();
    length = istream.readInt();

    data = new byte[length];
    int read = 0;
    while (read < length)
      read += istream.read(data, read, (length - read));

    numForUsers = istream.readInt();
    for (int count = 0; count < numForUsers; count ++)
      istream.readInt();

    if (fromUser != null)
      if (isIgnoredUser(fromUser.name))
        return;
```

```
    Image theImage = null;

    try {
      theImage = mainPanel.getToolkit().createImage(data);
      mainPanel.getToolkit().prepareImage(theImage, -1, -1,
                                mainPanel.canvas);
      while ((mainPanel.getToolkit().checkImage(theImage, -1, -1,
        mainPanel.canvas) & mainPanel.canvas.ALLBITS) == 0)
        {}
    }
    catch (Exception e) {
      e.printStackTrace();
      return;
    }

    mainPanel.canvas.drawPicture(x, y, theImage);
  }

  protected void sendClearCanv()
    throws IOException
  {
    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.CLEARCANV);
        ostream.writeInt(mainPanel.clientId);

        sendRecipients();
      }
  }

  protected void receiveClearCanv()
    throws IOException
  {
    mercuryUser fromUser = null;
    int numForUsers = 0;

    fromUser = readUser();

    numForUsers = istream.readInt();
    for (int count = 0; count < numForUsers; count ++)
      istream.readInt();

    if (fromUser == null)
      return;

    if (isIgnoredUser(fromUser.name))
      return;

    mainPanel.canvas.clear();
    mainPanel.messagesArea
      .append("<<" + fromUser.name + " " +
          mainPanel.strings.get(thisClass, "clearedcanvas") +
          ">>\n");
  }

  protected void sendPageUser()
    throws IOException
  {
    String[] selectedUsers;
```

```
    int numberUsers = 0;

    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.PAGEUSER);
        ostream.writeInt(mainPanel.clientId);

        sendRecipients();
      }

    if (mainPanel.sendToAllCheckbox.getState())
      mainPanel.messagesArea
        .append("<<" + mainPanel.strings.get(thisClass, "pagingall") +
            ">>\n");

    else
      {
        selectedUsers = mainPanel.sendToList.getSelectedItems();

        numberUsers = selectedUsers.length;

        if (numberUsers == 1)
          mainPanel.messagesArea
            .append("<<" + mainPanel.strings.get(thisClass,
                                    "paging") +
                " " + selectedUsers[0] + ">>\n");
        else
          {
            mainPanel.messagesArea
              .append("<<" + mainPanel.strings.get(thisClass,
                                      "pagingsome"));
            for (int count = 0; count < numberUsers; count ++)
              mainPanel.messagesArea.append(" "
                          + selectedUsers[count]);
            mainPanel.messagesArea.append(">>\n");
          }
      }
}

protected void receivePageUser()
  throws IOException
{
  mercuryUser fromUser = null;
  int numForUsers = 0;

  fromUser = readUser();

  numForUsers = istream.readInt();
  for (int count = 0; count < numForUsers; count ++)
    istream.readInt();

  if (fromUser == null)
    return;

  if (isIgnoredUser(fromUser.name))
    return;

  mainPanel.messagesArea.append("<<" + fromUser.name + " " +
                mainPanel.strings.get(thisClass,
                          "pagingyou") +
```

```
                    ">>\n");

  if (mainPanel.playSound)
    {
          try {
            URL soundURL =
              new URL(mainPanel.mercuryURL.getProtocol(),
                   mainPanel.mercuryURL.getHost(),
                   mainPanel.mercuryURL.getFile() +
                   "mercuryBuzzer.au");
            Applet.newAudioClip(soundURL).play();
          }
          catch (Exception argh) {
            for (int count = 0; count < 3; count ++)
              mainPanel.getToolkit().beep();

            sendError(fromUser.id,
                 mercuryCommand.ERROR_NOSOUND);
          }
        }
    }
  else
    sendError(fromUser.id, mercuryCommand.ERROR_NOPAGE);
}

protected void sendError(int toWhom, short code)
  throws IOException
{
  synchronized (ostream)
    {
      ostream.writeShort(mercuryCommand.ERROR);
      ostream.writeInt(mainPanel.clientId);
      ostream.writeShort(code);
      ostream.writeInt(1); // 1 recipient
      ostream.writeInt(toWhom);
    }
}

public void sendInstantMess(int whoFor, String message)
  throws IOException
{
  synchronized (ostream)
    {
      ostream.writeShort(mercuryCommand.INSTANTMESS);
      ostream.writeInt(mainPanel.clientId);
      ostream.writeInt(whoFor);
      ostream.writeUTF(message);
    }
}

public void receiveInstantMess()
  throws IOException
{
  mercuryUser fromUser = null;
  String message = "";

  fromUser = readUser();

  istream.readInt(); // Our id
  message = istream.readUTF();
```

```
    if (fromUser == null)
      return;

    if (isIgnoredUser(fromUser.name))
      return;

    for (int count = 0; count < mainPanel.instantMessageDialogs.size();
        count ++)
      {
        mercuryInstantMessageDialog tmp =
          (mercuryInstantMessageDialog) mainPanel
          .instantMessageDialogs.elementAt(count);

        if (tmp.fromUser == fromUser)
          {
            tmp.addMessage(message);
            return;
          }
      }

    mainPanel.instantMessageDialogs
      .addElement(new mercuryInstantMessageDialog(mainPanel, fromUser,
                                    message));

    return;
  }

  public void sendLeaveMess(String whofor, String message)
    throws IOException
  {
    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.LEAVEMESS);
        ostream.writeInt(mainPanel.clientId);
        ostream.writeUTF(whofor);
        ostream.writeUTF(message);
      }
  }

  public void sendReadMess()
    throws IOException
  {
    synchronized (ostream)
      {
        ostream.writeShort(mercuryCommand.READMESS);
        ostream.writeInt(mainPanel.clientId);
      }
  }

  public void receiveStoredMess()
    throws IOException
  {
    short numberMessages = 0;
    String from = "";
    String message = "";

    numberMessages = istream.readShort();

    if (numberMessages == 0)
```

```
    {
      new mercuryInfoDialog(mainPanel.parentWindow,
                  mainPanel.strings.get(thisClass,
                              "none"), true,
                  mainPanel.strings.get(thisClass,
                              "nounread"),
                  mainPanel.strings.get("ok"));
      return;
    }

  for (int count = 0; count < numberMessages; count ++)
    {
      from = istream.readUTF();
      message = istream.readUTF();

      new mercuryTextDialog(mainPanel.parentWindow,
                  mainPanel.strings.get(thisClass,
                              "messagefrom") +
                  " " + from, message, 40, 10,
                  TextArea.SCROLLBARS_VERTICAL_ONLY,
                  true, mainPanel.strings.get("dismiss"));
    }

  new mercuryInfoDialog(mainPanel.parentWindow,
              mainPanel.strings.get(thisClass, "done"), true,
              mainPanel.strings.get(thisClass,
                          "endmessages"),
              mainPanel.strings.get("ok"));
  return;
}

protected void receiveError()
  throws IOException
{
  mercuryUser fromUser = null;
  short errorCode = 0;
  int numForUsers = 0;
  fromUser = readUser();
  errorCode = istream.readShort();
  numForUsers = istream.readInt();
  for (int count = 0; count < numForUsers; count ++)
    istream.readInt();

  if (fromUser == null)
    return;


public void lostConnection()
{
  if (!stop)
    {
      shutdown(true);
      mainPanel.offline();
    }
  return;
}

public synchronized void shutdown(boolean notifyUser)
{
  stop = true;
```

```
        try {
          istream.close();
        }
        catch (IOException e) {}

        try {
          synchronized (ostream)
            {
              ostream.flush();
              ostream.close();
            }
          socket.close();
        }
        catch (IOException e) {}

        userList.removeAllElements();

        mainPanel.currentRoom = null;

        if (mainPanel.roomsDialog != null)
          {
            mainPanel.roomsDialog.dispose();
            mainPanel.roomsDialog = null;
          }
        if (mainPanel.roomControlDialog != null)
          {
            mainPanel.roomControlDialog.dispose();
            mainPanel.roomControlDialog = null;
          }
        for (int count = 0; count < mainPanel.instantMessageDialogs.size();
             count ++)
          {
            mercuryInstantMessageDialog tmp =
              (mercuryInstantMessageDialog)
              mainPanel.instantMessageDialogs.elementAt(count);
            tmp.dispose();
          }
        mainPanel.instantMessageDialogs.removeAllElements();

        if (notifyUser)
          new mercuryInfoDialog(mainPanel.parentWindow,
                                mainPanel.strings.get(thisClass,
                                                      "disconnected"), true,
                                mainPanel.strings.get(thisClass,
                                                      "disconnectedfrom") +
                                " " + mainPanel.host,
                                mainPanel.strings.get("ok"));
        System.gc();
        return;
      }
    }
```

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]. David M. Geary, "Graphic Java 2, Volume 1.2: AWT", 3rd Edition (1998), Prentice Hall PTR.

[5]. Iain Shigeoka, "Instant Messaging in JAVA - The Jabber Protocols", 2002, Manning Publications Co., ISBN: 1930110464

[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]. 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