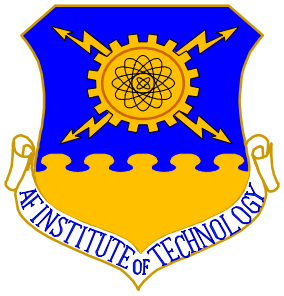
**July 2000**




# agentMom User's Manual

**Scott A. DeLoach**

GRADUATE SCHOOL OF ENGINEERING AND MANAGEMENT
AIR FORCE INSTITUTE OF TECHNOLOGY
WRIGHT-PATTERSON AIR FORCE BASE, OHIO

# Using agentMom

### *Introduction – What is agentMom*

agentMom is a framework upon which distributed multiagent systems can be developed. It is implemented in Java and provides the basic building blocks for building agents, conversations between agents, and the message that are passed in the conversations.

An overview of how agentMom works is shown in Figure 1. An agent allows itself to speak with other agents by starting a *message handler* that monitors a local port for messages. All agent communication is performed via *conversations,* which define valid sequences of messages that agents can use to communicate. When one agent wants to communicate with another agent, it starts one of its conversations as a separate Java thread. The conversation then establishes a socket connection with the other agent's message handler and sends the initial message in the conversation. When the message handler receives a message, it passes the message to the agent's *receiveMessage* method that compares the message against its known list of allowable message types to see if it is the start of a valid conversation. If the conversation is valid, the agent starts its side of the appropriate conversation, also as a separate Java thread. From that point on, all communication is controlled by the two conversation threads. Conversations send messages to each other using built in *readMessage* and *sendMessage* methods.

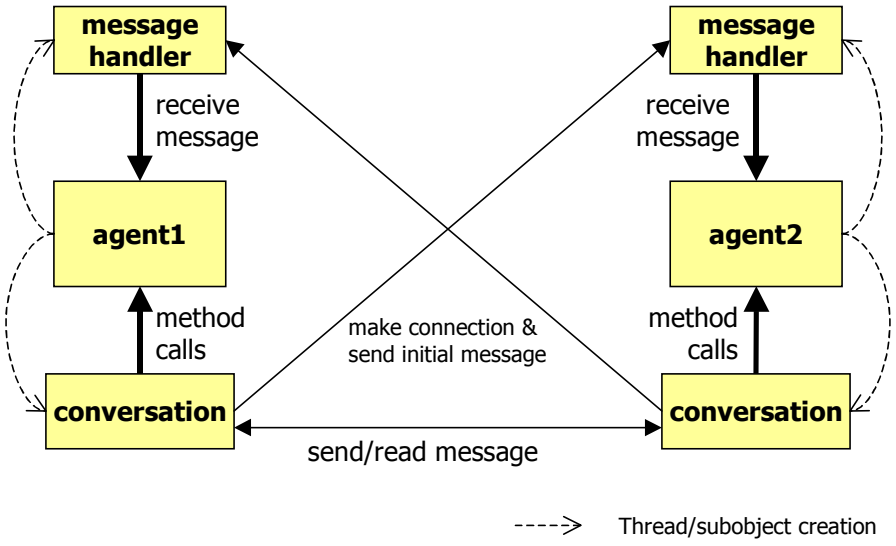



**Figure 1. agentMom**

While conversations handle the message passing back and forth between agents, they still must have a way to communicate with their parent agents. This is accomplish using method calls from the conversation back to their parents.

### *How to use agentMom*

The *afit.mom* package, which makes up the basics of agentMom, is shown in Figure 2. It consists of two abstract classes, *Agent* and *Conversation*, and three concrete classes *MessageHandler*, *Message*, and *Sorry*. The actual source code associated with these classes are in Appendix A.

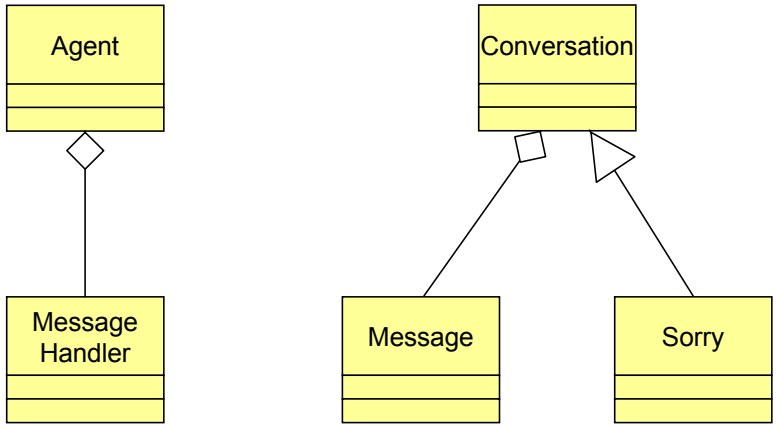



**Figure 2. agentMom Object Model**

## Agent Class

The Agent class is an abstract class that defines the minimum requirements for an agent to use agentMom. It must be runnable as a separate thread, which requires a *run* method. It also has two required parameters that must be set for each agent, the *name* of the agent and the *port* number on which it's message handler will listen for incoming messages. Also, the agent must implement the *receiveMessage* method. This is the method that will be called by the message handler when the agent has received a connection starting a new conversation. Each agent must implement this method by reading the message from the connection and determining if it is a valid conversation. Two agents that implement these methods, *ClientAgent* and *ServerAgent*, are shown in Appendix B. The receiveMessage method for the ServerAgent is shown in Figure 3. In this case, after the receiveMessage method reads the message using the `"m = (Message) input.readObject();"` method call, the performative of the message is checked to see if it is either *register* or *unregister*. In the case of the ServerAgent, these are the only two performatives the agent can recognize that start conversations in which it can participate. If it is either of these performatives, its creates a new conversation object as a new thread, sends it the initial message, and starts it running using the conversation's *run* method. If the message received does not start with a recognizable performative, the agent starts the default *Sorry* conversation, which simply sends a sorry message in reply to the performative. Therefore, all good conversations should be able to respond to a *sorry* performative when it tries to start a conversation.

## Conversation Class

The Conversation class is an abstract class that actually carries out the message passing between agents. There are two methods in the Conversation class, *readMessage* and *sendMessage* that actually pass the messages back and forth over the socket connection.

There are really two types of conversation classes that can be derived from the Conversation class, one for the conversation initiator and one for the conversation respondent. The basic difference lies in which constructor is used and the details in the abstract run method, which must be implemented in the concrete class derived from the Conversation class.

```
/**
 * receiveMessage method comment.
 */
public void receiveMessage(Socket server, ObjectInputStream input, ObjectOutputStream output) {
        int i;
        Message m;
        Server_Register s;
        Thread t;

        try {
                m = (Message) input.readObject();
                this.theText.append("Received message " + m.performative + " from "+m.sender+"\n");

                if (m.performative.equals("register")) {
                        t = new Thread(new Server_Register(server, input, output, this, m));
                        t.start(); // start new thread
                } else
                        if (m.performative.equals("unregister")) {
                                t = new Thread(new Server_Unregister(server, input, output, this, m));
                                t.start(); // start new thread
                        } else {
                System.out.println(" ** Invalid Attempt to start new conversation with performative " +
                                        m.performative + " from " + m.sender);
                                t = new Thread(new Sorry(server, input, output, this, m));
                                t.start(); // start new thread
                        }
        } catch (ClassNotFoundException cnfex) {
                System.out.println(" ** ClassNotFoundException ");
        } catch (IOException cnfex) {
                System.out.println(" ** IOException on port (ServerAgent.receiveMessage)");
        }
}
```

**Figure 3. receiveMessage Example**

An example of an initiator conversation class is the *Client_Register* class in Appendix B. To initiate this conversation, the ClientAgent creates a new Client_Register object (as a separate thread) using the Client_Register constructor. This constructor does not need to send a socket, input stream, or output stream (see second Conversation constructor in Appendix A) since, as an initiator, the conversation creates a new socket and opens an input and output stream with a second agent's message handler. When the ClientAgent starts the Client_Register conversation class, the Client_Register *run* method is started. This method controls the conversation. It creates a new connection using the following commands.

```
connection = new Socket(connectionHost, connectionPort);
output = new ObjectOutputStream(connection.getOutputStream());
output.flush();
input = new ObjectInputStream(connection.getInputStream());
```

After the connection is made, the method enters a *while* loop that iterates until the conversation is completed. Inside the while loop is a simple *switch* statement that has a case for each possible state of the conversation. Actually the state in the run method may or may not correspond one-to-one with the states of the conversation as defined in a MaSE

conversation diagram. Actually, it is possible to have one state for each state in the diagram plus a state for each transition out of a state. In a simple conversation such as the Client_Register conversation, this could be modeled as a simple sequence of statements; however, in the general case, conversations may have loops and many branches out of a single state, thus the switch within a loop provides the most general mechanism for modeling conversation states. The loop and switch statement are shown below.

```
while (notDone) {
   switch (state) {
      case 0 :
         m.performative = "register";
         m.content = service;
         sendMessage(m, output);
         state = 1;
         break;
      case 1 :
         m = readMessage(input);
         if (m.performative.equals("reply"))
            notDone = false;
         else
            parent.write("** ERROR - did not get reply back **");
         break;
   }
}
```

In the code above, the *state* variable starts at state zero. In state 0, the message performative is set to *register* and the message content is set to a string sent to the conversation by the ClientAgent when it was initialized. Actually the content of a message can take any Java *object* type. After sending the message, the state variable is set to 1 and the *break* statement takes us out of the switch statement. Since notDone is still true, we stay in the loop, this time entering the *case 1* option of the switch statement. At this point, we wait at the *readMessage* call until a message comes in from the other agent. Then, if it is what we expect (a reply performative), we process it, otherwise we print an error message. In this case, we do nothing with the reply and simply set the notDone variable to false so that we will exit the while loop.

After exiting the conversation, we close the connection with the other agent using the sequence of close statements shown below.

```
input.close();
output.close();
connection.close();
```

## Message Class

The message class defines the field used in the message passed back and forth between agents. It is fairly straightforward and consists of the following attributes.

```
public String host = null;
public int port = 0;
public String sender = null;
public String receiver = null;
public String performative = null;
public String force = null;
public String inreplyto = null;
public String language = null;
public String ontology = null;
public String replywith = null;
public Object content = null;
```

Not all of these fields need to be used. They were derived from the fields in a KQML message and only some of them are necessary. When a conversation calls the *sendMessage* method, it automatically fill the *sender*, *host*, and *port* fields using the parent agent's name and port attributes and automatically gets the host name from the system. The other fields of interest in an agentMom message are the *performative* and *content* fields. As seen above, the performative field describe the action that the message intends and is used in the agent and conversation classes to (1) determine the type of conversation being requested and (2) to control the execution of a conversation in the run method. agentMom does have any specific performative types. The user define any performative they feel are necessary. The content of an agentMom message is also very general. Basically, the message passes any valid Java object type. This can be as simple as a string, or a more complex object that encapsulates a number of attribute types. These complex objects can be used to pass multiple parameters in a single message as shown in the registration class below.

```
public class ComplexObject implements Serializable {
    String agent;
    String host;
    int port;
    String service;

    public ComplexObject(String a, String h, int p, String ser){
        agent = a;
        host = h;
        port = p;
        service = ser;
    }
}
```

This class encapsulates four parameters (three strings and an integer) that can be assigned to message content field. Note that in order to pass an object across a socket connection, it must implement the interface Serializable.

## Message Handler

The message handler class (*MessageHandler*) is very simple to use. When an agent is created, it simply creates a new message handler thread as shown below.

```
MessageHandler h = new MessageHandler(this.port, this);
h.start();
```

The two required parameters are simply the port number and a pointer to the parent agent object. When started, the message handler starts a socket server on the indicated port and waits for a connection from another agent. When a connection is received, the message handler simply calls the parent agent's *receiveMessage* method with the connection and the input and output streams. As described above the agent does the rest. The code for the main message handler *run* method is shown below.

```
public void run() {
    Socket connection;
    int counter = 1;
    ObjectOutputStream output;
    ObjectInputStream input;
    System.out.println("Server Running on " + portNo);
    while (true) {
        try {
            connection = server.accept();
            System.out.println("Connection " + counter++ + " received from "
                               + connection.getInetAddress().getHostName());
```

```
            output = new ObjectOutputStream(connection.getOutputStream());
            output.flush();
            input = new ObjectInputStream(connection.getInputStream());
            parent.receiveMessage(connection, input, output);
        } catch (IOException cnfex) {
            System.out.println(" ** IOException on port " + portNo);
        }
    }
```

## Sorry Class

The Sorry class is simply a default conversation class that sends a *sorry* message when a performative is received that is not expected by either the *receiveMessage* method of the parent agent or within the flow of a conversation.

## *Step By Step Construction*

To build an application using the agentMom framework, you need to perform the following:

1) Get a copy of agentMom classes as described in Appendix A.

2) Define your agent classes and conversations according the MaSE (Multiagent Systems Engineering) methodology. An environment, *agent*Tool, is available to help you with this.

3) For each agent class in your system, extend the agentMom Agent class.

   a) Define the *receiveMessage* method to handle all conversations for which the agent is a respondent (may be automatically generated by *agent*Tool).

   b) For each action defined in the set of conversations in which this agent may participate, define a method in the agent class. This will be your interface to the conversation.

   c) Implement the *run* method as the main procedure of the method. If your agent initiates any conversations, this is where they will originate.

   d) If you want to run your agent as a stand alone application, create a *main* method to initialize the agent running.

4) For each conversation in your system design, create two conversation classes, in initiator and a respondent class.

   a) For each initiator class, define a constructor that includes as parameters any parameters in the first message sent.

   b) For each respondent class, define a constructor that includes as a parameter the message read by the parent *receiveMessage* method before the conversation thread was started.

   c) Implement the run method (may be automatically generated by *agent*Tool).

      i) Define a *state* variable initialized to state 0.

      ii) If it is an initiator conversation, create a connection with the agent of interest.

iii) Create a switch statement within a while loop where each case in the switch statement corresponds to a state or a transition. Ensure at least one of the states exits the while loop.

iv) Close the connection.

5) Create any supporting classes for things such as

a) Objects that combine multiple parameters into a single object.

b) System setup/testing routines.

c) Components of intelligent agents.

# Appendix A

agentMom Source Code

## Agent

```
package afit.mom;

/**
 * This type was created in VisualAge.
 */
import java.net.*;
import java.io.*;
import java.awt.*;
public abstract class Agent implements Runnable {
    public String name;
    public int port;

/**
 * This method was created in VisualAge.
*/
protected Agent(String n, int p) {
    this.name = n;
    this.port = p;
}

public abstract void receiveMessage(Socket server, ObjectInputStream input,
ObjectOutputStream output);

/**
 * This method was created in VisualAge.
 */
public abstract void run();

/**
 * This method was created in VisualAge.
*/
private void write(String s) {
    System.out.println(s);
}
}
```

## Conversation

```
package afit.mom;

/**
 * This type was created in VisualAge.
 */
import java.net.*;
import java.io.*;
public abstract class Conversation extends Thread {
    protected Agent parent;
    protected Socket connection;
    protected ObjectInputStream input;
    protected ObjectOutputStream output;
    protected Message m;
    protected String connectionHost;
    protected int connectionPort;

/**
 * Conversation constructor comment.
 */
public Conversation(Agent a) {
    super();
    parent = a;
}

/**
 * Server_Register constructor comment.
 */
public Conversation(Agent a, String hostName, int portNum) {
    super(a);
    parent = a;
    connectionHost = hostName;
    connectionPort = portNum;
}


public Conversation(Socket s, ObjectInputStream i, ObjectOutputStream o, Agent a,
Message m1) {
    super(a);
    parent = a;
    connection = s;
    input = i;
    output = o;
    m = m1;
}

public Message readMessage(ObjectInputStream input) {
    Message m = null;

        try {
            m = (Message) input.readObject();
        } catch (ClassNotFoundException cnfex) {
            System.out.println(" ** ClassNotFoundException ");
        } catch (IOException cnfex) {
            System.out.println(" ** IOException on port ");
        }
    return m;
}
```

```
/**
 * receiveMessage method comment.
 */
public abstract void receiveMessage(Message m);

/**
 * This method was created in VisualAge.
 */
public abstract void run();

/**
 * This method was created in VisualAge.
 */
public void sendMessage(Message m, ObjectOutputStream output) {
    try {
        m.sender = parent.name;
        m.host = connection.getInetAddress().getLocalHost().getHostName();
        m.port = parent.port;
        output.writeObject((Object) m);
    } catch (IOException e) {
        System.out.println(" ** Exception: " + e);
    }
}
}
```

## Message

```
package afit.mom;

import java.lang.*;
public class Message implements java.io.Serializable {
    public String host = null;
    public int port = 0;

    public String sender = null;
    public String receiver = null;
    public String performative = null;
    public String force = null;
    public String inreplyto = null; // the expected label in a reply
    public String language = null; // name of language used in the content
    public String ontology = null; // name of the ontoloty used in the content
    public String replywith = null; // label user expects for reply
    public Object content = null;

public Message() {
    super();
}

public Object getContent() {
    return content;
}

public String getPerformative() {
    return performative;
}

public String getReceiver() {
    return receiver;
}

public String getSender() {
    return sender;
}

public void setContent(Object cont) {
    content = cont;
}

public void setContent(String cont) {
    content = cont;
}


public void setPerformative(String perf) {
    performative = perf;
}

public void setReceiver(String name) {
    receiver = name;
}

public void setSender(String name) {
    sender = name;
}
}
```

## MessageHandler

```
package afit.mom;

/* agentMOM message handler */

import java.net.*;
import java.io.*;
import java.lang.*;

public class MessageHandler extends Thread {
    public int portNo;
    public ServerSocket server;
    public Agent parent;
    public String endToken = "END";


public MessageHandler(int port, Agent p) {
    super("MessageHandler");
    parent = (Agent) p;

    if (port <= 1024)
        throw new IllegalArgumentException(
        " ** Bad port number given to Listener constructor for agent " + p.name);
    // Try making a ServerSocket to the given port
    try {
        server = new ServerSocket(port);
    } catch (IOException e) {
        System.out.println(" ** Agent " + p.name
                           + " Failed to start ServerSocket on port " + port);
        System.exit(1);
    }
    // Made the connection, so set the local port number
    portNo = port;
    System.out.println("Agent " + p.name
                       + " messageHandler Started on port " + port);
}

/**
 * This method was created in VisualAge.
 */
public void run() {
    Socket connection;
    int counter = 1;
    ObjectOutputStream output;
    ObjectInputStream input;
    System.out.println("Server Running on " + portNo);
    while (true) {
        try {
            connection = server.accept();
            System.out.println("Connection " + counter++ + " received from "
                                + connection.getInetAddress().getHostName());
            output = new ObjectOutputStream(connection.getOutputStream());
            output.flush();
            input = new ObjectInputStream(connection.getInputStream());
            parent.receiveMessage(connection, input, output);
        } catch (IOException cnfex) {
            System.out.println(" ** IOException on port " + portNo);
        }
    }
}
}
```

## Sorry

```
package afit.mom;

/**
 * It defines a general purpose conversation to reply "Sorry"
 * to any unknown type of conversation.
 *
 */
import java.net.*;
import java.io.*;
public class Sorry extends Conversation {
   protected Agent parent; // override parent

/**
 * Server_Register constructor comment.
 */
public Sorry(Socket s, ObjectInputStream i, ObjectOutputStream o,
                 Agent a, Message m1) {
   super(a);
   parent = a;
   connection = s;
   input = i;
   output = o;
   m = m1;
}

/**
 * This method was created in VisualAge.
 */
public void run() {
   int state = 0;
   boolean notDone = true;

   //set up conversation
   try {
      while (notDone) {
         switch (state) {
            case 0 :
               m.setPerformative("sorry");
               m.setContent("unknown conversation request");
               sendMessage(m, output);
               notDone = false;
               break;
         }
      }
      input.close();
      output.close();
      connection.close();
   } catch (UnknownHostException e) {
      System.out.println(" ** Unknown Host exception.");
   } catch (EOFException oef) {
      System.out.println(" ** Server terminated connection.");
   } catch (IOException e) {
      System.out.println(" ** IO Exception.");
      e.printStackTrace();
   }
}
}
```

# Appendix B

agentMom Example Source Code

## ClientAgent

```
package afit.momtest;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import afit.mom.*;
/**
 * This type was created in VisualAge.
 */
public class ClientAgent extends Agent implements ActionListener {
    private TextArea theText;
    private Panel textPanel;
    private TextField Tname;
    public String serverHost;
    public int serverPort;

/**
 * ClientAgent constructor comment.
 */
public ClientAgent(String agentName, int agentPort,
    String sHost, int sPort) {
    super(agentName, agentPort);

    this.serverHost = sHost; // the Host to connect to
    this.serverPort = sPort; // the Port to connect to

    /* server initialization */
    MessageHandler h = new MessageHandler(this.port, this);
    h.start();

    /* window initialization */
    setSize(400, 300); // set default size & height of new window

    addWindowListener(new WindowDestroyer());
    setTitle(this.name); // name the window
    setBackground(Color.red);
    setLayout(new BorderLayout());

    // text panel
    textPanel = new Panel();
    theText = new TextArea(12, 50);
    theText.setBackground(Color.white);
    theText.setForeground(Color.black);
    textPanel.add(theText);
    add(textPanel, "Center");

    // top label
    Panel namePanel = new Panel();
    namePanel.setForeground(Color.white);
```

```
    Label nameLabel = new Label("Enter text of your choice: ");
    namePanel.add(nameLabel, "Center");
    add(namePanel, "North");
}

/**
 * This method was created in VisualAge.
*/
public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("Exit")) {
        System.exit(0);
    } else
        theText.setText("Error in memo interface!");
    textPanel.repaint();
}

/**
 * This method was created in VisualAge.
*/
public String get_service() {
    if (this.name.equals("Client-1"))
        return "Laundry";
    if (this.name.equals("Client-2"))
        return "Dry Cleaning";
    if (this.name.equals("Client-3"))
        return "Drill Press Operator";
    else
        return "Unknown";
}

/**
 * This method was created in VisualAge.
*/
public static void main(String[] args) {
    String serverHost;
    int serverPort = 5000;

    try {
        serverHost = InetAddress.getLocalHost().getHostName();
        ClientAgent s =
            new ClientAgent("Client", 3000, serverHost, serverPort);
        s.run();
    } catch (UnknownHostException e) {
        System.out.println(" ** Host Exception when starting ClientAgent");
    }
}

/**
 * receiveMessage method comment.
 */
public void receiveMessage(Socket server, ObjectInputStream input,
                           ObjectOutputStream output) {
    int i;
    Message m;
    Server_Register s;
    Thread t;

    /* define all possible conversations receivable here */
    try {
        m = (Message) input.readObject();
        System.out.println(
            " ** Invalid Attempt to start new conversation with performative "
            + m.performative + " from " + m.sender);
```

```
            t = new Thread(new Sorry(server, input, output, this, m));
            t.start(); // start new thread
        } catch (ClassNotFoundException cnfex) {
            System.out.println(" ** ClassNotFoundException ");
        } catch (IOException cnfex) {
            System.out.println(" ** IOException on port (ServerAgent.receiveMessage)");
        }
    }

    /**
     * This method was created in VisualAge.
    */
    public void run() {
        Thread init;
        theText.append("Starting Client \n");
        setVisible(true);

        init = new Thread(new Client_Register(this, serverHost, serverPort,
                                get_service())); // start conversation
        init.start();

        int j;
        for (int i = 0; i < 10000000; i++)
            j = i - 1;
        init = new Thread(new Client_Unregister(this, serverHost, serverPort,
                                get_service())); // start conversation
        init.start();
    }

    /**
     * This method was created in VisualAge.
    */
    protected void write(String s) {
        this.theText.append(s + "\n");
    }
    }
```

## ServerAgent

```
package afit.momtest;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import afit.mom.*;

/**
 * This type was created in VisualAge.
 */
public class ServerAgent extends Agent implements ActionListener {
    private TextArea theText;
    private Panel textPanel;
    private TextField Tname;

    private Vector registrants = new Vector(25);

/**
 * ServerAgent constructor comment.
 */
public ServerAgent(String n, int p) {
    super(n, p);
    MessageHandler handler;          /* server initialization */
    handler = new MessageHandler(port, this);
    handler.start();

    /* window initialization */
    setSize(500, 300); // set default size & height of new window

    addWindowListener(new WindowDestroyer()); // create listener
    setTitle("Server");   // name the window
    setBackground(Color.blue);
    setLayout(new BorderLayout());

    // text panel
    textPanel = new Panel();
    theText = new TextArea(12, 60);
    theText.setBackground(Color.white);
    theText.setForeground(Color.black);
    textPanel.add(theText);
    add(textPanel, "Center");

    // top label
    Panel namePanel = new Panel();
    namePanel.setForeground(Color.white);
    Label nameLabel = new Label("Enter text of your choice: ");
    namePanel.add(nameLabel, "Center");
    add(namePanel, "North");

    setVisible(true);
}




/**
```

```
 * This method was created in VisualAge.
*/
public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("Exit")) {
        System.exit(0);
    } else
        theText.setText("Error in memo interface!");
    textPanel.repaint();
}

/**
 * This method was created in VisualAge.
*/
protected void add_service(String agent, String service, String host, int port) {
    this.registrants.addElement(new Registration(agent, host, port, service));
}

/**
 * This method was created in VisualAge.
*/
public static void main(String[] args) {
    ServerAgent s = new ServerAgent("Server", 5000);
    s.run();
}

/**
 * This method was created in VisualAge.
 */
protected void printRegistrants() {
    java.util.Enumeration enum = this.registrants.elements();
    Registration v;
    enum = this.registrants.elements();
    while (enum.hasMoreElements()) {
        v = (Registration) enum.nextElement();
        this.theText.append("< " + v.agent + ", " + v.service + ", " + v.host + ", "
                            + v.port + "> \n");
    }
}

/**
 * receiveMessage method comment.
 */

public void receiveMessage(Socket server, ObjectInputStream input,
                           ObjectOutputStream output) {
    int i;
    Message m;
    Server_Register s;
    Thread t;

    /* define all possible conversations receivable here */
    try {
        m = (Message) input.readObject();
        this.theText.append("Received message " + m.performative
                               + " from "+m.sender+"\n");
        if (m.performative.equals("register")) {
            t = new Thread(new Server_Register(server, input, output, this, m));
            t.start(); // start new thread
        } else
            if (m.performative.equals("unregister")) {
                t = new Thread(new Server_Unregister(server, input, output,
                                                     this, m));
                t.start(); // start new thread
```

```
            } else {
               System.out.println(
                   " ** Invalid Attempt to start new conversation with performative "
                   + m.performative + " from " + m.sender);
               t = new Thread(new Sorry(server, input, output, this, m));
               t.start(); // start new thread
            }
      } catch (ClassNotFoundException cnfex) {
         System.out.println(" ** ClassNotFoundException ");
      } catch (IOException cnfex) {
         System.out.println(" ** IOException on port (ServerAgent.receiveMessage)");
      }
   }

   /**
    * This method was created in VisualAge.
   */
   protected void remove_service(String agent, String service,
                                    String host, int port) {
      java.util.Enumeration enum = this.registrants.elements();
      Registration v;
      while (enum.hasMoreElements()) {
         v = (Registration) enum.nextElement();
         if (v.agent.equals(agent) && v.service.equals(service)
                && v.host.equals(host) && v.port == port)
            this.registrants.removeElement(v);
      }
   }

   /**
    * This method was created in VisualAge.
   */
   public void run() {
      theText.append("Starting Server \n");
      setVisible(true);
   }

   /**
    * This method was created in VisualAge.
   */
   protected void write(String s) {
      this.theText.append(s + "\n");
   }
   }
```

## Client_Register

```
package afit.momtest;

/**
 * This type was created in VisualAge.
 */
import java.net.*;
import java.io.*;
import afit.mom.*;
public class Client_Register extends Conversation {
    ClientAgent parent; // override parent
    String service;

public Client_Register(ClientAgent a, String hostName,
    int portNum, String theService) {
    super(a, hostName, portNum);
    parent = a;
    service = theService;
}

public void run() {
    Message m = new Message();
    int state = 0;
    boolean notDone = true;
    parent.write("Starting Client_Register conversation.");

    //set up conversation
    try {
        connection = new Socket(connectionHost, connectionPort);
        output = new ObjectOutputStream(connection.getOutputStream());
        output.flush();
        input = new ObjectInputStream(connection.getInputStream());
        while (notDone) {
            switch (state) {
                case 0 :
                    m.performative = "register";
                    m.content = service;
                    sendMessage(m, output);
                    state = 1;
                    break;
                case 1 :
                    m = readMessage(input);
                    if (m.performative.equals("reply"))
                        notDone = false;
                    else
                        parent.write("** ERROR - did not get reply back **");
                    break;
            }
        }
        input.close();
        output.close();
        connection.close();
    }  catch (UnknownHostException e) {
        parent.write("** Unknown Host exception in Client_Register.");
    } catch (EOFException oef) {
        parent.write("** Server terminated connection in Client_Register.");
    } catch (IOException e) {
        parent.write("** IO Exception in Client_Register.");
    }
}
}
```

## Client_Unregister

```
package afit.momtest;

/**
 * This type was created in VisualAge.
 */
import java.net.*;
import java.io.*;
import afit.mom.*;
public class Client_Unregister extends Conversation {
    ClientAgent parent; // override parent
    String service;

public Client_Unregister(ClientAgent a, String hostName,
    int portNum, String theService) {
    super(a, hostName, portNum);
    parent = a;
    service = theService;
}

public void run() {
    Message m = new Message();
    int state = 0;
    boolean notDone = true;
    parent.write("Starting Client_Unregister conversation.");
    //set up conversation
    try {
        connection = new Socket(connectionHost, connectionPort);
        output = new ObjectOutputStream(connection.getOutputStream());
        output.flush();
        input = new ObjectInputStream(connection.getInputStream());
        while (notDone) {
            switch (state) {
                case 0 :
                    m.setPerformative("unregister");
                    m.content = service;
                    sendMessage(m, output);
                    state = 1;
                    break;
                case 1 :
                    m = readMessage(input);
                    if (m.performative.equals("reply"))
                        notDone = false;
                    else
                        parent.write("** ERROR - did not get reply back ** \n");
                    break;
            }
        }
        input.close();
        output.close();
        connection.close();
    } catch (UnknownHostException e) {
        parent.write("** Unknown Host exception in Client_Unregister.");
    } catch (EOFException oef) {
        parent.write("** Server terminated connection in Client_Unregister.");
    } catch (IOException e) {
        parent.write("** IO Exception in Client_Unregister.");
        e.printStackTrace();
    }
}
}
```

## Server_Register

```
package afit.momtest;

/**
 * This type was created in VisualAge.
 */
import java.net.*;
import java.io.*;
import afit.mom.*;
public class Server_Register extends Conversation {
    ServerAgent parent; // override parent

public Server_Register(Socket s, ObjectInputStream i, ObjectOutputStream o,
ServerAgent a, Message m1) {
    super(s, i, o, a, m1);
    parent = a;
}

public void run() {
    int state = 0;
    boolean notDone = true;
    parent.write("Got >>" + m.getPerformative() + " - " + m.getContent()
                    + " from " + m.getSender());
    //set up conversation
    try {
        while (notDone) {
            switch (state) {
                case 0 :
                    parent.add_service(m.sender, (String) m.content, m.host, m.port);
                    parent.printRegistrants();
                    state = 1;
                    break;
                case 1 :
                    m.setPerformative("reply");
                    m.setContent("OK");
                    sendMessage(m, output);
                    notDone = false;
            }
        }
        input.close();
        output.close();
        connection.close();
    } catch (UnknownHostException e) {
        parent.write("** Unknown Host exception.");
    } catch (EOFException oef) {
        parent.write("** Server terminated connection.");
    } catch (IOException e) {
        parent.write("** IO Exception. (Server_Register)");
        e.printStackTrace();
    }
}
}
```

## Server_Unregister

```
package afit.momtest;

/**
 * This type was created in VisualAge.
 */
import java.net.*;
import java.io.*;
import afit.mom.*;
public class Server_Unregister extends Conversation {
    ServerAgent parent; // override parent

public Server_Unregister(Socket s, ObjectInputStream i, ObjectOutputStream o,
ServerAgent a, Message m1) {
    super(s, i, o, a, m1);
    parent = a;
}

public void run() {
    int state = 0;
    boolean notDone = true;
    parent.write("Got >>" + m.getPerformative() + " - " + m.getContent()
                    + " from " + m.getSender());

    //set up conversation
    try {
        while (notDone) {
            switch (state) {
                case 0 :
                    state = 1;
                    break;
                case 1 :
                    parent.remove_service(m.sender, (String) m.content, m.host, m.port);
                    parent.printRegistrants();
                    state = 2;
                    break;
                case 2 :
                    m.setPerformative("reply");
                    m.setContent("OK");
                    sendMessage(m, output);
                    notDone = false;
                    break;
            }
        }
        input.close();
        output.close();
        connection.close();
    } catch (UnknownHostException e) {
        parent.write("** Unknown Host exception.");
    } catch (EOFException oef) {
        parent.write("** Server terminated connection.");
    } catch (IOException e) {
        parent.write("** IO Exception.");
        e.printStackTrace();
    }
}
}
```

## Registration

```
package afit.momtest;

/**
 * This type was created in VisualAge.
 */
public class Registration {
    String agent;
    String host;
    int port;
    String service;

/**
 * Registration constructor comment.
 */
public Registration(String a, String h, int p, String ser) {
    agent = a;
    host = h;
    port = p;
    service = ser;
}
}
```

## Tester

```
package afit.momtest;

/**
 * This type was created in VisualAge.
 */
import java.net.*;
public class Tester {
/**
 * Tester constructor comment.
 */
public Tester() {
    super();
}
/**
 * Starts the application.
 */
public static void main(java.lang.String[] args) {
    int NUMCLIENTS = 3;
    int cPort = 3000; // starting port address
    int sPort = 5000;
    String host;
    Thread server;
    Thread clients[] = new Thread[NUMCLIENTS];
    //

    try {
        host = InetAddress.getLocalHost().getHostName();

        System.out.println("Starting Server Agent on host " + host);
        server = new Thread(new ServerAgent("Server", sPort), "Server");
        server.start();

        for (int i = 0; i < clients.length; i++) {
            System.out.println("Starting Client Agent" + (i + 1));
            clients[i] = new Thread(new ClientAgent("Client-" + (i + 1),
                            cPort + i, host, sPort), "Client-" + (i + 1));
            clients[i].start();
        }
        while (true);
    } catch (UnknownHostException e) {
        System.out.println(" ** Unknown Host Exception when starting ClientAgent");
    }
}
}
```

## WindowDestroyer

```
package afit.momtest;

import java.awt.*;
import java.awt.event.*;
/**
 * This type was created in VisualAge.
 */
public class WindowDestroyer extends WindowAdapter  {
/**
 * This method was created in VisualAge.
 */
public void windowClosing(WindowEvent e)  {
    System.exit(0);
}
}
```