Objective Viewpoint

A Simple SMTP Framework for Java

by *George Crawford III*

Introduction

In this installation of Objective Viewpoint, we'll examine the implementation of a Simple Mail Transfer Protocol (SMTP) client-side framework that can be used in Java applications and applets. This framework demonstrates the effective use of Java network communication.

Background

SMTP is part of the SMTP protocol suite that specifies a format for mail message exchange between two machines. An SMTP mail transaction is a fairly straight-forward process. The SMTP client establishes a connection with an SMTP server on port 25 of the SMTP host and waits for the server to respond with a **220 READY FOR MAIL** message. The client is then ready to construct a mail message. A mail message is constructed by first sending the **MAIL**command. The **MAIL** command is structured as follows:

```
MAIL FROM: <sender>
```

where is the e-mail address of the message transmitter. For example, the command

```
MAIL FROM: gec2@ra.msstate.edu
```

indicates to the server that the source of this message is gec2@ra.msstate.edu. Recipients of the message are specified by using the one or more **RCPT** commands. This command has the format:

```
RCPT TO: <recipient>
```

where is the target of the e-mail message to be sent. After an acknowledgement has been received for each **RCPT** command, and a sender has been identified, the client is able to enter the contents of the message using the **DATA** command. The **DATA** command is entered on a line by itself without any fields. The server responds to the client with a **354 Start Mail Input** message. Data can also contain memo header fields such as Date, Subject, To, From, and Cc. Data entry is terminated by entering a single period on a line by itself. When the message body is terminated, a **250 OK** message is sent to the client. The client can then end the session by sending the **QUIT** command, or can continue the process and construct a new message [2].

Figure 2-1 illustrates a typical SMTP connection and data exchange session. Upon the establishment of the connection with the SMTP server on port 25 of Ra.MsState.Edu, the server sends a **220 Service Ready** greeting. The client then

specifies the source of the message using the **MAIL** command. In this example, the source specified is gec2@Ra.MsState.Edu. The recipients are then identified using the **RCPT** command. The message body is entered by the client through the **DATA** command. As shown, we also include a **Subject** memo header. The message body is terminated by the client with a single period on a line by itself. The server responds with a **250 OK** message. Finally, the client closes the connection with by sending the **QUIT** command. This command is acknowledged by the server with a **221 Closing Connection** message. In figure 2-1, the client and server transactions are indicated by **C:** and **S:** respectively.

```
S: 220 Ra.MsState.Edu ESMTP Sendmail 8.8.8/8.8.7/MsState-Ra/1.8; Fri, 2 Jan 1998
15:29:50 -0600 (CST)
C: MAIL FROM: gec2@ra.msstate.edu
S: 250 gec2@ra.msstate.edu... Sender ok
C: RCPT TO: crawford@cs.msstate.edu
S: 250 crawford@cs.msstate.edu... Recipient ok
C: RCPT TO: george@ditch.erc.msstate.edu
S: 250 george@ditch.erc.msstate.edu... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Subject: Hi
C: Hi from Ra.MsState.edu.
C:
C: Bye.
C: .
S: 250 PAA29987 Message accepted for delivery
C: quit
S: 221 Ra.MsState.Edu closing connection
```

Figure 2-1: A typical SMTP session.

The Java SMTP Framework

The Java SMTP framework consists of three classes: MailMessage, EmailAddress, and SMTPMailExchanger. The following sections describe each component of the framework.

EmailAddress

The **EmailAddress** class simply contains the user I.D. and the domain name of an email address [1]. **EmailAddress** objects are created by the programmer to specify e-mail source and recipients.

The domain name part of an **EmailAddress** is an instance of **InetAddress**. The **InetAddress** is a standard component of the Java class library, located in the **java.net** package, and represents a valid Internet Protocol (IP) address. **InetAddress** objects are used throughout the **java.net** hierarchy.

Users do not instantiate an **InetAddress** object directly. Instead, an **InetAddress** object is obtained by the user through one of the following static **InetAddress** methods: **getByName()**, **getAllByName()**, and **getLocalHost**(). The first method, **getByName()**, returns an **InetAddress** object that represents the IP address of the specified host. The host name is specified using a **String** and can be represented either by the machine name or dotted decimal notation. The method **getAllByName()** returns an array of **InetAddress** objects that represent all the IP addresses of the specified

host. An invalid host name in either method results in a **java.net.UnknownHostException** being thrown. The method **getLocalHost()** returns the IP address of the local host. The **java.net.UnknownHostException** is thrown if no IP address for the host can be found [3].

The **EmailAddress** class is shown in figure 3-1.

```
 1: /**
 2:  * EmailAddress.java
 3:  *
 4:  * @author George Crawford III
 5:  * @version 0.1 12/23/97
 6:  */
 7: import java.net.InetAddress;
 8: import java.net.UnknownHostException;
 9:
10: public class EmailAddress {
11:     /**
12:      * Constructs a new EmailAddress with the
13:      * specified user I.D. and domain address.
14:      *
15:      * @param userID user I.D. portion of the email address.
16:      * @param domainAddress domain address portion of the email address.
17:      * @exception NullPointerException if the user I.D. or
18:      *     domain name are null.
19:      */
20:     public EmailAddress(String userID, String address)
21:         throws NullPointerException, UnknownHostException {
22:           if(userID == null || address == null)
23:             throw new NullPointerException("EmailAddress");
24:             this.userID = userID;
25:             domainAddress = InetAddress.getByName(address);
26:       }
27:     /**
28:      * Returns the domain address portion of this email address.
29:      *
30:      * @return the domain address for this email address.
31:      */
32:       public InetAddress getDomainAddress() {
33:         return domainAddress;
34:     }
35:     /**
36:      * Return the user I.D. portion of this email address.
37:      *
38:      * @return the user I.D. for this email address.
39:      */
40:     public String getUserID() {
41:         return userID;
42:     }
43:     /**
```

```
44:      * Return the domain address portion of this email address.
45:      *
46:      * @return the domain address for this email address.
47:      */
48:     public void setDomainAddress(String address)
49:         throws UnknownHostException {
50:         if(address != null) {
51:               domainAddress = InetAddress.getByName(address);
52:           }
53:     }
54:     /**
55:      *
56:     public void setUserID(String userID) {
57:     if(userID != null)
58:         this.userID = userID;
59:     }
60:     /**
61:      * Checks that obj is an EmailAddress and
62:      * has the same user I.D. and domain address as this
63:      * EmailAddress.
64:      *
65:      * @param obj the object we are testing for equality with this object.
66:      * @return true if obj is an instance of
67:      *     EmailAddress and has the same user I.D. and domain
68:      *     address as this EmailAddress.
69:      */
70:     public boolean equals(Object obj) {
71:         if(obj != null)
72:             if(obj instanceof EmailAddress) {
73:                   EmailAddress address = (EmailAddress)obj;
74:                   if(address.userID.equalsIgnoreCase(userID) &&
75:                     address.domainAddress.equals(domainAddress))
76:                       return true;
77:             }
78:         return false;
79:     }
80:     private InetAddress domainAddress;
81:     private String userID;
82: }
```

Figure 3-1: The EmailAddress Class.

MailMessage

The **MailMessage** class is very simple and consists of the following attributes: the e-mail address of the sender, a list of email addresses for the message recipients, the subject of the message, and the text body of the message. The e-mail address for the sender and each recipient is an instance of **EmailAddress**, discussed in the previous section.

Figure 3-2 shows the code for the **MailMessage** class.

```
 1: /**
 2:  * MailMessage.java
 3:  *
 4:  * @author George Crawford III
 5:  * @version 0.1 12/23/97
 6:  */
 7:
 8: import java.util.Enumeration;
 9: import java.util.Vector;
10:
11: public class MailMessage {
12:     /**
13:      * Creates a new email message with the specified sender and recipient.
14:      *
15:      * @param sender source of this email message.
16:      * @param recipient receiver of this email message.
17:      * @exception NullPointerException if either the sender
18:      *     or receiver is null.
19:      */
20:     MailMessage(EmailAddress sender, EmailAddress recipient)
21:         throws NullPointerException {
22:         if(sender == null || recipient == null) {
23:         throw new NullPointerException("MailMessage");
24:         }
25:         this.sender = sender;
26:         recipients.addElement(recipient);
27:     }
28:     /**
29:      * Adds another recipient to the list of recipients for this message.
30:      *
31:      * @param recipient receiver of this message.
32:      */
33:     public void addRecipient(EmailAddress recipient) {
34:         if(recipient != null)
35:             if(!recipients.contains(recipient))
36:                 recipients.addElement(recipient);
37:     }
38:     /**
39:     * Returns the text data contained in this MailMessage.
40:     *
41:     * @return the text data contained in this MailMessage.
42:     */
43:     public String getData() {
44:         return data;
45:     }
46:     /**
47:      * Returns an array of recipients.
48:      *
49:      * @return an array of EmailAddresses of the recipients of
```

```
 50:      *     this message.
 51:      */
 52:     public EmailAddress[] getRecipients() {
 53:         EmailAddress[] emailAddresses = new EmailAddress[recipients.size()];
 54:         int index = 0;
 56:           for(Enumeration e = recipients.elements(); e.hasMoreElements(); )
 57:               emailAddresses[index++] = (EmailAddress)e.nextElement();
 58:           return emailAddresses;
 59:     }
 60:     /**
 61:      * Returns the EmailAddress of the sender.
 62:      *
 63:      * @return the EmailAddress of the sender of this message.
 64:      */
 65:     public EmailAddress getSender() {
 66:         return sender;
 67:     }
 68:     /**
 69:      * Returns the subject of this MailMessage.
 70:      *
 71:      * @return the subject of this MailMessage
 72:      */
 73:     public String getSubject() {
 74:         return subject;
 75:     }
 76:     /**
 77:      * Sets the text data contained in this MailMessage to the
 78:      * specified string.
 79:      *
 80:      * @param data the text data that is to be included in this
 81:      *     MailMessage.
 82:      */
 83:     public void setData(String data) {
 84:         this.data = data;
 85:     }
 86:     /**
 87:      * Sets the address of the sender of this MailMessage.
 88:      *
 89:      * @param sender the address of the sender of this
 90:      * Email Message.
 91:      */
 92:     public void setSender(EmailAddress sender) {
 93:         if(sender != null)
 94:             this.sender = sender;
 95:     }
 96:     /**
 97:      * Sets the subject of this MailMessage.
 98:      *
 99:      * @param the subject of this MailMessage.
100:      */
```

```
101:     public void setSubject(String subject) {
102:         if(subject != null)
103:             this.subject = subject;
104:     }
105:     private EmailAddress sender;
106:     private String subject = "";
107:     private String data = "";
108:     private Vector recipients = new Vector(5);
109: }
```

Figure 3-2 The MailMessage Class.

SMTPMailExchanger

The **SMTPMailExchanger** class is the sole enabler for email exchange via SMTP.

The **SMTPMailExchanger** has three methods. The first method is a constructor method that requires the name of a host as parameter. In lines 28-31, we spawn a new **java.net.Socket** with the specified SMTP host name on port 25. A socket is a communication abstraction that allows an application to access the TCP/IP protocols on a given host [1]. We next obtain the input and output streams associated with the socket. For our purposes, we will use the **java.net. BufferedReader** and **java.net.DataOutputStream** classes for reading from and writing to the server, respectively. The **BufferedReader** class expects as a parameter to its constructor method an instance of **java.net.InputStream**. Likewise, the **DataOutputStream** class requires an instance of **java.net.OutputStream**. We obtain the input and output streams for a socket by calling the respective **getInputStream()** and **getOutputStream()** methods of the **Socket** instance.

```
 1: /**
 2:  * SMTPMailExchanger.java
 3:  *
 4:  * @author George Crawford III
 5:  * @version 0.1 12/26/97
 6:  */
 7: import java.io.BufferedReader;
 8: import java.io.DataOutputStream;
 9: import java.io.IOException;
10: import java.io.InputStreamReader;
11: import java.net.InetAddress;
12: import java.net.Socket;
13: import java.net.UnknownHostException;
14: public class SMTPMailExchanger {
15:     /**
16:      * Constructs a new SMTPMailExchanger
17:      * with the specified internet address.
18:      *
19:      * @param address the address of the SMTP server to connect to.
20:      * @exception NullPointerException if address is
21:      * null.
22:      */
23:     public SMTPMailExchanger(String hostName)
24:         throws NullPointerException, IOException, UnknownHostException {
```

```
25:      if(hostName == null)
26:             throw new NullPointerException("SMTPMailExchanger");
27:         this.address = InetAddress.getByName(hostName);
28:      socket = new Socket(this.address, 25);
29:         in = new BufferedReader(
30:             new InputStreamReader(socket.getInputStream()));
31:         out = new DataOutputStream(socket.getOutputStream());
32:     }
33:     public void send(MailMessage message) {
34:         EmailAddress sender = message.getSender();
35:      EmailAddress[] recipients = message.getRecipients();
36:      try {
37:     System.out.println(in.readLine());
38:             out.writeBytes("mail from: "+sender.getUserID());
39:          out.writeBytes("@"+sender.getDomainAddress()+"\n");
40:    System.out.println(in.readLine());
41:             for(int i = 0; i < recipients.length; i++) {
42:                 out.writeBytes("rcpt to: "+recipients[i].getUserID());
43:                 out.writeBytes("@"+recipients[i].getDomainAddress()+"\n");
44:        System.out.println(in.readLine());
45:    }
46:    out.writeBytes("data\n");
47:    out.writeBytes("Subject: "+message.getSubject());
48:    out.writeBytes(message.getData());
49:    out.writeBytes("\n.\n");
50:    System.out.println(in.readLine());
51:    System.out.println(in.readLine());
52:    out.writeBytes("quit\n");
53:         } catch(IOException ioe) {
54:          System.err.println(ioe);
55:      }
56:     }
57:     /**
58:      * Closes the connection with the SMTP server.
59:      */
60:     public void finalize() {
61:      try {
62:             socket.close();
63:      } catch(IOException ioe) {
64:          System.err.println(ioe);
65:      }
66:     }
67:     private BufferedReader   in;
68:     private DataOutputStream out;
69:     private Socket socket;
70:     private InetAddress address;
71: }
```

Figure 3-3: The SMTPMailExchanger Class.

The actual communication takes place in the **send()** method, lines 33-56. The parameter to this method is a **MailMessage**. In lines 34 and 35, we retrieve the **EmailAddress** for the sender and each recipient.

Next, we retrieve the server response to the connection established in line 28 (you can optionally move this line into the constructor method). For illustrative purposes, all server responses are dumped to the standard output stream. In an actual application, you will probably want to remove the **System.out.println** methods (but leave the **in.readLine()** methods in place). In lines 38 and 39, we identify the sender of the message to the SMTP server. The server response is captured in line 40. The message recipients are specified in lines 41-45 by iterating through the array of **EmailAddress** objects retrieved in line 35. The message content is transferred in lines 46-49. Notice in line 47 that we specify the subject of the message, after the SMTP **DATA** command. The message is terminated in line 49 and the session is closed with the SMTP **QUIT** command in line 52.

4. Example

The program shown in Figure 4-1 demonstrates how to use **SMTPMailExchanger**. First, several instances of class **EmailAddress** are defined. The first represents the sender of the messages, followed by recipients. Next we attempt to open a connection to the SMTP server by creating an instance of **SMTPMailExchanger** and passing the name of the SMTP server as a paramter to the constructor. If the connection is successful, we create a message and add all recipients beyond primary recipient, set the subject title, and specify the data to send. That's all there is to it!

```
 1: /**
 2:  * MailTest.java
 3:  *
 4:  * @author George Crawford III
 5:  * @version 0.1 12/27/97
 6:  */
 7:
 8: import java.io.*;
 9: import java.net.*;
10:
11: public class MailTest {
12:     public static void main(String[] args) {
13:         try {
14:            EmailAddress user1 = new EmailAddress("gec2", "ra.msstate.edu");
15:            EmailAddress recp1 = new EmailAddress("crawford",
16:                     "cs.msstate.edu");
17:            EmailAddress recp2 = new EmailAddress("george",erc.msstate.edu");
18:            MailMessage message = new MailMessage(user1, recp1);
19:             try {
20:                     SMTPMailExchanger exchanger =
21:                     new SMTPMailExchanger("ra.msstate.edu");
22:                     message.addRecipient(recp2);
23:                     message.setSubject("Hello");
24:                message.setData("hey!!!\n");
25:                exchanger.send(message);
26:              } catch(IOException ioe) {
27:                  System.err.println(ioe);
28:              }
```

```
29:                 System.out.println("Message Subject: "+message.getSubject());
30:                 System.out.println("Message Data: "+message.getData());
31:             } catch(UnknownHostException uhe) {
32:                 System.err.println(uhe);
33:             }
34:         }
35: }
```

References

1

Comer, Douglas E. Internetworking with TCP/IP Volume 1: Principles, Protocols, and Architectures , Prentice Hall, Englewood Cliffs, New Jersey, 1995.

2

Postel, Jonathan B. RFC 821: Simple Mail Transfer Protocol, http://www.kisco.co.kr/~hollobit/RFC/rfc/rfc821.html, August 1982.

3

Sun Microsystems, Java™ Platform 1.2 Beta 2 API Specification, 17 December 1997.

Biography

George Crawford III is a software engineer at MPI Software Technology, Inc. and is pursuing his M.S. degree in computer science at Mississippi State University. His technical areas of interest include software design, software process management, distributed object technology, and games programming. He has been using Java since its alpha release in 1995.