computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

WEB ACCESSIBLE MULTIACCOUNT CONFIGURABLE CLIENT EMAIL APPLICATION

Krzysztof Ratecki, Bartosz Sakowicz, Marcin Wójtowski, Andrzej Napieralski

Department of Microelectronics and Computer Science,
al. Politechniki 11, 90-924 Lodz, POLAND
icek@elek.osemka.p.lodz.pl, sakowicz@dmcs.pl, mw@dmcs.pl, napier@dmcs.pl,
http://www.dmcs.pl

Abstract: *The article presents alternative ways of checking emails and how to make an application mail enabled using library of J2EE Platform – JavaMail API. The article describes application attainable from WEB browser based on JSP Scripts and Java servlets. Application allows for access to multiple email accounts through one web page.*

Keywords: *J2EE, JAVA, JAVAMAIL API, EMAIL.*

1. INTRODUCTION

In the late 60's, Ray Tomlinson contributed his research to the invention of a service we know as an electronic mail (e-mail). Nowadays a billion of people from whole world send emails to each other every day. Messages contain not only a plain text, as it was at the beginning, but include images, attachments, documents etc.

The article presents an application, which can access to email box fast and easy from every terminal equipped with web browser.

The JavaMail API provides a set of abstract classes defining objects that comprise a mail system [2]. The API defines classes like Message, Store and Transport. The API can be extended and can be subclassed to provide new protocols and to add functionality when necessary. In addition, the API provides concrete subclasses of the abstract classes. These subclasses, including MimeMessage and MimeBodyPart, implement widely used Internet mail protocols and conform to specifications RFC822 and RFC2045 [1]. They are ready to be used in application development with standards and protocols:

SMTP (Simple Mail Transfer Protocol) is a simple protocol used to send messages through email server to advisable address.

POP (Post Office Protocol) and IMAP (Internet Message Access Protocol) are two various protocol used to receive messages from server. The difference is in what way the messages are managed on the server. POP works with emails received from server. It is a popular protocol but its abilities are limited. IMAP allows creating folders, marking messages, searching for, sending and managing the mailbox directly on server. Because of that, protocol cause more traffic and server load than POP.

MIME (Multipurpose Internet Mail Extensions) is standard for attachments and non-ASCII text in emails. Because POP and SMTP allows for MIME-formatted email, all Internet email comes MIME-formatted so POP clients should also understand and use MIME. IMAP, by design, assumes MIME-formatted email [8].

Developing a tool to deal with e-mail called E-mail Client, which sends and receives text mail as well as receives and sends attachments, works both plain and HTML mode has become the main objective of this article.

In major requirements this client should be accessible from web browser and should have features similar to e-business application (secure, fast, reliable, user friendly). Additionally it differs from existing applications in the way that it allows for accessing any mail account, not only related to the server on which the client is deployed.

2. APPLICATION ARCHITECTURE

Application is based on J2EE platform. It consists of two layers (Fig.1).

- Communication layer, which is an interface between user and core engine layer. Communication layer has been built from JSP scripts and Java servlets
- Core engine layer is based on JavaBeans, which use ready to run tools to interact with

remote email box and to exchange data with database. This database stores data needed to login and basic but necessary information to establish connection with remote box.

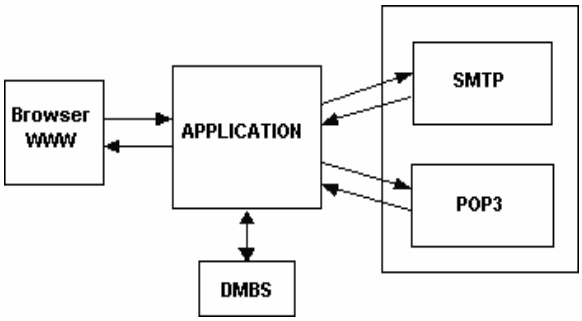

Fig. 1 - Overview of application

3. APPLICATION STRUCTURE

The application consists of JSP scripts and Java servlets. JSP scripts are used as forms; servlets are responsible for data processing and interact with mailbox (Fig.2). All needed information about users and configuration of application are stored in MySQL database. There are items such as: login and password to access the application, information necessary to check and send messages, role of the user (administrator or regular user).

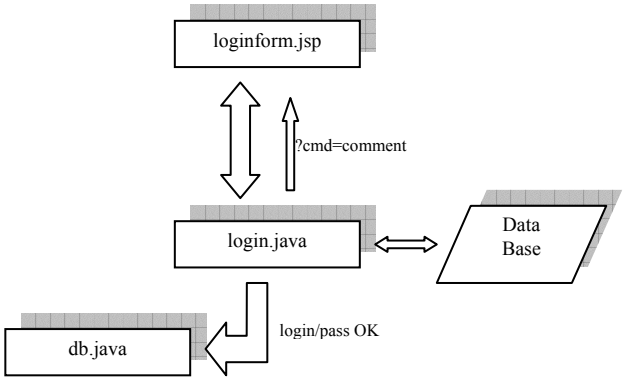

Fig. 2 - Login process

Servlet login.java takes data from application form loginform.jsp(Fig. 2):

```
String login=
request.getParameter("user");
String
pass=request.getParameter("pass");
```

and next using JNDI Resources checks user and password in database (Fig. 3) [5].

If login process was successful servlet db.java reads information about user stored in database and controlling is moved to main controller of application main.java (Fig. 4).

```
Connection connection = null;
Context initctx = new
   InitialContext();
Context envCtx = (Context)
   initctx.lookup("java:comp/env");
DataSource ds =
   (DataSource)envCtx.lookup("jdbc/
   AccountsDB");
connection = ds.getConnection();
Statement statement =
   connection.createStatement();
ResultSet rs = null;
SQLQuery = "SELECT * FROM javamail
   WHERE login ='"+login+"';";
rs =
   statement.executeQuery(SQLQuery)
   ;
while(rs.next()){
pwd = rs.getString("pass");
        }
connection.close();
       }
```

Fig. 3 - Part of login.java file

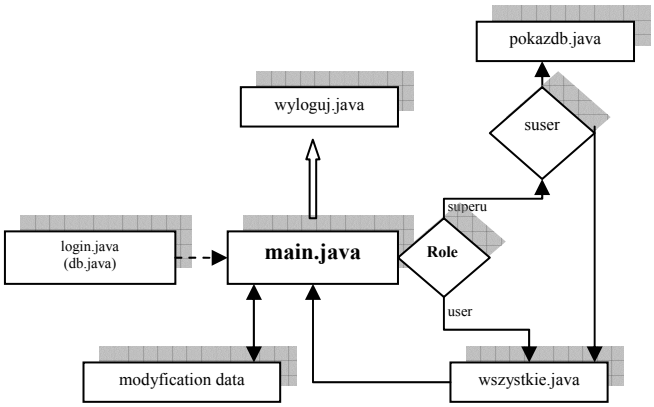

Fig. 4 - Main controller of application

4. SERVLETS CONTAINER

Tomcat implements the servlet and the Java Server Pages (JSP) specifications from Sun Microsystems [5]. That is to say, it provides an environment for Java code to run in cooperation with a web server. Tomcat is a web server that supports servlets and JSPs. Tomcat comes with the Jasper compiler that compiles JSPs into servlets.

The Tomcat servlet engine often appears in combination with an Apache web server or other web servers. Tomcat can also function as an independent web server in it. Earlier in its development, the perception existed that standalone Tomcat was only suitable for development environments and other environments with minimal requirements for speed and transaction handling. However, that perception is no longer true. Tomcat is increasingly being used as a standalone web

server in high-traffic, high-availability environments.

The Tomcat server plays in described email client application three major functions. First of all makes the application accessible for registered users, second makes secure connection between client and application using SSL and finally supports communication with database server based on MySQL [7,9]. Configurations examples are introduced on Fig. 5 and Fig. 6.

Application server Jakarta Tomcat is intermediate layer between client using WWW browser and application layer. Tomcat can be a standalone WWW server or can co-operate with the other servers like Apache, which may work with static part of service. That cooperation may in considerably way improve speed of all service. Server Tomcat makes possible to use SSL protocol, can also restrict access to server's resources or support access to exterior resources (JNDI Resources, JDBC DataSources).

```
<Resource name="jdbc/AccountsDB"
  auth="Container"
  type="javax.sql.DataSource"
  maxActive="100" maxIdle="30"
  maxWait="10000"
  username="root"
password="pass_to_DB"

  driverClassName="com.mysql.jdbc.Dri
  ver"
  url="jdbc:mysql://localhost:3306/na
  me_of_DB?autoReconnect=true&amp;
   useUnicode=true&amp;
   characterEncoding=latin2"/>
```

Fig. 5 - Part of Tomcat's configuration file – server.xml

```
<resource-ref>
  <res-ref-name>jdbc/AccountsDB
  </res-ref-name>
  <res-type>javax.sql.DataSource
  </res-type>
  <init-param
  driver-name="com.mysql.jdbc.Driver"/>
  <init-param url="

jdbc:mysql://192.168.1.62:3306/name_DB"/>
  <init-param user="root"/>
  <init-param password="pass_to_DB "/>
  <init-param max-connections="20"/>
  <init-param max-idle-time="30"/>
</resource-ref>
```

Fig. 6 - Part of Application's configuration file – web.xml

5. SECURITY

In this kind of applications safety needs to be considering as secure connection between client and application server and between email box and application (Fig. 7).

"SSL, or Secure Socket Layer, is a technology which allows web browsers and web servers to communicate over a secured connection. This means that the data being sent is encrypted by one side, transmitted, then decrypted by the other side before processing. This is a two-way process, meaning that both the server and the browser encrypt all traffic before sending out data [3]".

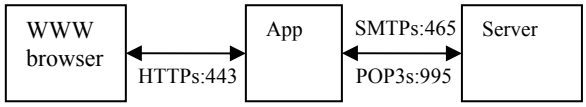

Fig. 7 - Secured connection: browser-App-POP/SMTP Server

Tomcat server supports secure communication (over SSL) between web browser and application [3].

JavaMail supports accessing mail servers over secured connections using SSL or TLS. To simplify such access, three new protocols have been added. In addition to the non-SSL JavaMail protocols "imap", "pop3", and "smtp", the protocols "imaps", "pop3s", and "smtps" can be used to connect to the corresponding services using an SSL connection [6].

This SSL/TLS supported in JavaMail works only when JavaMail is used in version of J2SE that includes SSL support (J2SE 1.4 and J2SE 1.5). The SSL support is provided by the JSSE package, which is also available for earlier versions of J2SE.

When using the new protocol names, configuration properties must also use these protocol names. For instance, set the property "mail.smtps.host" to specify the host name of the machine to connect to when using the SMTP protocol over SSL. The `Transport.send()` method will use the default transport protocol, which remains "smtp". To change the default transport protocol to SMTP over SSL, is necessary to set the property "mail.transport.protocol" to "smtps" (Table 1).

Table 1. Names of the supported protocols used in the JavaMail API [1]

Name	Store or Transport	Uses SSL ?	Supports STARTTLS?
imap	store	no	yes
imaps	store	yes	yes
pop3	store	no	no
pop3s	store	yes	no
smtp	transport	no	yes
smtps	transport	yes	yes

6. WORKING WITH MAILBOX

The JavaMail API supports many different messaging system implementations — different message stores, different message formats, and different message transports. The JavaMail API provides a set of base classes and interfaces that define the API for client applications. Many simple applications will only need to interact with the messaging system through these base classes and interfaces.

To use JavaMail API completely is important to install JavaBeans Activation Framework (JAF). JavaMail API includes lots number of classes. All of them contains in `javax.mail.`

Class *Session* defines settings email's session, configurational parameters for mail server. The object of class the *Session* serves to management with configurational options such: user's name and password, or other properties, which can be used by whole application [1].

The default instance is:

```
Properties props = new Properties();
Session defaultsession =
Session.getDeaultInstance(props,
authenticator);
```

Object props initializes session, contains default properties and setting environmental variables:

- mail.store.protocol - determining default protocol to fetch the message
- mail.transport.protocol - determining applied protocol for sending the message
- mail.host - defines the name of the mail server
- mail.user - defines the user's name, which is used at the connection with the mail server
- mail.protocol.host - is determining default name of the mail server protocol

The object of the Authenticator class is second from parameters. It belongs to the `javax.mail.Authenticator` class, which is used while user's authorization. Object of this class implements `getPasswordAuthentication()` method, which contains the name and passwords data.

Using the Authenticator object isn't obligatory, in that way the other argument is "null".

```
Session session =
Session.getDefaultInstance(props,null);
```

Class *`Message`* is the abstract, defining the group of settings of mail message such: the addresses (the sender, recipient), date of send, subject and content. It contains interface *`Part`* which is used to service of content of message and also defines structure, it fetch and set headlines of message. Main part can to be divided on smaller, what it is possible to use to sending in different formats message or sending attachments. With regard that it is an abstract class, often is used *`javax.mail.internet.MimeMessage`* subclass, which is the representation of electronic message in standard MIME or RFC822.

```
Message message =
new MimeMessage(session);
```

Class `javax.mail.Address` is an abstract class. JavaMail API possesses subclass for SMTP (InternetAddress) and newsgroups (NewsAddress). The object of class the InternetAddress is the standard Internet address defined in RFC 822.

Example of create the object of `Address` class:

```
Address adres = new InternetAddress
("user@dmcs.pl","name surname");
```

Class javax.mail.Store provide connection with mailbox using defined protocol. After establish connection it is possible to get the folder contains messages via object javax.mail.Folder class.

Fig.8 shows part of java code used to fetch messages to local computer. First is important to get the Store object and choose the protocol (pop3, pop3s, imap, imaps), then connect to store (`store.connect()`) using HOST – pop3 server address, USERNAME – username of mailbox and PASSWORD – password to mailbox. Next step is opening the folder and list folders in the store and list/view messages in a folder:

```
Folder inbox =
store.getFolder("INBOX");
```

Than we should get a message's attributes like e.g.:

```
String subj =
MimeUtility.decodeText(m.getSubject());
```

`MimeUtility.decodeText()` is used to decode the Mime Message subject line. The mail.mime.decodetext.strict property controls decoding of MIME encoded words.

The MIME specification requires that encoded words start at the beginning of a white space separated word. Some mailers incorrectly include encoded words in the middle of the word.

Return the MIME type of a message's content:

```
String mimeType = m.getContentType();
```

Get a message's content:

```
Object o = m.getContent();
```

The type of the returned object depends on the type of the actual content. When done, close all open folders and then, the store:

```
inbox.close(false);
store.close();
```

Developers writing a JavaMail client need to write additional viewers that support some of the basic content types-- specifically `message/rfc822`, `multipart/mixed`, and `text/plain`. These are the usual content-types encountered when displaying a `Message`, and they provide the look and feel of the application. Content developers providing additional data types should refer to the JAF specification that discusses how to create `DataContentHandlers` and Beans that operate on those contents.

To send email first at all it is necessary to get the Session object and set “props”.

The "props" Properties file contains mail protocols to use, mail host and port to connect to, username etc. that will be used later to connect to the mail server. It can be blank although you may want to set `mail.store.protocol`, `mail.transport.protocol`, `mail.host`, `mail.user` and `mail.from` properties.
A new message is created by (Fig. 4):

```
Message msg = new MimeMessage(sess);
```

Next step is setting message's attributes like: addresses to:, from:, set subject, date of send and set message's content:

```
msg.setContent(messageText,"iso-8859-
2");
```

To create MIME multipart content, first instantiate a MimeMultipart object. The default subtype of a multipart content is "mixed". You can specify other subtypes such as "alternative", "related", "parallel" and "signed".

```
MimeBodyPart mbp =
new MimeBodyPart(“alternative”);
mbp.setText(messageText);
mbp.setContent(messageText,"iso-8859-
2");

Multipart multip = new
MimeMultipart();
multip.addBodyPart(mbp);
mbp = new MimeBodyPart();
```

Add BodyPart objects to Multipart object

```
    multip.addBodyPart(mbp);
```

and other parts:

```
multip.addBodyPart(mbp1);
    multip.addBodyPart(mbp2);
```

Finally, set the Multipart object as the message's content:

```
    msg.setContent(multip);
```

And send the message:

```
Transport.send(msg);
```

Complete procedures of receiving and sending emails using JavaMail API are introduced on Fig. 8 and Fig. 9 respectively.

```
import javax.mail.*;
import javax.activation.*;
...
   Properties props = new Properties();
   Session sess =
   Session.getDefaultInstance(props,
null);
   Store store = sess.getStore("pop3s");
   store.connect(HOST, USERNAME,
PASSWORD);
   Folder inbox =
store.getFolder("INBOX");
      Message m = inbox.getMessage(num);
   Address[] from = m.getFrom();
   String subj =
 MimeUtility.decodeText(m.getSubject());
   String frm=
MimeUtility.decodeText(from.toString());
   String mimeType = m.getContentType();
   Object o = m.getContent();
   if (o instanceof String) {
     //declaration how to process plain
     //messages
   }
   else if (o instanceof Multipart) {
      Multipart mp =
(Multipart)m.getContent();
      for (int i=0; i< mp.getCount();
i++){
      Part p = mp.getBodyPart(i);
      //declaration how to process
//multipartmessages (attachments)
      }
   else if (o instanceof InputStream) {
   InputStream is = (InputStream)o;
      int c;
      while ((c = is.read()) != -1) {
      out.write(c);
   }
        inbox.close(false);
        store.close();
```

Fig. 8 - Example of receiving message using JavaMail API

7. CONCLUSIONS

The JavaMail API is designed for multiple purposes. Client, server or middleware developers interested in building mail and messaging applications are using the Java programming language. Application developers who need to “mail-enable” their applications. Service Providers who need to implement specific access and transfer protocols.

```
import javax.mail.*;
import javax.activation.*;
...
   Properties prop =
System.getProperties();

prop.put("mail.host",”mail2.p.lodz.pl”);

prop.put("mail.transport.protocol","smtp"
);
   Session sess =
Session.getInstance(prop,
   null);
   Message msg = new MimeMessage(sess);
   msg.setContent(messageText, "iso-8859-
2");
   msg.setSentDate(new Date());

msg.setRecipient(Message.RecipientType.TO
,
   “email_to@domena.pl”);
   msg.setFrom(“email_from@domena.pl”);
   msg.setSubject(
   MimeUtility.encodeText(subject,"iso-
8859-2",
   "Q"));

//below segment if attachment required

   MimeBodyPart mbp = new MimeBodyPart();
            mbp.setText(messageText);
            mbp.setContent(messageText,
iso-8859-2");
            Multipart multip =
   new MimeMultipart();
   multip.addBodyPart(mbp);
   mbp = new MimeBodyPart();
   String FileSource =
   (“path_to_file”);
            FileDataSource FILE =
   new FileDataSource(FileSource);
   mbp.setDataHandler(
   new DataHandler(“FILE”));
            mbp.setFileName(“FILENAME”);
   multip.addBodyPart(mbp);
   msg.setContent(multip);


   Transport.send(msg);
```

Fig.9 - Example of sending message using JavaMail API

A great advantage of this API is independence from the system platform of a user. This feature is more then welcome nowadays in world of business applications and non-commercial software as well.

The author's task has been to gather all this tools together and put them to work as a basic but fully functional email system.

The most important part is to understand ways of working JavaMail API and present it using JSP script or Java servlets, it's also crucial to find balance between performance, easy of use and reliability.

All this process demands from developer to be familiar with networking issues like Java programming, using Java Beans, JNDI Resources, JDBC DataSources, SQL, JavaMail API, configuring Tomcat Server, e-mail system (protocols and standards), security.

8. ACKNOWLEDGEMENTS

This research was supported by the Technical University of Lodz Grant K-25/1/2006/Dz.St.

9. REFERENCES

[1] JavaMail™ API documentation, Sun Microsystems, Inc.

[2] Sun Developers Network, Products & Technologies, JavaMail API http://java.sun.com/products/javamail

[3] http://tomcat.apache.org/tomcat-5.0-doc/ssl-howto.html

[4] Enterprise Java Technologies Tech Tips, http://java.sun.com/developer//EJTechTips/

[5] The Apache Tomcat 5.5 Servlet/JSP Container Documentation, *The Apache Jakarta Project*, http://jakarta.apache.org/tomcat/

[6] Brzózka R., Dzieniecki M., Sakowicz B., Napieralski A.,"Metody zabezpieczania transmisji internetowej", *X Konf."Sieci i Systemy Informatyczne*, Lodz, 17-19 october 2002, pp.49-62

[7] Sakowicz B. , Wojciechowski J., Dura K., "Metody budowania wielowarstwowych aplikacji lokalnych i rozproszonych w oparciu o technologię Java 2 Enterprise Edition." *Slesin 2004*, ISBN: 83-919289-5-0, pp. 163-168

[8] Dura K., Sakowicz B., Napieralski A., "Proposal of extensions to electronic mail client applications" ISBN 966-553-380-0, *TCSET Lviv* 2004

[9] Wilk S., Sakowicz B., Napieralski A., "Wdrażanie aplikacji J2EE w oparciu o serwer Tomcat 5.0 i bazę danych MySQL", *XII Konferencja Sieci i Systemy Informatyczne*, 21-22.10.2004

[10] Ratecki K., Sakowicz B., Wojtowski M., Napieralski A.: "Configurable Client Email Application Working as Web Page", TCSET, Feb. 2006, Lviv, Ukraine

Krzysztof Ratecki, *He received the M.Sc. from Technical University of Lodz in 2005 year. His interests are about Internet applications, Internet protocols, Java language and microchips.*

Bartosz Sakowicz. *He received M.Sc. and currently is Ph.D. student and lecturer at Technical University of Lodz. His interests are about Internet applications, J2EE technology and stochastic optimization. He is an author or co-author of about 30 publications published on national and international conferences. He is supervisor of co-supervisor of about 20 finished Master Thesis. He is also author or co-author of about 20 commercial Internet applications and web sites.*

Marcin Wójtowski. *He received M.Sc. and currently is Ph.D. student at Technical University of Lodz. His interests are about network structures, network administration, and programming languages. . He is an author or co-author of about 10 publications published on national and international conferences. IEEE Member.*

Andrzej Napieralski. *He received the M.Sc. and Ph.D. degrees from theTechnical University of Lodz in 1973 and 1977, respectively, and a D.Sc. degree in electronics from theWarsaw University of Technology (Poland) in January 1989, and in microelectronics from the Université de Paul Sabatié (France) in May 1989. Since October 1991 he has been a Professor at the Technical University of Lodz. From 1992 until 1996 he was the Vice-Director of the Institute of Electronics, and since 1996 he has been Director of the Department of Microelectronics and Computer Science. In 2002 he has been elected as the Vice-Rector for Promotion and International Co-operation. In 1995 he received the title of Professor, and become the Tenured Professor in 1999. He is an author or co-author of 530 publications. In 52nd World Exhibition of Innovation, Research and New Technology, EUREKA 2003 in Brussels, he received the Commander Cross Merite dInvention of the Kingdom of Belgium.*