MVC Model, Struts Framework and File upload Issues in Web Applications Based on J2EE Platform

J.Wojciechowski, B.Sakowicz, K.Dura, A. Napieralski
Department of Microelectronics and Computer Science,
Technical University of Łódź, Poland

Abstract – **This paper describes the web application based on the J2EE platform indicating the MVC model, Struts framework and file upload issues. The development of internet technology and the will to standardize the mechanisms used in implementation of internet applications was the basis background for the development of J2EE platform.**
The project of International Office TUL internet service is an application based on the above mentioned environment. The specification of the project is closely connected with the activities and needs of the staff of the International Office. The article presents the use of MVC model on J2EE platform on an example of open source Apache/Tomcat application server and the database management system MySql.

Keywords - **MVC, Struts, J2EE, JSP, servlet, file upload, JavaBeans, Tomcat.**

I. INTRODUCTION

The J2EE specification is inseparably connected with the Java language. The solution based on Java 2 platform creates a strong, solid, hardware independent base on which one can successfully build systems providing interactive and dynamic Internet sites. J2EE is the platform working in a multi-layer architecture, layered set of system services that are consistently available to J2EE applications across implementations. The J2EE platform runs on top of the J2SE platform, which itself runs on top of the host operating system. In the Web tier, a J2EE Web container provides services related to serving Web requests. The only application needed by the client using the system designed according to J2EE standard is a WWW browser. Since the whole system is located on the server, and not on the clients' computers, new systems can be upgraded and developed easily and efficiently. The Java language with its core features i.e. multitasking and full control upon realizing of applications became the informal standard for solutions made for web and it also combines heterogeneous applications by means of CORBA.

II. MVC -WEB-TIER APPLICATION FRAMEWORK DESIGN

Model-View-Controller ("MVC") is architectural design pattern for interactive applications. MVC organizes an interactive application into three separate modules: one for the application model with its data representation and business logic, the second for views that provide data presentation and user input, and the third for a controller to dispatch requests and control flow. Most Web-tier application frameworks use some variation of the MVC design pattern.

A Model 1 architecture [1] [2] consists of a Web browser directly accessing Web-tier JSP pages. The JSP pages access Web-tier JavaBeans that represent the application model, and the next view to display is determined either by hyperlinks selected in the source document or by request parameters. A Model 1 application control is decentralized, because the current page being displayed determines the next page to display. In addition, each JSP page or servlet processes its own inputs (parameters from GET or POST).

Such approach is partly implemented in project of International Office TUL Internet service. This methodology of Internet applications management is good for a small project but whole the structure-background of communication model must be designed by programmer. This technique is for advanced developers.

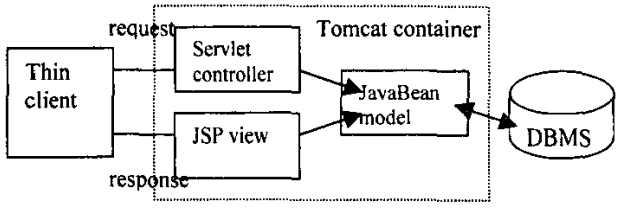

Fig.1 MVC model 1.

A Model 2 architecture [2] introduces a slightly different approach where controller servlet is placed between the browser and the JSP pages. The controller centralizes the logic for dispatching requests to the next view, based on the URL of the request, input parameters, and application state. Model 2 applications are easier to maintain and extend, because views do not refer to each other directly. The Model 2 controller servlet provides a single point of control for security and logging, and often encapsulates incoming data into a form usable by the back-end MVC model.

III. STRUTS FRAMEWORK

An MVC application framework can greatly simplify implementing a Model 2 application. Application frameworks such as Apache Struts [2] and JavaServer Faces™ a configurable front controller servlet, provide abstract classes that can be extended to handle request dispatches. The key scheme is based on servlets for processing requests and selecting views. The Front Controller architectural design pattern centralizes an application's request processing and view selection in a single component. Each type of Web client

sends requests to and receives responses from a single URL, what simplifies client development. The Front Controller receives requests from the client and dispatches them to the application model.

In the J2EE platform, a Front Controller is typically implemented as a servlet. The sample application's Front Controller servlet handles all HTTP requests.

THE VIEW: JSP PAGES AND PRESENTATION

The *View* portion of a Struts-based application is most often constructed using JavaServer Pages (JSP) technology. JSP pages can contain static HTML The JSP environment includes a set of standard action tags, such as <jsp:useBean>. In addition to the built-in actions, there is a standard facility to define your own tags, which are organized into "custom tag libraries."

THE MODEL: BUSINESS LOGIC

The model is predominantly the logic of the application so Struts does not support this since logic depends on the purpose of the application. Anyway model should be separated from both layers. It is not very difficult to obtain separation because we can program the model as a single object without focusing on the rest of Internet application.

THE CONTROLLER: ACTIONSERVLET AND ACTIONMAPPING

The *Controller* portion of the application is focused on receiving requests from the client (typically a user running a web browser), deciding what business logic function is to be performed, and then delegating responsibility for producing the next phase of the user interface to an appropriate View component. In Struts, the primary component of the Controller is a servlet of class ActionServlet

When initialized, the controller parses a configuration file (struts-config.xml) and uses it to deploy other control layer objects. Together, these objects form the Struts Configuration. The Struts Configuration defines (among other things) the ActionMappings [org.apache.struts.action.ActionMappings] for an application.

The Struts controller servlet consults the ActionMappings as it routes HTTP requests to other components in the framework. Often, a request is first forwarded to an Action and then to a JSP (or other presentation page). The mappings help the controller turn HTTP requests into application actions.

IV. STRUTS FILE UPLOAD EXAMPLE

File upload must be started with an HTML form *including <input> element of type file, defaulting to the* specified value or the specified property of the bean associated with our current form. With the corresponding HTML <input> element, enclosing form element must specify "POST" method attribute, and "multipart/form-data" for the enctype attribute. For example:

```
<html:form method="POST" enctype="multipart/form-data">
    <html:file property="theFile" />
</html:form>
```

Handling multipart forms is to provide more than one input of type "file". The first step to create a multipart form is to utilize the struts-html taglib to create the presentation page:

```
<%@page language="java">
<%@taglib uri="/WEB-INF/struts-html.tld"
    prefix="html">
<html:form action="uploadAction.do">
 Please Input Text:
 <html:text property="myText"><br/>
 Please Input The File You Wish to Upload:<br/>
 <html:file property="myFile"><br />
 <html:submit />
</html:form>
```

The next step is to create the ActionForm bean:

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.upload.FormFile;
public class UploadForm extends ActionForm {
 protected String myText;
 protected FormFile myFile;
 public void setMyText(String text) {
  myText = text;
 }
 public String getMyText() {
  return myText;
 }
 public void setMyFile(FormFile file) {
  myFile = file;
 }
 public FormFile getMyFile() {
  return myFile;
 }
}
```

The FormFile class provides the methods to manipulate files in file uploading. To retrieve the FormFile one should call method in action class (znowu nie rozumiem!)

```
((UploadForm) form).getMyFile();
```

V. UPLOADING FILES WITH BEAN

What org.apache.struts.upload.FormFile does is described below. HTTP request is critical because when you process an uploaded file, you work with raw data obtained from an HttpServletRequest object's methods such as

```
ServletInputStream in = request.getInputStream();
```

THE CLIENT PART

Multipart/form-data defines a new MIME media type [4] and specifies the behavior of HTML user agents when interpreting a form with enctype="multipart/form-data" or <input type="file" /> tags. An HTML form should have

enctype parameter according to RFC 1867 "Form-based file upload" document. By the way Struts defines its own tags like in example above. So if we do not use this framework we should write the following code:

```
<FORM ACTION="upload.jsp ENCTYPE="multipart/form-
data"        METHOD=POST>
   What is your name? <INPUT TYPE=TEXT
NAME=submitter>
   What files are you sending? <INPUT TYPE=FILE
NAME=pics>
   </FORM>
```

When an input tag of 'FILE' type is encountered, the browser might show previously selected file names and a "Browse" button or selection method.

If the user indicates one file (file1.txt) as the answer, the client might send back the following data [3] (the HTTP header is omitted):

```
Content-type: multipart/form-data, boundary=AaB03x

    --AaB03x
    content-disposition: form-data; name="field1"

    Joe Blow
    --AaB03x
    content-disposition: form-data; name="pics";
filename="file1.txt"
    Content-Type: text/plain

    ... contents of file1.txt ...
    --AaB03x--
```

If the user indicates two files for the answer (file1.txt and file2.gif), the client might send back the following data [3] (the HTTP header is omitted):

```
        Content-type: multipart/form-data,
boundary=AaB03x

    --AaB03x
          content-disposition: form-data; name="field1"

    Joe Black
    --AaB03x
    content-disposition: form-data; name="pics"
    Content-type: multipart/mixed, boundary=BbC04y

    --BbC04y
    Content-disposition: attachment; filename="file1.txt"

    Content-Type: text/plain

    ... contents of file1.txt ...
    --BbC04y
    Content-disposition: attachment; filename="file2.gif"
    Content-type: image/gif
    Content-Transfer-Encoding: binary

    ...contents of file2.gif...
    --BbC04y--
    --AaB03x--
```

THE HTTPSERVLETREQUEST INTERFACE

The ServletRequest interface is a generic interface that is extended by HttpServletRequest to provide request information for HTTP Servlets.
The HttpServletRequest interface has the following signature.

public interface HttpServletRequest extends ServletRequest

The Servlet container creates an HttpServletRequest object and passes it as an argument to the Servlet's service methods (doGet, doPost, etc). In presented implementation this HttpServletRequest object is created by the Servlet/JSP container and is passed to a File Upload Bean, for further processing.
This object represents the HTTP request that contains the element name-value pairs from the form and the uploaded file. The method starts by obtaining the ServletInputStream object using the getInputStream method of the HttpServletRequest object.

Those input values are separated from each other by a delimiter, called a boundary. This delimiter consists of a dash characters followed by a random number. In the example above, the boundary is the following line.

--AaB03x

All HTML form elements begin with a boundary followed by the content-disposition line that starts with the following characters.
Content-Disposition: form-data; name=
There are two types of form elements: file and non-file (normal form elements such as TEXT or HIDDEN elements). The difference between the two is based on the file element, which contains the following string.
filename="*filename*"
Therefore, this information enable us to distinguish the file from the non-file input elements using the following two "if statements".

```
if (newLine.startsWith("Content-Disposition: form-data;
name=\"")) {
if (newLine.indexOf("filename=\"") != -1) {
// a file form element, code to extract the file here
. . . }
else {
//this is a field, code to extract fields here}
}
```

The situation is more complicated when if sending more files but this problem will not be discussed in this article.
After parsing the tokens indicating the appropriate read position for the file upload bean. The procedure should write the file to the disk.

```
PrintWriter pw = new PrintWriter(new BufferedWriter(
new FileWriter(
( savePath==null? "" : savePath ) + filename
)));
```

Where the file should be saved to depends on whether the savePath field has been set. If the savePath was not set, its value is null, and the file should be saved in the default directory. If the savePath field has been set, its value will not be null so the uploaded file must be uploaded to a given directory [3].

We can then extract the file content by using a while loop that reads one line at a time and writes it to the disk using the write method of the PrintWriter. The last line of the file includes two carriage return line feed characters. Therefore, the bytes saved to the disk must not include these two characters. So if the line read is not the last line of the file, write all the bytes read to the disk. If it is, write all the bytes read but the last two characters.

The next line after the file content is another boundary. Or, if the file is the last HTML form element, the next line is the boundary followed by two dash characters. Therefore, by checking whether or not the next line is a boundary we will know how to exit from the while loop. To determine the end of file one could compare the length of the byte array read by readLine. The latter should be equal to boundaryLength + 2.

Or, if it is the last line in the HttpServletRequest object, it should be equal to boundaryLength + 4 because of the last two dash characters.

```
while (i != -1 && !newLine.startsWith(boundary)) {
i = in.readLine(line, 0, 128);
if ((i==boundaryLength+2 || i==boundaryLength+4) // + 4 is
eof
&& (new String(line, 0, i).startsWith(boundary)))
pw.print(newLine.substring(0, newLine.length()-2));
else
pw.print(newLine);
newLine = new String(line, 0, i);
}
pw.close();
```

VI. REFERENCES

[1] J.Goodwill, *Pure Java Server Pages*, Sams, Paperback, Published June 2000.

[2] The Apache Jakarta Project - documentation, http://jakarta.apache.org/struts/

[3] B. Karnivan, "Uploading Files with Beans," http://www.oreillynet.com/pub/au/136

[4] Network Working Group, RFC 1867 - Form-based File Upload in HTML, November 1995, http://www.faqs.org/rfcs/.

[5] The Struts console http://www.jamesholmes.com/struts/console/

VII. CONCLUSIONS

In this paper the model-view-controler-parradigma for servlets was presented. One approach is to design MVC model into Internet applications by its own scheme or to use Struts implementation. Undoubtedly applying this model to our code will split the presentation layer from the logic layer of our application making it more flexible and scalable. The Struts framework enforces on programmer good object-oriented programming techniques thus decreasing mess in the code. However Struts is not the part of J2EE platform sensustricte but proposes good solution especially when the project is getting bigger. Struts has also some disadvantages. Its weaknees is weak support for complex interactions with user. The programmer of Internet Java-driven applications should take into account concurrent event-driven Java Server Faces environment.

Struts tag library supports file upload but it can be implemented as well by a programmer by himself.