Applying a Model/View/Controller Pattern in J2EE Platform Using Struts Framework

Niko Ibrahim
Jurusan Sistem Informasi
Fakultas Teknologi Informasi, Universitas Kristen Maranatha
Jl. Prof. Drg. Suria Sumantri No. 65 Bandung 40164
Email: niko.ibrahim@gmail.com

Abstrak

Pola desain *Model/View/Controller* (MVC), yang juga dikenal dengan Model-2 pada pemrograman J2EE, adalah pola desain yang telah mapan dalam dunia pemrograman. Pola ini telah diterapkan sebagai pemodelan yang sanggup memisahkan berbagai komponen dalam suatu aplikasi. Dalam pemrograman Web, pola desain MVC ini dapat membantu para pemrogram untuk mengkontrol perubahan pada aplikasi yang dibuatnya. Hal ini disebabkan karakteristik pola desain MVC yang sanggup memisahkan antarmuka dari logika program (*business rule*) serta dari data (*database*) yang digunakan. Pada studi literatur ini akan dibahas dasar-dasar dari MVC dan penerapannya dalam mengembangkan aplikasi Web berbasis Java. Pola desain MVC yang dibahas disini adalah yang berbasis *open source* yang dikenal dengan nama *Struts framework*.

Kata kunci: *Model/view/controller, Open source, Struts framework*

1. Introduction

The Internet has revolutionized our business by providing an information highway, which acts as a new form of communication backbone [Nag03]. This new information medium has shifted business from the traditional infrastructures to a virtual world where they can serve customers anytime and anywhere. Additionally, it enhances our organizations with significant benefits in terms of business productivity, cost savings, and customer satisfaction. As a result, modern organizations are required to re-evaluate their business models and plan on a business vision to interact with their stakeholders using an Internet-based technology space.

With respect to Internet-based technology, modern organizations need to develop their web-based business applications which have a set of fundamental elements [Bro03]: business objects, process-oriented or service-based objects, and user interaction components. For example, in banking industry, businesses deal with different entities all of the time. These range from higher-level entities such as a customer to lower levels such as deposit and withdrawal. These entities share a number of common characteristics:

- Behaviors
- Properties
- Relationships with other entities
- Rules or policies

Those business entities are of course the foundation of object-oriented design and development which undoubtedly is a key aspect in any J2EE application development.

2. Application Design Using MVC Pattern

Much discussion and confusion have been relayed in the literature about what best approaches in developing complex business web applications. Any business applications will need to incorporate three primary elements [Bro03]:

- User interaction
- Business Process
- Business Entities

The structure and relationship of these elements is shown in Figure 1.

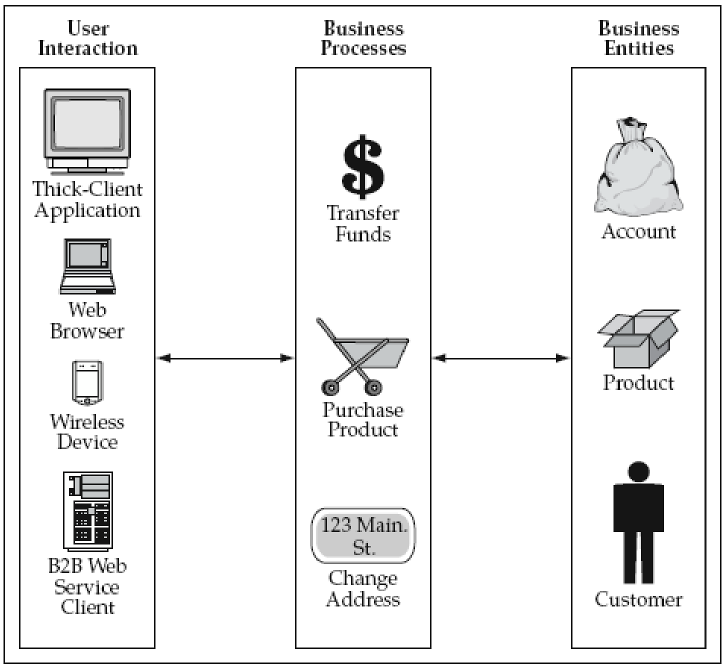


Figure 1: The Structure of a Business Application

Referred to a current research in IBM [Tai04], as web application grow in size, it becomes more and more critical to support modular application design and parallel development. It also needs to support the integration of various kinds of programming technologies.

2.1. The Model/View/Controller Architectural Pattern

Model/View/Controller architectural design is projected to help developers modularise an application. The benefit of using the MVC pattern is that developers isolate the different portions of the application in order to provide greater flexibility and more opportunity for reuse. A primary isolation point is between the presentation objects and the application back-end objects that manage the data and business rules. This allows a user interface to have many different screens that can be changed to a large degree without impacting the business logic and data components [Bro03][Tai04][Lef01].

Figure 2 shows how the Model, the View, and the Controller interact with one another in J2EE platform:

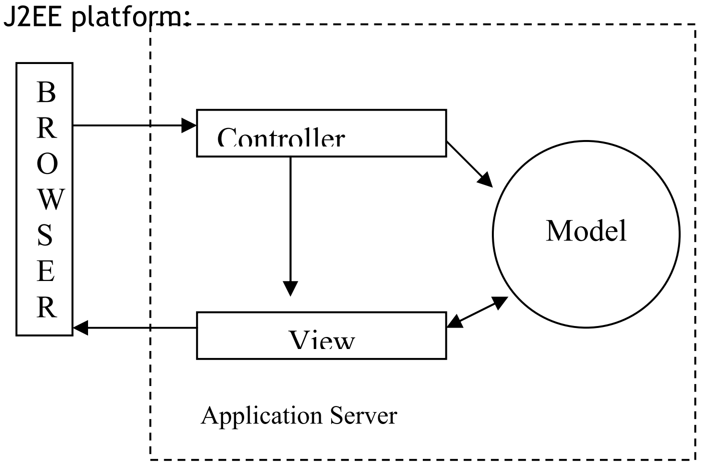


Figure 2: MVC architecture for Java applications

Some of the major benefits of using the MVC are [Goo04]:

- **Reliability**: The presentation and transaction layers have clear separation, which allows we to change the look and feel of an application without recompiling Model or Controller code.
- **High reuse and adaptability**: The MVC lets we use multiple types of views, all accessing the same server-side code. This includes anything from Web browsers (HTTP) to wireless browsers (WAP).
- **Very low development and lifecycle costs**: The MVC makes it possible to have lower-level programmers develop and maintain the user interfaces.
- **Rapid deployment**: Development time can be significantly reduced, because Controller programmers (Java developers) focus solely on transactions, and View programmers (HTML and JSP developers) focus solely on presentation.

- **Maintainability**: The separation of presentation and business logic also makes it easier to maintain and modify a Struts-based Web application.

The Model

The model encapsulates the functional core of an application, its business logic. The goal of MVC is to make the model independent of the view and controller which together form the user interface of the application. An object may act as the model for more than one MVC triad at a time.

Since the model must be independent, it cannot refer to either the view or controller portions of the application. The model may not hold direct instance variables that refer to the view or the controller. It passively supplies its services and data to the other layers of the application [Tai04].

The View

The view obtains data from the model and presents it to the user and represents the output of the application. It generally has free access to the model, but should not change the state of the model. Views are read only representations of the state of the model and read data from the model using query methods provided by the model

The Controller

The controller component isolates how a user's actions on the screen are handled by the application. This allows for an application design to flexibly handle things such as page navigation and access to the functionality provided by the application model in the case of form submissions. This also provides an isolation point between the model and the view. Because the controller component handles the user requests and invokes functions on the model as necessary, it allows for a more loosely coupled front and back end. Interaction between the model and the view is only through an event-based mechanism that informs the view of changes to the model's data [Bro03].

3. J2EE Platform

The J2EE platform specifies technologies to support multitier enterprise applications. These technologies fall into three categories: component, service, and communication. The component technologies are those used by developers to create the essential parts of the enterprise application, namely the user interface and the business logic. The component technologies allow the development of modules that can be reused by multiple enterprise applications. The component technologies are supported by J2EE platform's system-level services. These system-level services simplify application programming and allow components to be customized to use resources available in the environment in which they are deployed [Sin02].

Since most enterprise applications require access to existing enterprise information systems, the J2EE platform supports APIs that provide access to databases, enterprise information systems such as SAP and CICS, and services such as transaction, naming and directory, and asynchronous communication. Finally, the J2EE platform provides technologies that enable communication between clients and servers and between collaborating objects hosted by different servers.

Figure 3 shows the basic architecture of J2EE platform:

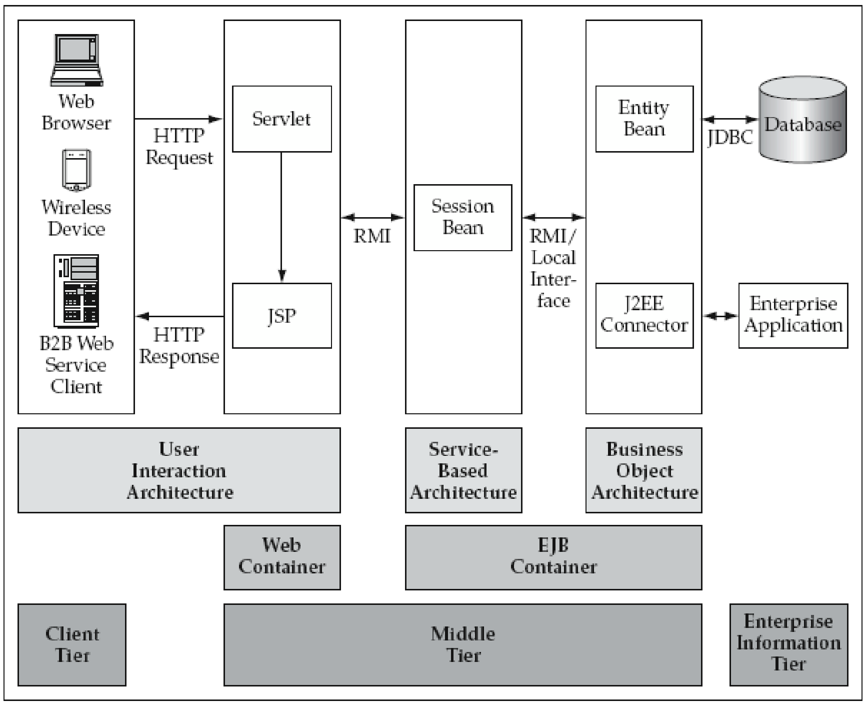


Figure 3: Basic J2EE Architecture

3.1. The Client Tier

From a developer's point of view, a J2EE application can support many types of clients. J2EE clients can run on laptops, desktops, palmtops, and cell phones. They can connect from within an enterprise's intranet or across the World Wide Web, through a wired network or a wireless network or a combination of both. They can range from something thin, browser-based and largely server-dependent to something rich, programmable, and largely self-sufficient.

From a user's point of view, the client is the application. It must be useful, usable, and responsive. Because the user places high expectations on the client, we must choose our client strategy carefully, making sure to consider both technical forces (such as the network) and non-technical forces (such as the nature of the application) [Bro03].

3.2. The Web Container

A typical Java Web application consists of a collection of JavaServlets and JSPs that run inside a J2EE server's Web container. The container manages each component's lifecycle, dispatches service requests to application components, and provides standard interfaces to context data such as session state and information about the current request. [Bro03].
JavaServlets and JSPs are deployed in the Web container and typically performs the following functions in a J2EE application:

- Web-enables business logic - they manage interaction between Web clients and application business logic.
- Generates dynamic content - they generate content dynamically, in entirely arbitrary data formats, including HTML, images, sound, and video.
- Presents data and collects input - they translate HTTP PUT and GET actions into a form that the business logic understands and present results as Web content.
- Controls screen flow - The logic that determines which "screen" (that is, which page) to display next usually resides in these components, because screen flow tends to be specific to client capabilities.
- Maintains state - they have a simple, flexible mechanism for accumulating data for transactions and for interaction context over the lifetime of a user session.
- Supports multiple and future client types - Extensible MIME types describe Web content, so a Web client can support any current and future type of downloadable content.
- May implement business logic - While many enterprise applications implement business logic in enterprise beans, Web-only, low- to medium-volume applications with simple transactional.

Java Servlets

A Java Servlet is a Java class that extends a J2EE-compatible Web server. Each servlet class produces dynamic content in response to service requests to one or more URLs.
Servlets offer some important benefits over earlier dynamic content generation technologies. Servlets are compiled Java classes, so they are generally faster than CGI programs or server-side scripts. Servlets are safer than extension libraries, because the Java Virtual Machine (JVM) can recover from a servlet that exits unexpectedly. Servlets are portable both at the source-code level (because of the Java Servlet specification) and at the binary level (because of the innate portability of Java bytecode). Servlets also provide a richer set of standard services than any other widely adopted server extension technology.

Figure 4 shows a Java Servlets that prints “Hello World” in a browser.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
      public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException, IOException
       {
            PrintWriter out = response.getWriter();
            out.println("Hello World");
      }
}
```

Figure 4: A “Hello World” Java Servlet

JavaServer Pages (JSP)

Most Web applications produce primarily dynamic HTML pages that, when served, change only in data values and not in basic structure. For example, all of the catalog pages in an online store may have identical structure and differ only in the items they display. JSP technology exists for producing such content [Rod03][Bro03].

A JSP page is a document containing fixed template text, plus special markup for including other text or executing embedded logic. The fixed template text is always served to the requester just as it appears in the page, like traditional HTML. The special markup can take one of three forms: directives, scripting elements, or custom tags (also known as “custom actions”) [Sin02].

Figure 5 shows a JSP page that prints a “Hello World in a browser:

```
<html>
      <hody>
      <p>
            <%= "Hello, world!" %>
      </p>
      </body>
</html>
```

Figure 5: A “Hello World” JSP

The first time the JSP file is requested, it is translated into a servlet and then compiled into an object that is loaded into resident memory. The generated servlet then services the request, and the output is sent back to the requesting client. On all subsequent requests, the server checks to see whether the original JSP source file has changed. If it has not changed, the server invokes the previously compiled servlet object. If the source has changed, the JSP engine re-parses the JSP source. The steps are as follow (Figure 5):

1) Client requests a JSP page
2) The JSP Engine compiles the JSP into a Servlet
3) The generated servlet is compiled and loaded
4) The compiled servlet services the request and sends a response back to the client

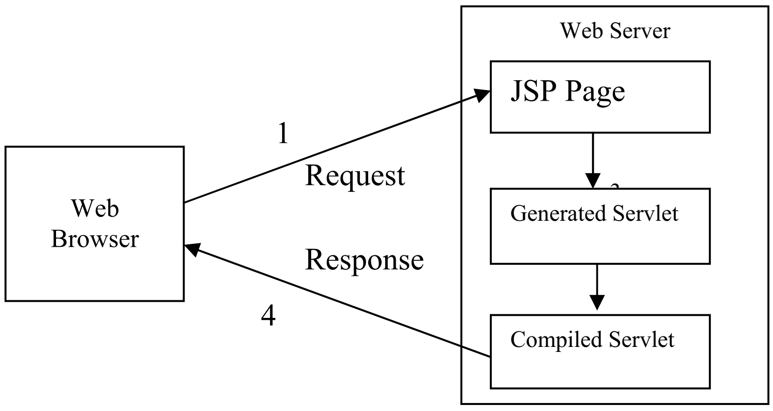


Figure 6: The execution of a Java servlet

4. Development Frameworks

Frameworks, like many things, are often born out of necessity. In the process of creating multiple applications, developers discover that there are some areas of functionality that are common. These areas are then abstracted and generalized to form the features of the framework.
Frameworks provide benefit at many different levels of application development [Sin02][Goo04]:

- Structure: frameworks provide a structure for application development - a backbone to build from, making the development process a little easier. They ensure that our project includes all of the key elements such as security, database access, UI, and so forth by providing an outline into which the details of our particular application are filled.
- Services: a framework provides a wide array of services to its applications, for use in both the development and deployment phases. These services very often begin with either a component container, or access to a component container provided by the application server.
- Completeness: using a framework can expand the horizons slightly: if adding some of the "nice to have" functionality does not cost any

additional time, we can end up with a richer, more complete application that is comparatively more polished than it could have been without the framework.

An application framework is difficult to create. It is difficult, not so much because the code is difficult to write, or that there is a large amount of code. Indeed, many enterprise applications have several times more code than the frameworks that helped create them. The code of a framework is often complex; however, proper design helps to make it understandable. This complexity, though, is not the reason for frameworks being so difficult to create. The difficulty lies in obtaining the experience that is distilled into them. Frameworks evolve from the effort to reuse both design and code, and from repeated refactoring that results in widely applicable services and components. This is what is difficult about their creation.

Because frameworks tend to be large and fairly complex pieces of software, they are often not a cost-effective project for a single organization. It is also a good thing when a particular framework is used in many projects - it grows, it becomes a de facto standard to a degree, and the organizations using it get the benefits. All of these factors mean that frameworks are good candidates for open-source projects, and indeed many of the best frameworks are open source.

4.1. Overview of Struts Frameworks

The most widely adopted MVC framework is the open source Apache Struts (http://jakarta.apache.org/struts/). Struts was originally written by Craig McClanahan, the main developer of the Tomcat servlet engine, and was released in mid 2000, making it the longest-established open source web application framework. Partly because of its relatively long history, Struts has achieved remarkable buy-in by developers, with many add-ons now available around it [Goo04].

Struts provides key capabilities to the developer for building virtually any Web application - especially those that make use of JSP pages as their user interface technology. Struts can, however, interact with many other presentation technologies that include XML/XSLT [Nas03].

The primary elements of Struts are [Bro03]:

- A controller ActionServlet that delegates specific request handling to Action classes
- An extensive JSP custom tag library
- A library of utility classes to support Web application development

The basic Struts architecture is shown in Figure 6:

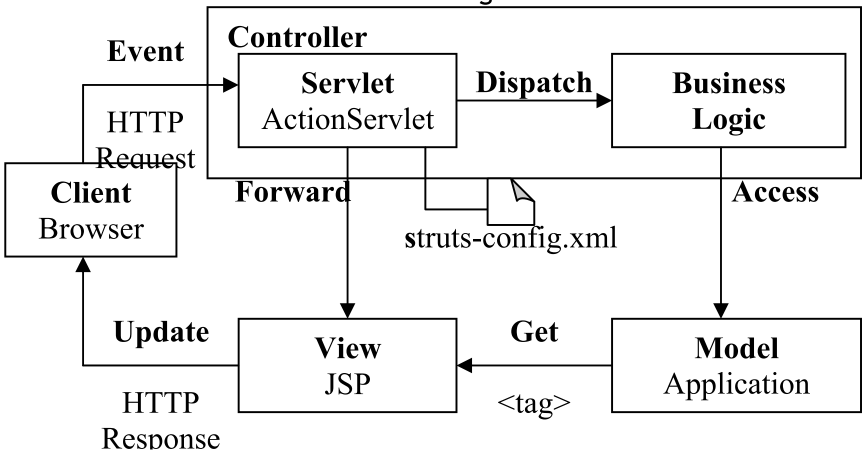


Figure 6: The basic Struts architecture

In figure 6, we can see that the Struts controller layer has a component called Actions. Action is essentially a Java class that is responsible for examining information from the request, performing some operation, optionally populating data that will later be used by the view layer, and then communicating to the ActionServlet where control should be forwarded next.

4.2. Benefits and Issues of Struts Framework

Struts addresses a gap in the J2EE technological stack not covered by Servlet/JSP or EJB. Struts should be one of the top choices when we are considering the adoption of an MVC framework due to its numerous benefits. However, despite of the benefits of Struts, we should also consider some issues of using it [Rod03].
The following are some benefits of using Struts:

- **Use of JSP tag mechanism**
  The tag feature promotes reusable code and abstracts Java code from the JSP file. This feature allows nice integration into JSP-based development tools that allow authoring with tags.
- **Tag library**
  Struts provides developers with an extensive range of tag library.
- **Open source**
  We have all the advantages of open source, such as being able to see the code and having everyone else using the library reviewing the code. Many eyes make for great code review.
- **Manage the problem space**
  Divide and conquer is a nice way of solving the problem and making the problem manageable.

The following are some issues of using Struts that might be considered by developers:

- **Limited scope**
  Struts is a Web-based MVC solution that is meant be implemented with HTML, JSP files, and servlets.   Developers must be expert in J2EE platform if they intended to use Struts in their development phases.
- **Complexity**
  Separating the problem into parts introduces complexity. There is no question that some education will have to go on to understand Struts. With the constant changes occurring, this can be frustrating at times.

4.3. Example of Struts Application: Login

The purpose of the login application is to give a look at the basic of a Struts application. To help us stay on track, this application contains only the components needed to demonstrate the framework. It contains no real business logic, unit tests, or complex forms.
Basically, the application has two screens: welcome and login page (Figure 7).

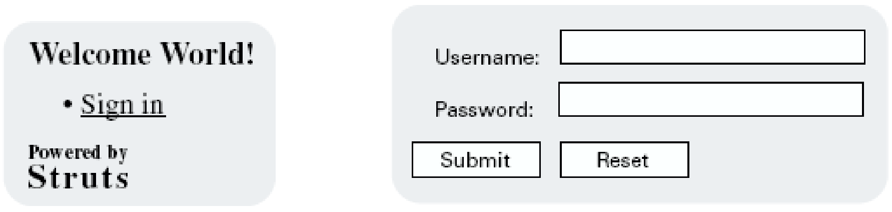
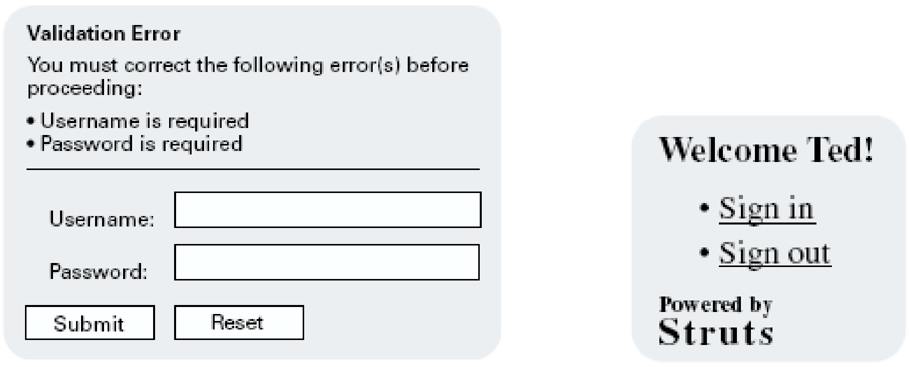


Figure 7: Welcome and Login Page

The application also has a built-in validation mechanism that check an empy input. Finally, right after the users logged in, they will be redirected to the Welcome page (Figure 8).


Validation Error

You must correct the following error(s) before proceeding:

- Username is required
- Password is required

Figure 8: Login Page (validation) and Redirected Welcome Page

We put all the source code in the appendix A for space-saving purposes. We also draw the UML diagram for the login process in the appendix B. The source code for the example application may also be obtained from the author.

Conclusions and Suggestions

We have discussed that J2EE is used for developing, deploying, and executing applications in a distributed environment. The J2EE application server acts as a platform for implementing various server-side technologies such as servlets, Java Server Pages (JSP), and Enterprise JavaBeans. J2EE allows us to focus on business logic in your programs. The business logic is coded in Enterprise JavaBeans, which are reusable components that can be accessed by client programs.

Moreover, we also talked about Struts which solved some problems using tags and Model/View/Controller architecture pattern. This approach aided in code re-usability and flexibility. By separating the problem into smaller components, we will be more likely to reuse when changes do occur in the technology or problem space. Additionally, Struts enabled page designers and Java developers to focus on what they do best. Yet, the trade off in increased robustness implies an increase in complexity. Struts is much more complex than a simple single JSP page, but for larger systems Struts actually helps manage the complexity.

As final words, we suggest that we may have a look at other framework technologies available in the market. Struts is not the only framework that we can use for developing Java Web applications. Recently, Sun Microsystems released a new approach for developing Web applications namely JavaServer Faces (JSF). JSF has learnt a lot from Struts although it has not considerably matured yet.

Bibliography

Broemmer, D. (2003). *J2EE Best Practices - Java Design Patterns, Automation, and Performance*. Canada : Wiley Publishing, Inc.

Goodwill, J., & Hightower, R. (2004). *Professional Jakarta Struts*. Canada : Wiley Publishing, Inc.

Leff, A., & Rayfield, J.T. (2001). Web-Application Development Using the Model/View/Controller Design Pattern. Proc. *IEEE International Enterprise Distributed Object Computing Conference,* Seattle, USA, 11:118-127, IEEE.

Nagappan, R., Skoczylas, R. & Sriganesh, R.P. (2003). *Developing Java Web Services*. Canada : Wiley Publishing, Inc.

Nash, M. (2003). *Java Frameworks and Components - Accelerate Your Web Application Development*. United Kingdom : Cambridge University Press.

Rod, J. (2003). *Expert One-on-One J2EE Design and Development*. USA : Wiley Publishing, Inc.

Singh, I., Stearns, B., & Johnson, M. (2002). *Designing Enterprise Applications with the J2EETM Platform, 2nd Edition*. USA : Sun Microsystems, Inc.

Tai, H., Mitsui, K., Nerome, T., Abe, M, Ono, K. & Hori, M. (2004). Model-driven development of large- scale Web applications. *IBM Journal of Research and Development* 48(5): 797-809.

Appendix A

The purpose of our login application is to give a look at the basic of a Struts application. To help us stay on track, this application contains only the components needed to demonstrate the framework. It contains no real business logic, unit tests, or complex forms.

1. The Welcome Screen

The first time we visit the welcome screen, there will be only one link, which reads, “Sign in” (Figure A-1)

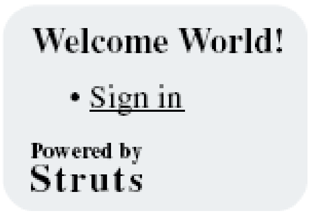


Figure A-1: Welcome Screen

The following is the JSP source code for the welcome screen:

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<HTML><HEAD><TITLE>Welcome World!!</TITLE><html:base/>
</HEAD><BODY>
<logic:present name="user">
<H3>Welcome <bean:write name="user" property="username"/>!</H3>
</logic:present>
<logic:notPresent scope="session" name="user">
<H3>Welcome World!</H3>
</logic:notPresent>
<html:errors/>
<UL>
<LI><html:link forward="logon">Sign in</html:link></LI>
<logic:present name="user">
<LI><html:link forward="logoff">Sign out</html:link></LI>
</logic:present></UL>
<IMG src='struts-power.gif' alt='Powered by Struts'>
</BODY></HTML>
```

2. Login Screen

The logon screen submits the username and password, as you can see in figure A-2.
When we submit the form without entering anything, the login screen returns but with a message, like the one shown in the figure A-3.

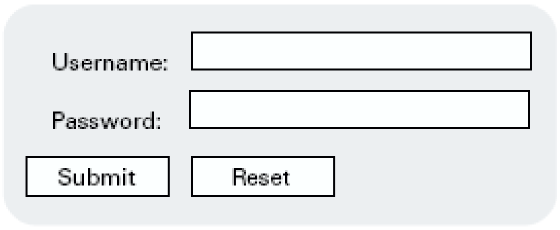
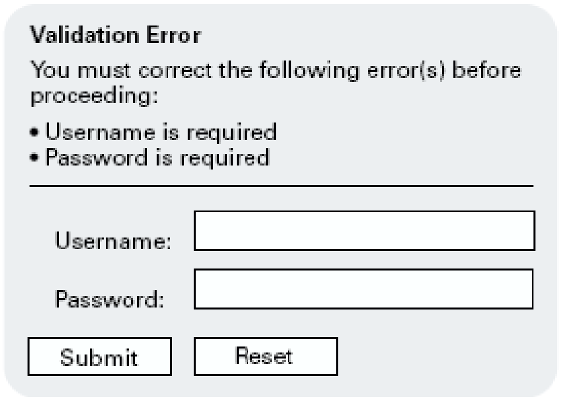


Figure A-2: Login Screen


Validation Error

You must correct the following error(s) before proceeding:

- Username is required
- Password is required

Figure A-3: Login Screen with Validation Messages
The following is the JSP source code for the Login Screen:

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<HTML><HEAD><TITLE>Sign in, Please!</TITLE></HEAD><BODY>
<html:errors/>
<html:form action="/LogonSubmit" focus="username">
<TABLE border="0" width="100%">
<TR><TH align="right">Username:</TH>
<TD align="left"><html:text property="username"/></TD></TR>
<TR><TH align="right">Password:</TH>
<TD align="left"><html:password property="password"/></TD></TR>
<TR><TD align="right"><html:submit/></TD>
<TD align="left"><html:reset/></TD></TR>
</TABLE></html:form></BODY></HTML>
```

3. The Login Form Source

When an HTML form is submitted, the name-value couplets are caught by the Struts controller and applied to an ActionForm. The ActionForm is a JavaBean with properties that correspond to the controls on an HTML form. Struts compares the names of the ActionForm properties with the names of the incoming couplets. When they match, the controller sets the property to the value of the corresponding couplet. Extra properties are ignored. Missing properties retain their default value (usually null or false). Here are the public properties from our LogonForm:

```
private String password = null;
public String getPassword() {
      return (this.password);
}

public void setPassword(String password) {
      this.password = password;
}

private String username = null;
public String getUsername() {
      return (this.username);
}

public void setUsername(String username) {
      this.username = username;
}
```

4. Validation Source

Here is the validate method from our LogonForm. It checks that both fields have something entered into them.

```
public ActionErrors validate(ActionMapping mapping,
HttpServletRequest request) {
  ActionErrors errors = new ActionErrors();
    if ((username == null) || (username.length() < 1))
       errors.add ("username",new
       ActionError("error.username.required"));
    if ((password == null) || (password.length() < 1))
       errors.add("password", new
ActionError("error.password.required"));
   return errors;
}
```

5. LoginAction Source (LoginAction.java)

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;
public final class LogonAction extends Action {
// Validate credentials with business tier
public boolean isUserLogon (String username, String password) throws
UserDirectoryException {
return (UserDirectory.getInstance().isValidPassword(username,password));
} // end isUserLogon
public ActionForward perform(ActionMapping mapping,ActionForm
form,HttpServletRequest request,HttpServletResponse response) throws
IOException, ServletException {
// Obtain username and password from web tier
       String username = ((LogonForm) form).getUsername();
       String password = ((LogonForm) form).getPassword();
// Validate credentials
boolean validated = false;
try { validated = isUserLogon(username,password);
}
catch (UserDirectoryException ude) {// couldn't connect to user directory
       ActionErrors errors = new ActionErrors();
       errors.add(ActionErrors.GLOBAL_ERROR,new
       ActionError("error.logon.connect"));
       saveErrors(request,errors);
// return to input page
return (new ActionForward (mapping.getInput()));
}
// Save our logged-in user in the session, because we use it again later.
  HttpSession session = request.getSession();
  session.setAttribute(Constants.USER_KEY, form);
// Log this event, if appropriate
  if (servlet.getDebug() >= Constants.DEBUG) {
       StringBuffer message = new StringBuffer("LogonAction: User '");
       message.append(username);
       message.append("' logged on in session ");
       message.append(session.getId());
       servlet.log(message.toString);
}
// Return success
       return (mapping.findForward (Constants.WELCOME));
} // end perform
} // end LogonAction
```

6. Welcome Screen After Users Logged In

After a successful login, the welcome screen displays again, but with an added ‘Sign Out’ link (Figure A-4).

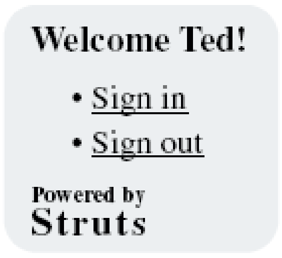
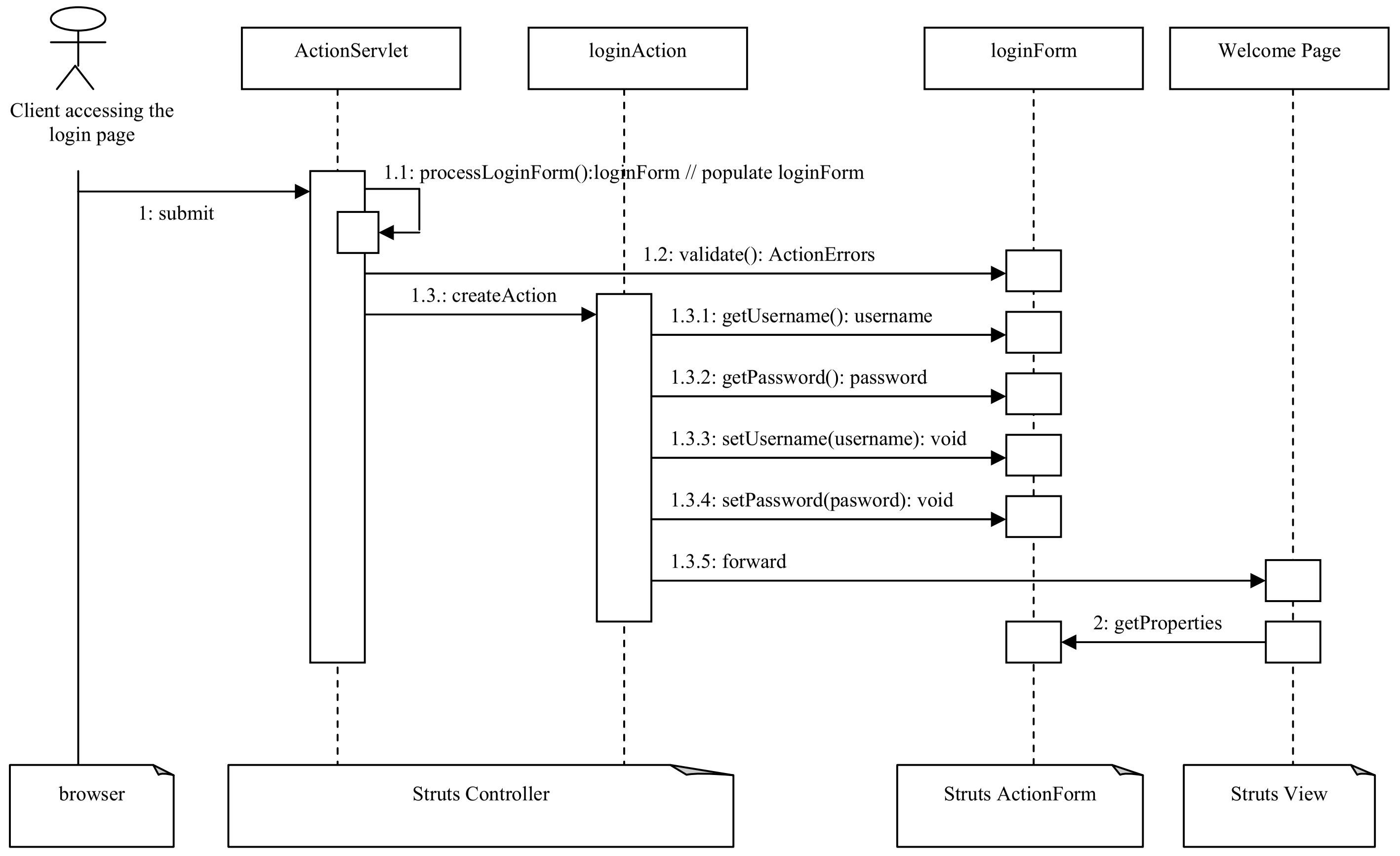


Figure A-4: Welcome Screen Revisited
The folloging is the LogoffAction.java source code:

```
public ActionForward perform(ActionMapping mapping,ActionForm
form,HttpServletRequest request,HttpServletResponse response)
throws IOException, ServletException {
// Extract attributes we will need
  HttpSession session = request.getSession();
  LogonForm user = (LogonForm)
  session.getAttribute(Constants.USER_KEY);
// Log this user off
  if (user != null) {
    if (servlet.getDebug() >= Constants.DEBUG) {
       StringBuffer message = new StringBuffer("LogoffAction: User
'");
       message.append(user.getUsername());
       message.append("' logged off in session ");
       message.append(session.getId());
       servlet.log(message.toString());
}}
  else {
  if (servlet.getDebug() >= Constants.DEBUG) {
       StringBuffer message = new StringBuffer("LogoffAction: User
'");
       message.append(session.getId());
       servlet.log(message.toString());
}}     // Remove user login
  session.removeAttribute(Constants.USER_KEY);
// Return success
  return (mapping.findForward (Constants.SUCCESS));
}} // end LogoffAction
```

- end of appendix -

Appendix B – The UML Diagram for the Login Process


Client accessing the login page
ActionServlet
loginAction
loginForm
Welcome Page
1: submit
1.1: processLoginForm():loginForm // populate loginForm
1.2: validate(): ActionErrors
1.3.: createAction
1.3.1: getUsername(): username
1.3.2: getPassword(): password
1.3.3: setUsername(username): void
1.3.4: setPassword(pasword): void
1.3.5: forward
2: getProperties
browser
Struts Controller
Struts ActionForm
Struts View