Comparison of Four Popular Java Web Framework Implementations: Struts1.X, WebWork2.2X, Tapestry4, JSF1.2

Peng Wang

University of Tampere
Department of Computer Sciences
Master's thesis
Supervisor: Roope Raisamo
May 2008

University of Tampere
Department of Computer Sciences
Peng Wang: Comparison of Java Web Framework: Struts1.X, WebWork2.2X, Tapestry4, JSF1.2
Master's thesis, 101 pages
May 2008

Java web framework has been widely used in industry Java web applications in the last few years, its outstanding MVC design concept and supported web features provide great benefits of standardizing application structure and reducing development time and effort. However, after years of evolution, numerous Java web frameworks have been invented with different focuses, it becomes increasingly difficult for developers to select a suitable framework for their web applications. In this thesis, we conduct a general comparison of four popular Java web frameworks: Struts1.X, WebWork2.2X, Tapestry 4, JSF1.2, and we try to help web developers or technique managers gain a deep insight of these frameworks through the comparison and therefore be able to choose the right framework for their web applications. The comparison preformed by this thesis generally takes three steps: first it studies the infrastructure of four chosen frameworks through which the overall view of different frameworks could be presented to readers; second it selects six basic but essential web features and fulfill the feature comparison by discussing different frameworks' web feature implementation; third it presents a case study application to provide practical support of feature comparison. The thesis ends with an evaluation of pros and cons of different framework web features and a general suggestion of web application types that the four chosen Java web frameworks can effectively fit in.

Key words and terms: Java web framework, MVC, web features, Struts1.X, WebWork2.2X, Tapestry 4, JSF1.2, feature comparison.

Acknowledgement

I would like to thank Professor Roope Raisamo and Professor Jyrki Nummenmaa for being my thesis tutor and examinator. Their patient, responsible attitude and meaningful advice help me steer clear of various serious mistakes and direct successfully my thesis to the final destination.

With this chance I would also like to thanks my parents and my girlfriend Li Meng, their solicitude and encouragement is the foremost factor supporting me to finish this thesis.

Tampere – Finland, June 2008

Peng Wang

Contents

1. Introduction

With the increasing need for the maintainability and extensibility of web applications, it is very important to select a robust, efficient and suitable framework to standardize and bring structure to web application development. Among numerous technologies now existing, many web developers show great preference for the Java web frameworks because of the outstanding design concepts and the popularity of Java programming language. Java web framework is a platform based on Model-View-Control (MVC) design pattern which dictates structure and separates web application into different components to help safeguard it from a potential mess of tangled code. Currently as almost every Java web application adopts Java web frameworks as the implementation of the web presentation tier, the Java web framework has already became an indispensable part of the Java web development.

In the early days of building Java web applications, developers often used JSP *scriptlets* and printed out content they wanted to display directly within their scriptlets—the same place where critical business logic was located. Although to some degree this could greatly reduce the time spending and increase the efficiency of development, it soon becomes clear that this technique too tightly coupled the core business code with the presentation, which greatly limits the readability, maintainability and extensibility of a web application. As the elicitation of the concept: “Web MVC”, it is now possible to divide web applications easily into “Model-View-Controller" three tier structure with each tier capable of being developed and tested independently without affecting each other. Although extra integration work for the different tiers is needed, the benefit we could procure from the separation is incontestable. The first mature Java Web MVC implementation is the “JSP Model 2” structure defined by Sun Microsystem, which has been proved as the foundation of building Java web applications [Ford, 2004].

The success of the Web MVC has triggered a proliferation of the Java web presentation frameworks. During the last few years, there are a glut of Java web frameworks invented, each of which has its special design concept, advantage and disadvantage, it has thus becomes increasingly difficult for Java web developers to choose the right framework to use. Moreover, because of the complexity and distinctness of design concepts between different frameworks, it often takes months for developers to learn a new framework. Considering the time and effort needed to spend for choosing and learning java web frameworks, the term “framework” has

actually turned into a “burden” for project teams. To solve this problem, a few researches related to this field have been preformed such as “Architectural models of J2EE Web tier frameworks” [Timo Westkämper, 2004] and “Art of Java Web development” [Neal ford, 2003]. However, the purpose of these researches is to help readers to understand java web presentation tier development, and although they listed and introduced several popular java web frameworks, not enough feature comparison of web frameworks is provided. In addition, Java open source expert Matt Raible has given several conference presentations for comparing java web frameworks, for instance, “Java web framework sweet sport” [Matt Raible,2006] and “Comparing Java web frameworks” [Matt Raible,2007], although in these presentations pros and cons of different framework features have been pointed out, measurements were restricted to concept discussion, there were no detailed examples and practical issues presented, Indeed developers with little experience of a specific framework can barely comprehend the points referring to that framework.

The goal of this thesis is to help web developers or technique managers gain deep insight of these frameworks through a comparison and therefore are able to choose the right framework for their web application. This work investigates four popular Java web frameworks: Struts, WebWork, Tapestry, and JSF. It focuses on comparing various web features of these frameworks such as “Type conversion”, “Internationalization”, “Post and Redirect” and “Navigation rules”. In addition to the theoretical analysis, a case study web application is also presented to provide practical support for feature comparison.

After the introduction chapter, the background technology information is presented in chapter two which includes a basic introduction of technologies used in Java web, MVC design pattern information and the concept of Java web frameworks. In the chapter three the infrastructure of the four chosen frameworks is introduced, the content includes framework overview, framework lifecycle and core components of the framework, we also give a general summarization of different frameworks at the end. Chapter four discusses the methodology used in this thesis, the case study “Project Track” application is also introduced in this chapter. Chapter five is the core part of the thesis, six web features are discussed for each framework. After the discussion of each web feature, corresponding part of the case study web application is presented to prove the author’s statements. In the chapter six the advantage and disadvantage of different framework’s feature implementation and suitable web application types that different frameworks can fit in are summarized. The last chapter includes a general conclusion for this thesis and some future work.

2. Technology

The main purpose of this chapter is to offer some basic technology background information for this topic. In the arrangement of the content, different Java web technology is first introduced. After that, the MVC design pattern is presented with a focus on the evolution from traditional MVC to web MVC and Java web framework concept is also probed in the rest of the chapter.

2.1 Technology used in Java web

The core technology used in the Java web is JSP and Servlet. However, in order to build an integrated Java web application, the technologies listed below are also needed:

- Java Bean components
- EJB components
- JavaServer Pages Standard Tag Library (JSTL) and Expression Language
- XML language

Figure 2.1 shows the whole structure of the Java web application:

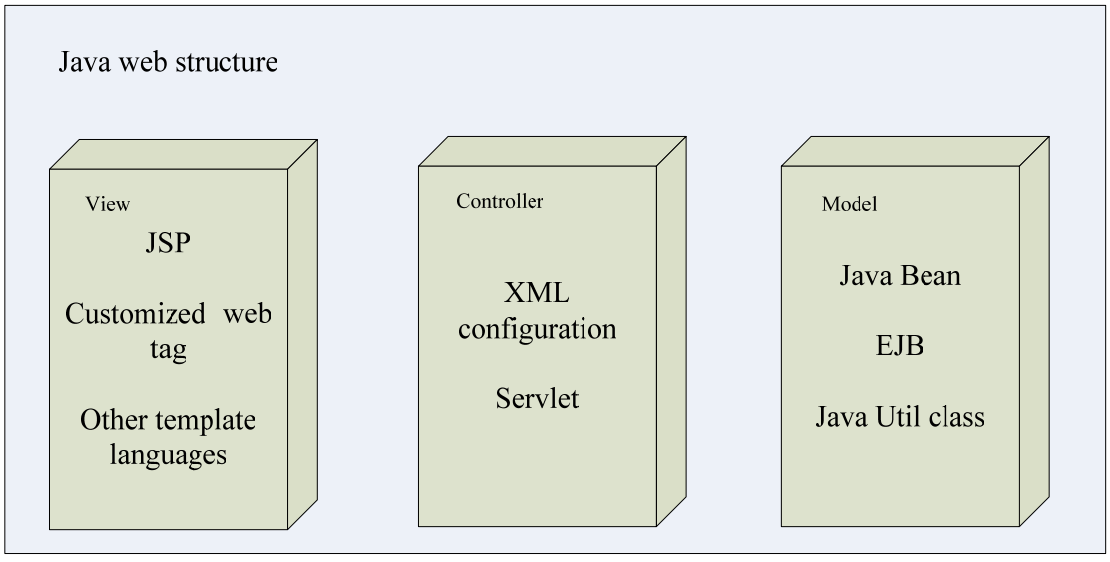


Figure 2.1 Java web application structure

2.1.1 Java Servlet technology

Java Servlet is the most important component in Java web application. It is designed as a general extensible framework and provides a Java class-based mechanism for handling the web request-response mode. Generally a Servlet should only exist in a Servlet container which will dynamically load the Servlet to supply specific service

and extend the functionality of the web server.

When a web client try to visit a Servlet, the Servlet container will first create a “ServletRequest” and “ServletResponse” instance for the clients, then it encapsulates the request information and passes both of the instances to the appointed Servlet. After the execution of the Servlet, the response result will be written into the “ServletResponse” instance and return back to the clients via the Servlet container. The whole process is illustrated in figure 2.2:

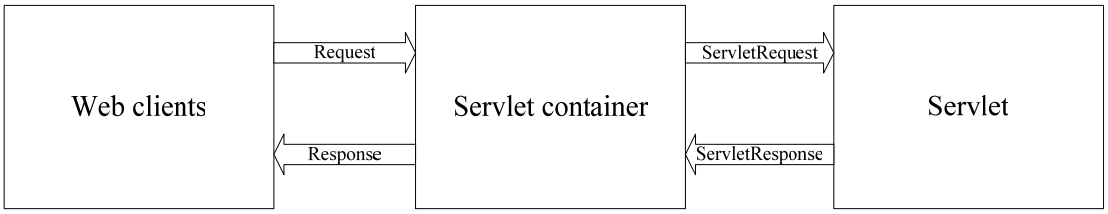


Figure 2.2 The process of Servlet container answering to the web clients

Java Servlet API also introduces “HttpServletRequest”, “HttpSession” and “ServletContext” three classes to store the web shared data, which enable the web clients to save their status and important information within the web scope of “Request”, “Session” and “Application” [Bryson, 2002]. These classes are quite useful because they act as a bridge that transfers the stateless Http connection into the stateful world, later in the chapter five we would discuss how different Java web frameworks utilize their “IoC” feature to make use of these classes.

2.1.2 JavaServer Page

JavaServer Page (JSP) offers a simplified, fast way to create dynamic web content, it was developed in 1999 by Sun Microsystem, Inc and introduced to overcome the problems raised by using the pure Servlet for web applications, such as tedious web content generation and the difficulty of maintenance.

The nature of the JSP is actually a Servlet, the Servlet container will use an internal “JSP Engine” to compile the JSP pages and save them into RAM memory as a Servlet class if the compilation is successful. However, in contrast to the pure Java code Servlet, the JSP adopt a more flexible mechanism –combination of static HTML page, Java scriptlets, JSTL and its Expression Language– to generate the dynamical content for the web clients which is much more efficient and convenient

than directly editing the Servlet java source code. Another difference between Servlet and JSP is that the JSP-Servlet would not be compiled and generated until the first call from the web clients, and if the original JSP page was modified, the container would automatically detect and recompile it without restarting the web application.

2.1.3 JavaBean component

JavaBean is a Java class which conforms to the special standard based on Sun's JavaBean specification [Sun JavaBean Spec, 1997], it provides a series of private properties and defines public accessing method for each of these properties. Originally JavaBean was designed as a reusable software component that can be manipulated visually in a builder tool to make the GUI application more efficient, when used in the web application, JavaBean inherits its original advantage and adds more function support to the special needs of Java web frameworks, for examples "ActionForm" in Structs1.X is implemented as a plain JavaBean class to transfer the data between different tiers of the application.

In JSP page, there are some special tags used to define and visit JavaBean, for instance, if there is a JavaBean named as *CounterBean* and have an attribute *count*, the code below displays the JSP tag grammar of defining the JavaBean and setting and getting the *count* property value:

<jsp:useBean id="YourID" scope="request/session/application" class=" CounterBean">

<jsp:setProperty name="YourID" property=" count" value="0">

<jsp:getProperty name="YourID" property ="count">

When JSP and JavaBean come into play together, the JSP page focuses on the dynamical generation of web content, it supplies web templates for the application data to fit in, whereas JavaBean components offer business logic and data to the web page. By adopting this policy, the reusable characteristic of JavaBean components could be fully used and the web application development could be more efficient and maintainable than putting scriptlets into the JSP page.

2.1.4 Enterprise JavaBeans (EJB)

The JavaBeans that we discussed in the above have little in common with

"Enterprise JavaBeans" or EJB. Enterprise JavaBeans are server-side components with support for transactions, persistence, replication, and security [Horstmann and Cornell, 2004]. At a very basic level, they too are components that can be manipulated in builder tools. However, the Enterprise JavaBeans technology is quite a bit more complex than the "Standard Edition" JavaBeans technology. According to J2EE specification defined by Sun, the EJB components are distributed and must be contained in the EJB containers which could be supplied by the third-part producers and offer the service of security, resource sharing, continuing operations, parallel processing and transaction integration to EJB.

Same as JavaBeans, EJB supplies the business service for the web application, it does not concern with the user view or anything related to the presentation tier. The detailed EJB discussion is beyond the scope of this thesis, the elaborate EJB development technique could be found in the web site of Sun Microsystem, Inc.

2.1.5 JavaServer Pages Standard Tag Library

JavaServer Pages Standard Tag Library (JSTL) is the technology introduced to overcome the serious shortcoming of JSP: mixing presentation and business logic and difficult to understand and maintain. It supports for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags [Sun JSTL, 2003]. JSTL is composed of three different parts: Standard Action Libraries, Tag Library Validators and Expression language [Geary, 2002]. The Standard Action Libraries provide a solid base of functionality for building Web applications, from general actions that iterate over collections and display variable values to more specific tasks such as accessing databases or XML manipulation. Tag Library Validators are used to validate the tag libraries used in JSP pages, they are provided as a proof of function concept and are transparent to programmers. Expression language is the foremost feature of JSTL, it makes easily access implicit objects such as the servlet request and response and scoped variables (i.e. variables stored in request, session, or application scope) in the JSP page. Following are two examples of JSP page showing the request variable “username” value with JSP scriptlets and with JSTL and Expression language technology:

(1) With JSP Sniplet:

```
<%String username = request.gerParameter(“username”);
    Out.println(username);%>
```

(2) With JSTL and its Expression language

```
<c:out value='${param.username}'/>
```

These two examples are simple and only cover little functions of JSTL tag and Expression language, but they do reveal the fact that with an expression language and a comprehensive standard tag library, JSTL nearly eradicates the need for JSP scriptlets and expressions.

In order to increase the page usability and application practicality, most of the Java web frameworks invent their own tag library (customized JSP tag) other than JSTL to display the HTML content, some framework such as Tapestry even gives up the JSP technology totally and turns to the help of new template language. Many Java web frameworks also make use of expression language in their customized JSP tag, for instance, WebWork and Tapestry use the Object Graph Navigation Language (OGNL), and JSF use the JSF Expression Language (JSF EL). Although the grammars of OGNL, JSF EL and JSTL Expression Language are different from each other, they basically fulfill the same responsibility in the Java web application.

2.1.6 XML language

The Extensible Markup Language (XML) is a general-purpose markup language. It is classified as an extensible language because it allows its users to define their own tags. XML language primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet [Wiki XML, 2007]. A XML sentence usually includes a pair of markups to denote the starting and the ending between which the text contents or other XML sentences could be inserted. The following example consists of four XML sentences which represents the communication information.

```
<friend>
    <name> Linda </name>
    <phone> 0442723957 </phone>
    <address> Tampere </address>
</friend>
```

XML files often behave as the configuration files of the software, in the context of Java web application, the “web.xml” file is the one which define the configurations for Java Servlet, Tag library, security, resource reference and some other

initialization configuration. In addition to this file, different Java web frameworks also have their own XML file to configure their specific service, such as “navigation rules”, “JavaBean definition” and “internationalization”. The biggest advantage we could receive from the XML configuration file is that we have no need to modify and recompile the source code once we change the low level configurations, we just need to edit the configuration variable in the XML file and restarting the web application.

2.2 Introduction to MVC

The Model-View-Controller (MVC) design pattern was originally brought forward by Trygve Reenskaug and applied in the SmallTalk-80 environment for the first time, the purpose of it is to implement a dynamic software design which simplifies the maintenance and extension of the software application. MVC seeks to break an application into different parts and define the interactions between these components, thereby limiting the coupling between them and allowing for each one to focus on its responsibilities without worrying about the others [Lightbody and Carreira, 2005]. MVC consists of three categories of the components: Model, View and Controller. This means that it separates the input, processing, and output for the applications and constructs them into a Model-View-Controller three tier structure.

The “View” represents the interactive user interface, when concerning to the web application, it could be generalized to HTML page, or possibly XHTML, XML and Applet. As the complexity and scale of the application gradually increase, the handling to the “View” become challenging because the single application may consist of various kinds of the “View” page, fortunately the processing to the “View” of MVC only limited to the data gathering and presentation, the detailed business logic is left to the “Model” tier, for instance, a “Order” view is only responsible for receiving and presenting the order information as well as transferring the user’s order request to “Controller” and “Model” tier. This view handling policy could set the user interface programmers free from the understanding of the complicated business rules.

The “Model” components is the core part of the application, it contains business workflow, business status and the definition of the business rules. The internal processing of “Model” tier is transparent to other tiers, from “View” and “Controller” point of view, the “Model” tier is a black box which receives the user input and then broadcasts the corresponding result code. The output of the “Model” class should be independent and adiaphorous which means that it can be utilized by different kinds of

"View".

In order to keep the more generally reusable domain model code and the view-specific code from being too aware of each other, MVC also introduce "Controller" component to be the coordinator role for the "Model" and "View" component. The "Controller" component is actually a dispatcher, it decides which business rule to use, which view page to present and how the user request should be addressed. There are mainly two functions supplied by the "Controller": First, it is responsible for interpreting user input and updating the "Model" in response. Second, it registers the "View" to receive notifications of changes to the domain model so that the "View" can refresh itself with the updated data.

2.2.1 Traditional MVC has become outdated

Although the original MVC pattern worked well for desktop GUI applications, it failed to map directly to the World Wide Web [Lightbody and Carreira, 2005]. In the traditional MVC, after the "View" component interacting with the user, typically a button submitting from users, the "Controller" component receives the view-changing events and modified the relative data in the "Model" component, then the "View" component acquires the model-change event from the "Model" and refresh itself to show the updated data to users (Figure 2.1). However, this process is broken when applying to a web application because of different hardware and software structure between a desktop application and a web application. In the web version of MVC the "View" is typically rendered in a browser on the client side, whereas the "Controller" and "Model" are on the server side, the view-changing event can not make the direct access to the "Controller", it has to first visit the web application server by means of URL request to make a handshake with the "Controller". Later after the modification of domain objects, the "Model" component also can not directly notify the model-change event to the "View" component since they are located in different machines, the solution to this problem is that the "Model" component sends the result code to "Controller" which would later reconstruct the application data and select a new appropriate view page to send back to the client side through the web connection.

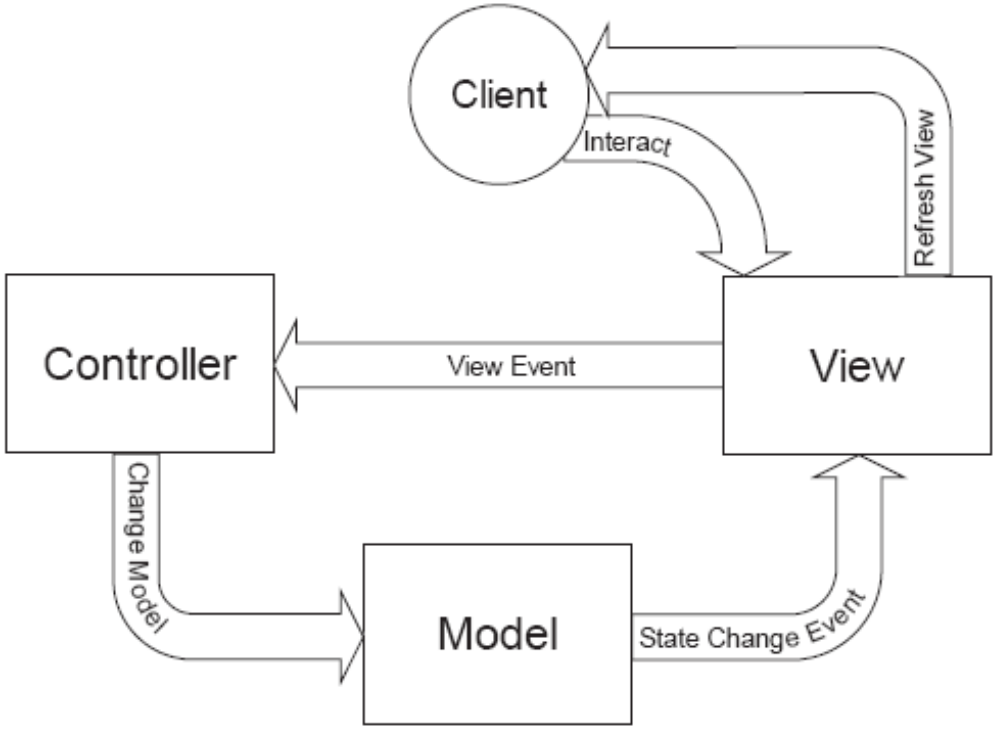


Figure 2.3 Traditional MVC workflow [Lightbody and Carreira, 2005].

2.2.2 Web version MVC: Front Controller Pattern

Most of the Java web presentation frameworks adopt the Front Controller Pattern (see figure 2.2) to handle users' request. According to the description of Sun Microsystem, Inc, the front controller pattern utilizes a core controller, commonly implemented as a Servlet, as the initial point of contact for handling a URL request. This core controller provides a centralized entry point that addresses common services such as security services, delegating business processing, exception strategy and configuration initialization so that the web application could have a centralized, unitive control without resulting in duplicated code.

In the traditional MVC, the view management and navigation is integrated into the "Controller" component, there is no obvious partition between the view management and other functions. In the context of web MVC, the Front Controller Pattern brings forward the "dispatcher" concept and abstracts them out of the traditional "Controller". A dispatcher is mainly responsible for view management and navigation, managing the choice of the next view to present to the user, and providing the mechanism for vectoring control to this resource [Sun J2EE Blueprint, 2005], it could either utilize a static dispatching or a more sophisticated dynamic dispatching mechanism depending on the detail implementation.

To manage the domain model, Front Controller Pattern introduces View Helper Pattern to store the view's intermediate data model and serve as business data adapters. The foremost purpose for applying this “View Helper” pattern is to bring the separation between business logic and view displaying, it helps a view or controller complete its processing without touching upon business model details. Multiple view pages could make use of the same “Helper” class, if they apply for the similar service. In practice, common logics could be wrappered in a single “Helper” class so that the common “Helper” could act as a “second” point of contact to handle the common service specific to some web applications.

Figure 1.2 shows the sequence diagram representing the Front Controller Pattern. When the “Controller” receives UI events such as users’ submit, it will transfer the control right to the “Dispatcher” and “Helper” components, the “Dispatcher” chooses and sends the appropriate view page as a response to the clients according to the result code returned by the “Helper” class. From the figure we could see that the “Controller”, “Dispatcher” and “Helper” together act as the “Controller” role in MVC pattern, and “View”, normally a web page, act as “view” role in MVC pattern.

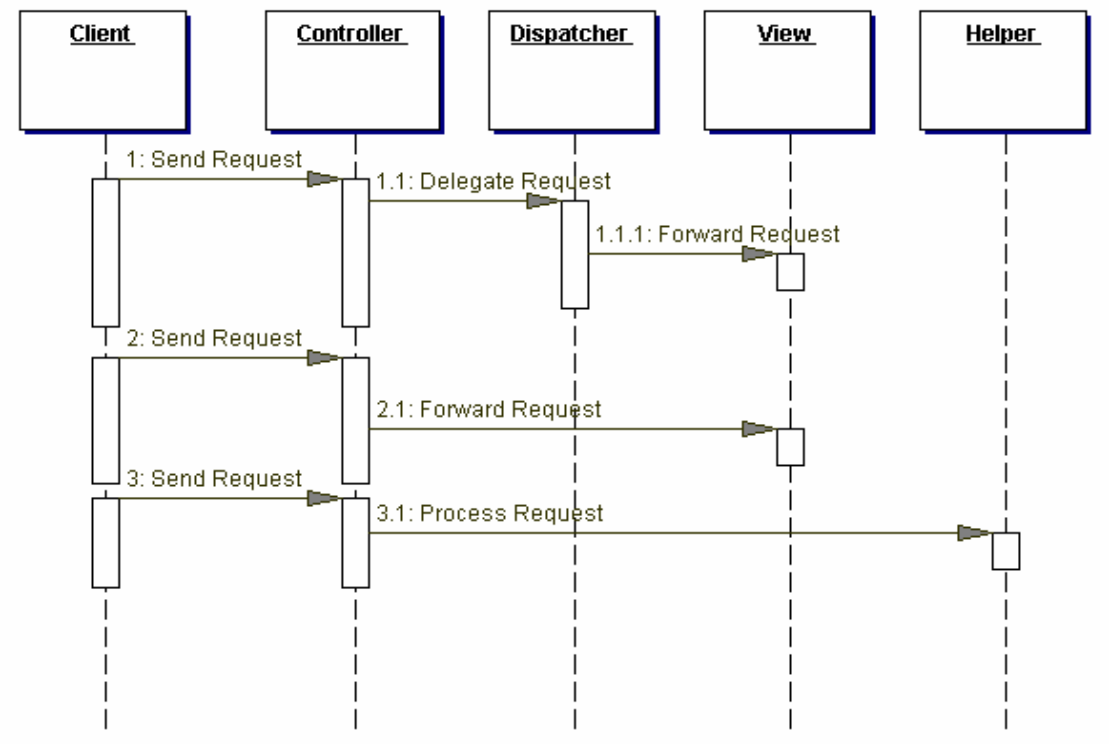


Figure 2.4 Front Controller pattern sequence diagram [Sun J2EE Blueprint, 2005]

2.2.3 Web version MVC involves: Page Controller Pattern

An alternative implementation of web MVC is the Page Controller Pattern, which has been popularized by frameworks like Microsoft's ASP.NET. Comparing to the Front Controller Pattern, the Page Controller requests do not go through a centralized controller, from which the "dispatcher" could be invoked to seek a appropriate view page for the user, instead the view is hit directly and the specific controller for this view will be called to obtain the data from domain model and fill in the content for the view before rendering. Despite this pattern would generate some reduplicate code and somewhat give up the decoupled nature of traditional MVC, it could popularize the concept of web component development due to the fact that the individual "Controller" is closely tied to each view and integrate all the function of "Controller", "Dispatcher" and "Helper" in Front Controller Pattern. The web component development is quite tempting, because it could set the web developers free from the torture of Html and JSP tag editing and gain great convenience and productivity from the help of the powerful modern tools such as Microsoft Visual Studio.

Some of the Java web framework's design concepts are also based on web component development, such as JSF and Tapestry. We discuss those in the chapter four.

2.3 Introduction to frameworks

A framework dictates the overall architecture of the application and predefines features in the form of reusable classes, utility classes, and base classes for developers to extend and utilize. Normally developers just need to fill the vacancy which the frameworks leave for and customize it to their specific needs. Frameworks become popular because they solve common problems in a simplified way and do so without seriously compromising the intent of the application they support [Ford, 2004].

2.3.1 They are the enhancement of JSP and Servlet API

As people build more and more Java web application, it becomes increasingly clear that although the JSP and Servlet API are extremely useful, they can not deal with the common tasks without the tedious code [Mann, 2005], the natural of Servlet API is stateless and operation-centric [ship,2004], it just covers the basic infrastructure necessary for building web applications and offers low level abstraction for the developers, developers always need extra effort to deal with problems such as type conversion, exception handling, internationalization and so on, these problems are where the java web framework set foot in. Comparing to the clean-state Servlet API, frameworks are semi-manufactured goods. All the features of the frameworks are

designed to make it simpler to create robust applications that are easier to construct, debug, maintain, and extend than traditional Servlet applications. The end result of using web framework would be less code and more consistency across the whole application, not just from the developer's point of view but from the end users' perspective as well.

2.3.2 Acting as JSP Model 2

JSP Model 2 is the first successful Java MVC structure which combines the different Java web technologies together. It makes use of JavaBeans to represent the "Model" and utilizes JavaServer Pages (JSP) and Servlet technique to act as "View" and "Controller" role of MVC. (See figure 2.5)

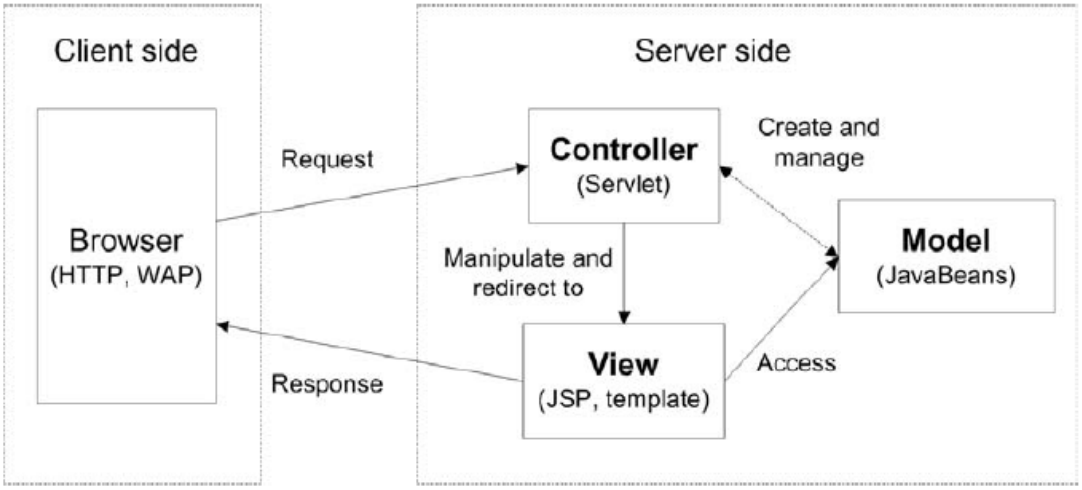


Figure2.5 JSP Model 2 [Mann, 2005]

The Java Servlet controller in "JSP Model 2" takes charge of handling the requests of clients, creating and managing the JavaBean instances and manipulating and redirecting the suitable view pages for the web clients. The view tier, normally implemented as JSP pages (or other template languages), processes no business logic, it only searches the JavaBean instance created by Servlet and put the dynamic contents into the static view templates. This breakthrough design method clearly defines the circumscription between the presentation tier and business domain model and nails down the work division for the web page designers and application programmers, as a result the more complexity the web application has the more benefit it could obtain from the JSP Model 2 structure.

Although the design concepts and the structure of Java web frameworks are diversified, it can not mask the fact that most of Java web frameworks (including the

four chosen framework in this thesis) are built on the basis of the JSP Model 2. They normally reinforce the JSP Model 2 with three aspects: first most Java web frameworks apply the “Front Controller” design pattern which supplies a centralized Servlet to fulfill the “Controller” responsibility. Second, they abstract and encapsulate the raw servlet API to a high level programming API to set web developers be free from the low lever trivial tasks and make them be able to place more concentration on the development of system logic. Last but not least, in addition to use the JSP and JSTL language, frameworks prefer their owe presentation technique, such as customized JSP tag, FreeMarker, Velocity, Web components displaying tag.

The Model View Controller pattern based Java web frameworks hold a lot of potential to make the developer’s life easier, their development time faster, and their application more maintainable. So the time invested in deciding on which framework to use is worthwhile.

3 Infrastructure investigation

To have comprehensive understanding of a framework, it is necessary for us to first be acquainted with the ingredients of Java web framework, the typical characteristics and core functionalities of the frameworks are often contributed by those ingredients and their cooperation. In this chapter, we first give a general concept introduction of the frameworks and then discuss framework's key components that are essential for building web applications, at the end we walk though the framework lifecycle and expatiate how different components works together to help the framework to process a HTTP request.

3.1 Struts1.X

In the past years, Struts1.X is the irrefragable winner of all the MVC frameworks. No matter the market share or the possession of developers, Struts1.X always comes out top and has more tremendous predominance than other frameworks. The triumph of Struts1.X benefits not only from the status that it is the first official MVC framework in the world but also from its elaborate documentation and active development community.

Structs1.X is the typical framework that follows the JSP Model 2 structure, it utilizes a centralized servlet controller, multiple business logic adapters, JSP page and a "structs-config.xml" configuration file to establish the basic the MVC structure of this framework (See figure 4.1).

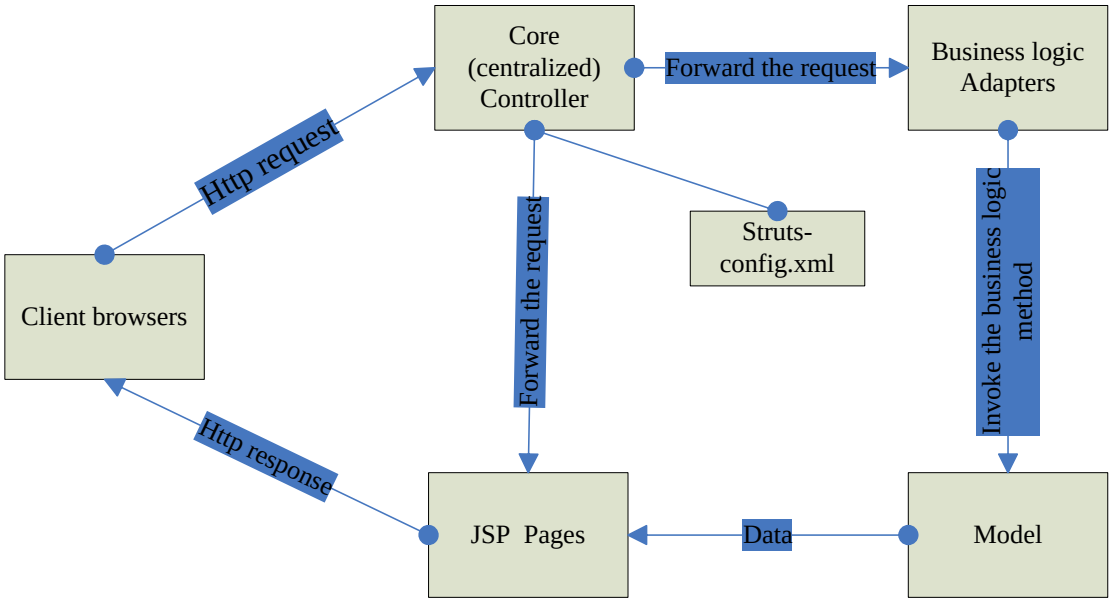


Figure 3.1 Struts1 MVC structure

In the Controller tier

The controller of Structs1.X consists of two parts: the first part is the class "ActionServlet" which is the "Core controller" in the figure 4.1, the second part is the various "Action" class, it correspond to the "Business logic adapter" and should be implemented by the web developers. The "ActionServlet" class extends the "HttpServlet" class and could be configured to a standard "Servlet". As a centralized controller, it heads off all the HTTP requests and decides whether it should redirect the request to the "Business logic adapter" or return the JSP pages directly to the clients. The "Action" classes in Struts1.X are actually the "Command" design pattern implementation of the Java Servlet technology, it connects user requests to the business domain model and supplies the space for invoking business methods in the "Model" tier. In the small-scale web application, detailed business logics and rules could also be implemented in the "Action" class.

In the View tier

As for Struts1.X, the "View" tier mainly utilizes the JSP technology and Struts customized JSP tag (Struts taglib) to present the web content. Struts customized JSP tag is one of the largest advantages of Struts framework, its richful tag library helps to reduce the JSP scriplet and makes a smooth interaction with the business model by means of "ActionForm" JavaBean components.

Although the Struts1.X framework could be integrated to the page decoration framework "Tile", and after year's evolution, it could support Velocity, XLST and Struts Cocoon as the alternatives of JSP pages, the presentation technique of Struts1.X is still monotone. The history reason limits the combination with more advanced view technology and greatly reduces the utilization of the Struts1.X framework, which is the one of the reasons that Struts framework has been updated to the second version.

In the Model tier

Just as most Java web frameworks, Struts1.X does not provide any support in the Model tier, the model tier mainly implemented with JavaBeans or EJB components and offers various business logic interfaces for the Struts "Action" classes to invoke. Because of the Model's neutral and framework-independent nature, I would not discuss Model tier for the rest frameworks.

Struts-configuration.xml

The "Struts-configuration.xml" file is the place where Struts1.X framework stores its

configurations, the most important function for this file is to define the navigation rules for the web application (i.e. how the “ActionServlet” selects the correct “Action” class to invoke according to the request URL), other configurations include database data source configuration, “ActionForm” JavaBeans definition, core controller attribute configuration, internationalization and Struts plug-in configuration.

3.1.1 Struts1.X components

In the Struts1.X API, “Org.apache.struts.action” package contains the core classes of the Struts frameworks, their relationship and collaboration constitute the basic work mechanism of the Struts framework. Figure 4.2 shows the basic constitution of the “Org.apache.struts.action” package.

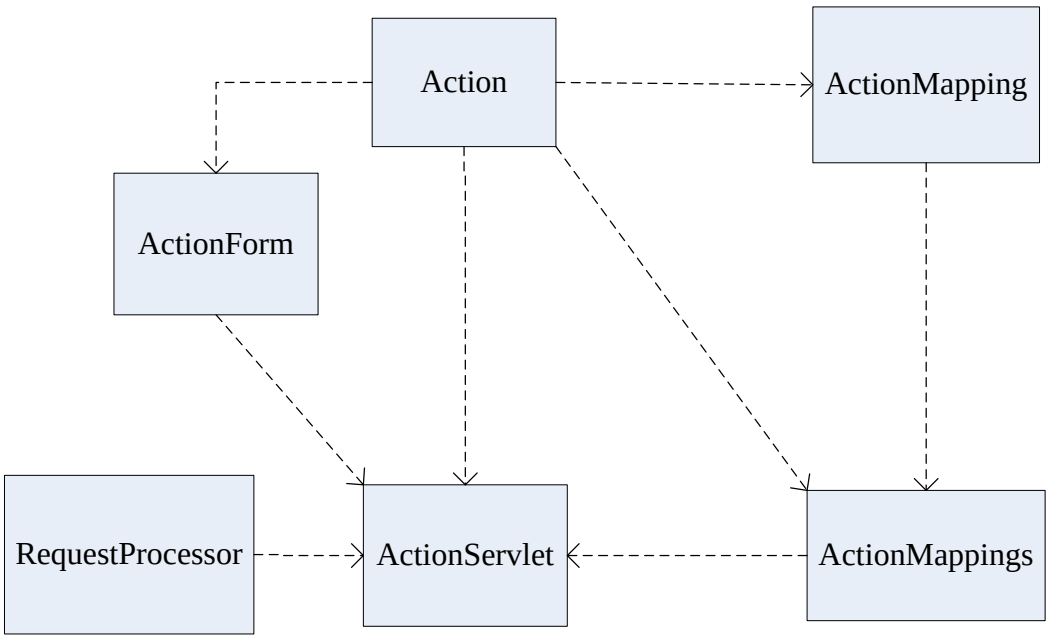


Figure 3.2 simple UML graph for “Org.apache.struts.action” package

ActionServlet

As mentioned before, ActionServlet is the core controller of the Struts1.X framework, it is always the first component that receives the client requests and distribute them to other components in the light of Struts configuration. During the web application life cycle, only one ActionServlet instance is allow to created and used to simultaneously response to multiple requests.

ActionServlet`s initiation happens when the Servlet container starts up, its initial work refers to read the configurations stored in the “Struts-configuration.xml” file

and load them into the memory. The Struts API has one class “ModuleConfig” act as the main container of these configurations, the instances of this class could be used by web developer later to dynamically read or reset the Struts configuration in the web application.

RequestProcessor

As of Strut1.1, the framework introduces the multiple sub-applications mechanism to offer module-division function support to the web application. With the help of this mechanism, the application using the Struts framework could not only logically but also physically define the separate sub-modules by designating the sub-module name and sub-module configuration file in the “web.xml” file, and conforming to the predefined format of the sub-module URL.

In the Struts application, every sub-module possesses one “RequestProcessor” instance, the class path of which is configured in the sub-module configuration file. Once the ActionServlet selects the correct the sub-module for the web request, it will invoke the “process” method of the corresponding “RequestProcessor” instance and pass the current “Request” and “Response” object as parameters. The “RequestProcessor” is actually the main processor of the user`s request, it prehandles and validates various HTTP request information such as “locale” and “Content type”, and invokes appropriate “Action” class` method to address further business details. Generally every sub-module shares the default Struts “RequestProcessor” class as the module- processor, but developers could easily utilize their customized one by reconfiguring the “RequestProcessor” class path in the sub-module configuration file.

ActionForm

ActionForm JavaBean is the Data transfer object (DTO) supplied by Struts framework. It is used to transfer the HTML form date between view tier and controller tier. The controller could not only read the submitted data from “ActionForm” and pass it to Model tier, but also put the model data into the “ActionForm” and via which pass to the view tier. “ActionForm” also has the validation function for the submitted data, before the controller hand over the control right to the “Action” class, the “RequestProcessor” will invoke the validation function of “ActionForm” to check and leach the invalided form data.

“ActionForm” has two kinds of existing scope: “Request” and “Session”. If “ActionForm” exists in the “Request” scope, it can only be valid in the

request-response life cycle which means it becomes invalid after the controller send the response to the clients, next time when the clients send the new request or revisit the old pages, the controller will create the new “ActionForm” instance to carry the form data. Conversely, when existing in “Session” scope the “ActionForm” will be valid across the whole HTTP session process.

Action

“Action” class is the bridge between the user request and the business domain model, it encapsulates various business rules and transform web requests to different business service invoking. For Struts developers, in order to use the “Action” class they must inherit the abstract “Action” class and customize it by overriding the “execute” method which is invoked by sub-module’s RequestProcessor instance and passed with “ActionForm” parameter.

In the entire Struts application life cycle, each “Action” class could only have one instance which will be shared by all the “Action” visitors, as a result it is meaningless to define the global attributes of the “Action” class since one client’s modification could almost simultaneously be changed by others. To solve this thread-safety problem, web developers should only define local variables in the “execute” method so that each request thread will have their own local variable value and avoid being shared resource by other threads. In the situation that sharing resource is necessary to the application, the developer should make use of the Java synchronization mechanism to control the conflict.

ActionMappings and ActionMapping

“ActionMapping” contains the single reflection information between URL and “Action” class, it includes “Action” class path, input page name, forward page name, URL forwarding string and other information configured under the “<action>” tag of the Struts configuration file. When a user makes a request, the RequestProcessor will find the corresponding “ActionMapping” instance according to the request URL and pass it as the parameter to the “Action” execute method.

“ActionMappings” is the collection of the “ActionMapping” instances, it represents all the URL-Action reflection information. Just like other configuration information, it is created during the initiation of the ActionServlet and stored as an attribute in a “ModuleConfig” class instance which represents the memory form of all the XML reflection configuration. The “ModuleConfig” exists in the “application” scope which means it is valid across the whole application life cycle and can be

dynamically visited in the application “ServletContext” instance (see chapter 2.1.1).

3.1.2 Struts1.X workflow

Before Struts1.X web application receives any HTTP requests, the first thing the framework do is to init its core controller “ActionServlet”, “ActionServlet” will read various the configuration information into the memory and save them into different container classes, for instance, the “Action” reflecting information will be stored into the “ActionMapping” class. All these container instances will be at the end store as an attribute of the “ModuleConfig” class instance.

When the “ActionServlet” receives a web client’s request, the Struts1.X framework will first find the corresponding “RequestProcessor” instance of the specific sub-module and delegate it to handle the HTTP request and its head information. Then it checks whether there is a “ActionMapping” instance matching the client`s request path. If there is no such instance existed, then the framework returns the invalid request path information to the client. Next, it save the submitted form data into the corresponding “ActionForm” instance (if it is not existed, then create a new one) and invoke the validate() method of it. If there is any error happened during the “ActionForm” validation, the process will not go further and the JSP page where the client submitted will be returned back to the clients again. If everything goes successfully, the corresponding “Action” class will be invoked and result codes returned by action execute() method will be used by “ActionServlet” controller to find a suitable JSP page and returns it to the client. Figure 3.3 shows the whole responding process.

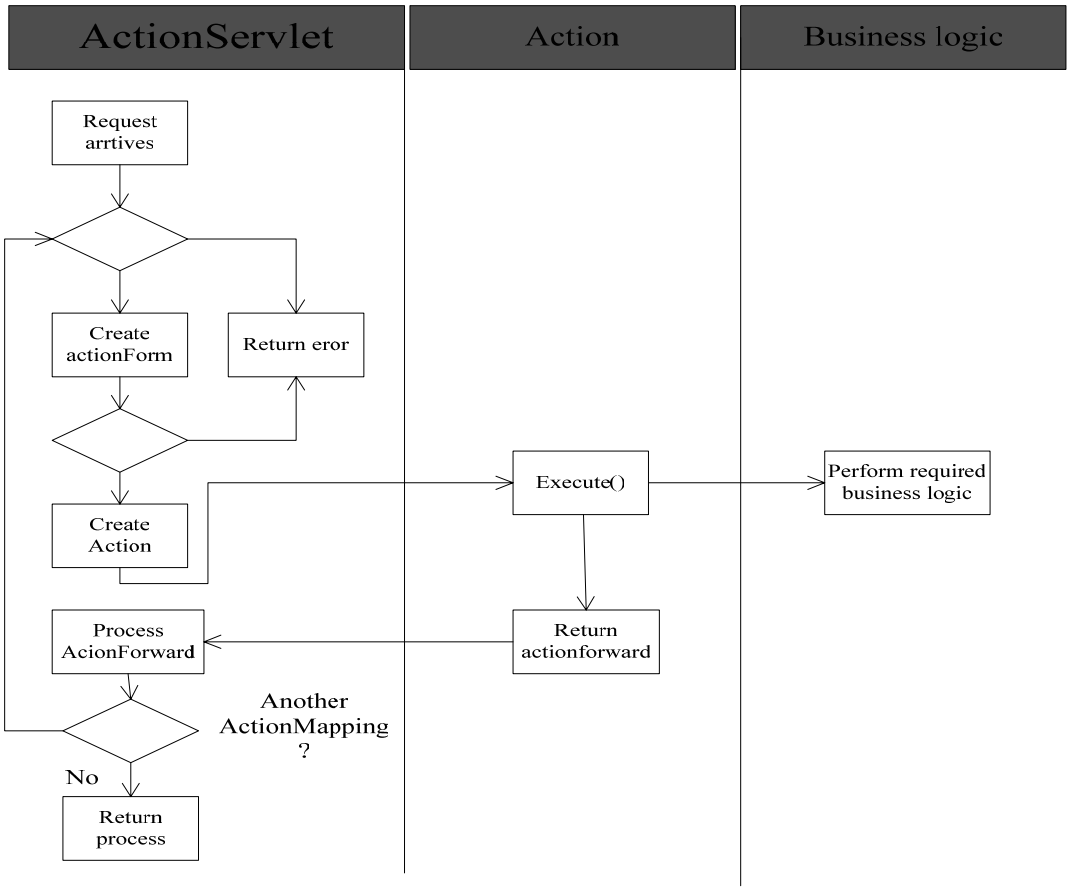


Figure 3.3 Struct1.X responding process [Sun, 2004]

3.2 WebWork2.2.X

WebWork is a Java web framework produced by the OpenSymphony open source organization and applying itself to “Pull Hierarchical Model-View-Controller” structure [Lightbody and Carreira, 2005] and test-driven development. After a series of evolution, WebWork framework has divided itself into two different parts: web-unrelated part and web-related part. The web-unrelated part, XWork, is the core components of the framework, it is implemented with standard “Command” design pattern and supplies crucial functions to the application development such as interceptor， type conversion, Inversion of control (IoC) and so forth. The web-related part, WebWork, is built on top of the XWork, this part utilizes a centralized controller to interact with the web clients and encapsulates various web-scope data (request, session, application) into a “Map” structure which will be transferred to XWork with client`s requests later. This division sets the XWork free from the awareness of low level Servlet API and makes the real business processing part easier to test. Although XWork is an important and critical part of the framework, developers probably won’t need to know the two part’s difference unless they plan to

dig deeply into the core implementation of both projects, so in order to avoid confusion, in this thesis I discuss this framework as a whole, the term “WebWork” would simply mean both parts.

The architecture of WebWork follows the common architecture of most Model 2 web application frameworks. Figure 4.4 shows the overall architecture and flow.

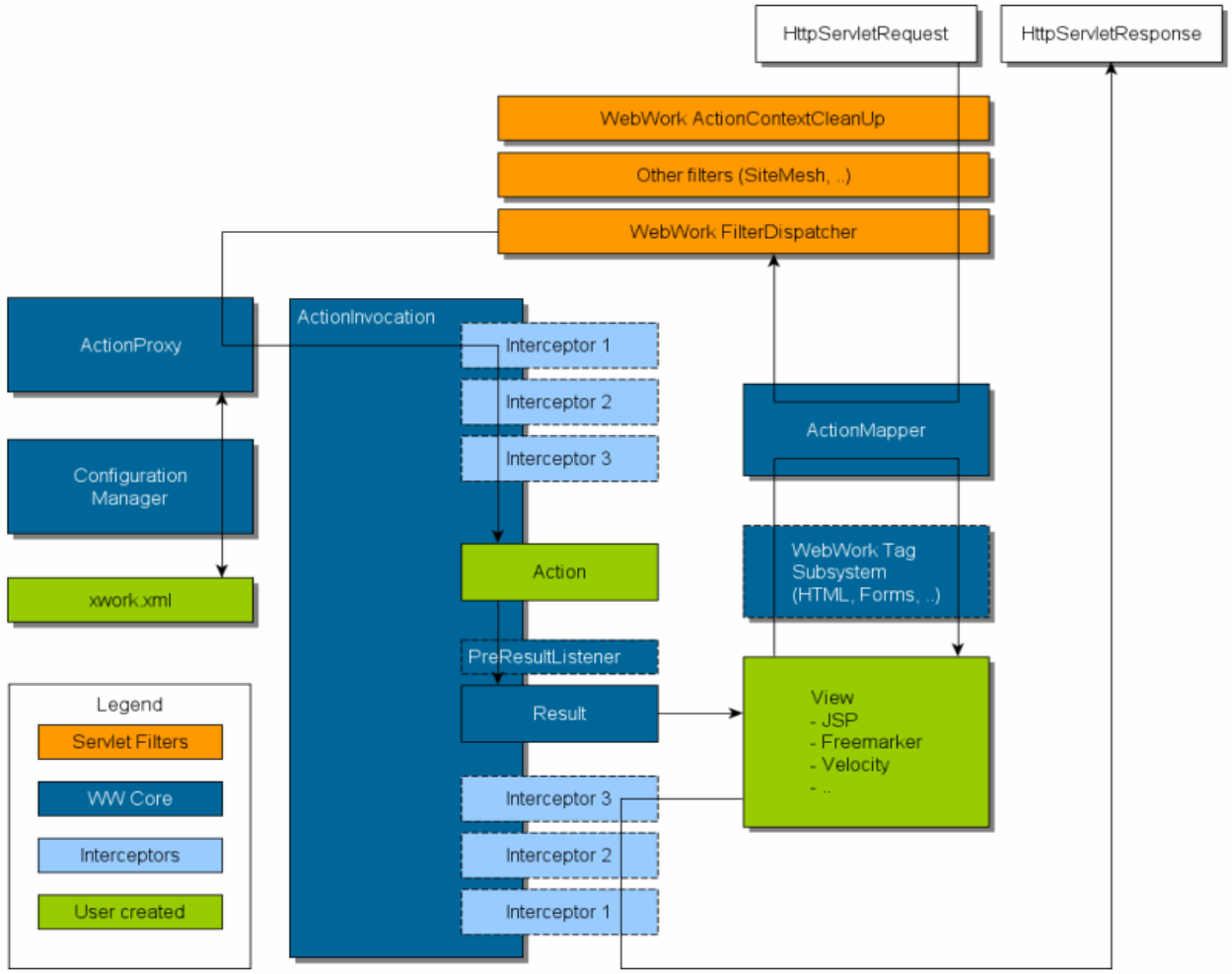


Figure 3.4 WebWork2.26 architecture [Opensymphony WebWork Wiki, 2006]

In the controller tier

The “FilterController”, “ActionMapper”, “ActionProxy”, “ActionInvocation”, “Result” and “Action” together constitute the “Controller” role of the MVC.

The “FilterController” is main controller of the WebWork, in addition to interact with clients, it is also in charge of reading the configuration file and setting the work environment for the application. The “ActionMapper” is responsible for providing a mapping between HTTP requests and action invocation requests and vice-versa. [Opensymphony WebWork Wiki, 2006]. “ActionProxy” and “ActionInvocation” are

the main workflow controllers, they guarantee the correct work sequence in WebWork. “Action” is the business logic adapter, it is quite similar to the one in Struts1.X but more flexible and decoupled from Servlet API. The “Result” is introduced to manager the transferring mechanism between the “Action” class and view pages.

In the view tier

In addition to JSP, the WebWork2.2X support seven other kinds of displaying techniques, they are “Velocity”, “Freemark”, “XSLT”, “Plain text”, “Jasper report”, “HTTP header” and “Stream”, these technique offer more choices for web applications to present their domain model data.

Same as Struts, WebWork also support his own customized JSP tags, they normally start with prefix “ww”. WebWork tags are spited into two groups: non-UI tags and UI tags. Non-UI tags assist with control flow and data access. UI tags are used to build consistent user web interfaces. These tags could be placed not only in JSP page but also in “Velocity” and “Freemark” template language.

Pull Hierarchical Model-View-Controller

WebWork enforces the normal semantics of traditional JSP Model 2, but with a different twist on how that model information is made available [Ford, 2004]. There has been embodied by the two words “Pull” and “hierarchical”. In Neal Ford`s book, *Art of Java Web Development,* he stated that the “pull” part of this definition indicates that the view component is responsible for pulling the model information from the controller on demand. This is different from the traditional Model 2, where the view accesses information that has been placed within the model and passed to it from the controller. In this case, the WebWork framework no longer needs the intermedium DTO to be the information carrier that transfers the data between “View” and “controller” tier, it could access the information actively using WebWork expression language “OGNL” without necessarily having to wait for a controller to make it available. Ford also explained that the “hierarchical” describes the repository of view data. In the case of WebWork, the “value stack” is used to provide information to the view, correlative model data and web scope data (request, session, application) will be put into the “value stack” for the view “OGNL” expression language to dynamically visit.

3.2.1 WebWork key components

From the developer's point of view, with the similar development process of Struts1.X they could always get benefits more from numerous feature supports when using the WebWork framework, this is owed to the cooperation of WebWork internal components. These components supply encapsulation of low level Servlet API and predefine numbers of "plug and play" web service for web development so that developers can utilize them on demand with a little of extra XML configuration. Comparing to compulsory manual work in Struts, this mechanism truly bring much convenience and preponderance for the web development.

FilterDispather

The FilterDispatcher is the main entry point of requests in WebWork, it serves as the adapter between the HTTP request-response world and the generic "Command" pattern Action-Result world of WebWork. When first started up, it reads configuration files "webwork.properties" and "velocity.properties" to set the working environment and initiate "Velocity" template language engine for the web development. If HTTP requests visit the application, it is also responsible for analyzing URL path and creating the context (ActionContext) for executing an action, the control right will be finally transfered to "ActionProxy" which is created by "FilterDispatcher" and passed with URL path and "ActionContext" information.

ActionMapper

The ActionMapper is responsible for providing a mapping between HTTP requests and action invocation requests and vice-versa [Opensymphony WebWork Wiki, 2006]. It is the first gate that judge if the request URL would invoke an action invocation that the WebWork framework should at least to try. The default ActionMapper implementation in WebWork use the standard extension pattern (*.[ext]) to make judgment for the URL, normally the "ext" equal to "action" which is configured in "webwork.action.exection" field in the "webwork.properties" file

ActionProxy/ActionInvocation

The "ActionProxy" serves as a proxy of client codes to execute an action. Because "Action" classes are executed through the framework rather than "Action" instances itself, so WebWork makes use of "ActionProxy" to encapsulate extra functionality of the "Interceptor", "Result" to embellish the execution of the "Action". The corresponding "Action", "Interceptor" and "Result" information are located in the navigation configuration file "XWork.xml" and they are read by "ActionProxy" via a configuration manager. Depending on the configuration in the "webwork.properties"

file, the framework could visit “XWork.xml” file for every request or could visit just once and cache the navigation information for later using.

“ActionInvocation” is a class instance contained by “ActionInvocation” which represents the current state of the execution of the action. It holds the all of the configuration information and utilizes a masterly algorithm to guantee the framework work in the Interceptors-Action-result-Interceptors sequence (see figure 4.4), it is the main workflow controller of the WebWork framework.

Actions

Action class is the core function unit of the framework, it is the “Command” pattern implementation of the WebWork framework and its “execute” method is the default function entry point to business domain data. Comparing to Struts1.X “Action”, “Action” in WebWork are more flexible and framework-independent. First, it does not need to inherit WebWork build-in classes which prohibit users’ customized inheritance. Second, it is more like a plain JavaBean class which eliminates the desire of coupled parameters of framework. Below is the comparison for the Struts1.X and WebWork “Action” entry method declaration.

Struts1.X:

```
public ActionForward execute(ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException
```

WebWork2.2X

```
public String execute() throws Exception
```

Interceptors

Interceptors are one of WebWork’s most powerful features, it allows developers to encapsulate code to be executed before or after the execution of an action and they also let developers modularize common code out into reusable classes [Lightbody and Carreira, 2005]. Interceptors are defined outside the action class, but have access reference to the “Action” class and the “Action” runtime execution environment, in addition, they are implemented as “plug and play” services, web developers have the right to designate the specific interceptors for each action to avoid unnecessary service for their actions.

Many of the core features of WebWork are implemented as interceptors, including parameter setting, chaining action properties setting and internationalization setting. Developers could also define their own customized interceptors by implementing the default “Interceptor” interface defined by WebWork framework.

Results

The “Results” represents a general consequence of the execution of an “Action”, theoretically “Results” can produce any kind of output needed from the action execution in WebWork framework, such as displaying a web page, generating a report or send a email. Currently WebWork support ten types of results for mapping the result code in the “Action” configuration. They are Servlet dispatcher, Servlet redirect, Velocity, Freemark, JasperReports, XSTL rending, Action chaining, plain text and http header. Developers could also define their own result type by implementing the default “Result” interface defined by WebWork framework.

Configuration files

There are three configuration files that developer should configure before the web development.

webwork.properties

This file is used to define application-wide settings and configure parameters that change the behavior of the framework.

XWork.xml

The framework navigation rule is defined in this file, the content includes “Action” profile information, Interceptor configuration and result mapping.

Velocity.properties

This file is used to define “Velocity” macros libraries, it is used when developers utilize “Velocity” template language and define their own macros in separate library files.

3.2.2 WebWork2.2X workflow

The main workflow of WebWork framework is illustrated in Figure 4.4, in the diagram when a web request goes into the Servlet container, it will first go through the filter chain, if the web application has been integrated with page decoration framework “SiteMesh”, the web request must go through the

“ActionContextCleanUp” filter which is used to tell the main controller “FilterDispatcher” the exact time to clean the request. Next, the required FilterDispatcher is called, which decides whether it should delegate the “ActionProxy” to handle the rest of the work according to the URL request judgments of the “ActionMapper”. If the request is qualified to invoke an action, the FilterDispatcher will create the “ActionProxy” and wrap low-level Java Servlet information (so called “ActionContext”) into it. Subsequently, the “ActionProxy” visit the “XWork.xml” via the configuration manager and create an “ActionInvocation” with the information it has.

As mentioned before “ActionInvocation” is the main workflow controller of the WebWork, it first invokes the various predefined interceptors and finally to the requested “Action”, when the “Action” finish its execution, the returned code will be used to find the proper “Result” in the light of action mapping information, then the “Result” is executed and the interceptors will be invoked again in the reverse order of before. Finally “Result” view will be returned to the web clients in the form of “HTTPSevletResponse”.

3.3 Tapestry 4

Tapestry is a component-based Java Web framework, it effectively hides the web topic such as URLs, request parameters and other trivia of HTTP and utilizes a page-based object model to simulate traditional graphical user interface development. Developers coming from a desktop development background could easily find that when using Tapestry framework they can still capitalize on their skills without getting too far into web-specific APIs.

Comparing to operation-based framework such as Struts and WebWork, Tapestry has a complete different development concept and process. There are three core concepts related to Tapestry development environment: Page, Template and Component. The “Page” is the basic unit of the Tapestry application, each Tapestry “Page” is compose of several “Component” and should be only reflected by one “Template”. The “Template” is the descriptor the “Page” structure, it consists of standard HTML markups and special tags used to specify the different “Component”. The “Component” represents reusable objects in the Tapestry “Page”, when the “Page” renders itself the “Component” would be converted to corresponding HTML code, which means that the nature of the “Component” is the encapsulation of the HTML tags collection and a bunch of properties of the component. In the background, there

is always an a java object reflected to each Tapestry page, this page object acts as the gate between the View tier and Model tiers, it defines attributes and methods used to set and get “Component” properties, and it also defines the event-handling code used to make response to the various Tapestry “Component” events.

Figure 4.5 shows the overall structure of Tapestry Framework:

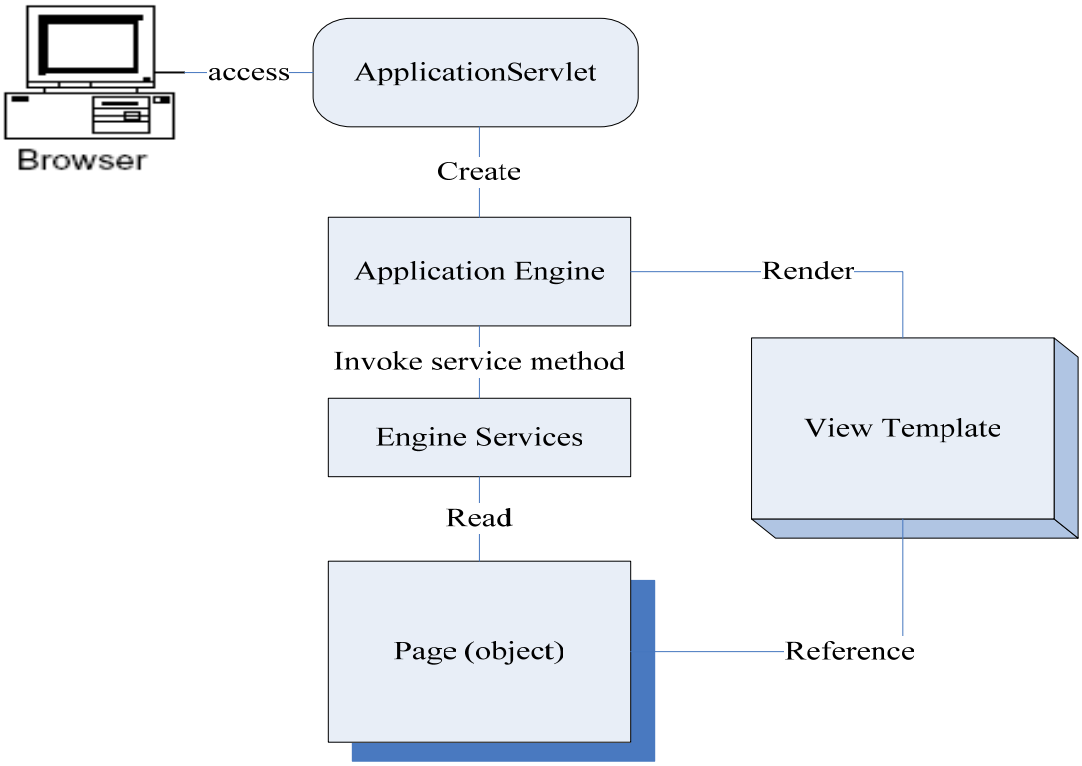


Figure 3.5 Tapestry overall structure

In the controller tier

ApplicationServlet, Engine, EngineService and the Page object constitute the “Controller” role of MVC. Same as other frameworks, Tapestry utilizes a centralized controller “ApplicationServlet” to interact with clients. However, it does not perform any utility functions but transfer the request to the application “Engine”. The “Engine” is the main processor of the Tapestry application, it parses the request URL and select a suitable service handler, an “EngineService” object, to address the URL. The “EngineService” is mainly used to address different services defined in Tapestry which are identified by the service parameter in the Tapestry URL. The page object, as mentioned before, establishes relationships between the “Model” and the “View”, the tapestry view “Template” use the OGNL expression language to obtain the component properties defined in the page object.

In the view tier

The Tapestry supports its view presentation tier by means of individual “Component” parsing and “Page” self-rendering. By default, Tapestry generates a “HTMLwriter” instance and passes it to the “Page” render method to ensure every “component” in the “Page” will be parsed and converted into HTML tags. However, HTML is not the only choice for Tapestry, Tapestry is designed to be compatible with XML, WML and XHTML and developers could override the default “Page” getResponseWriter() method to create a customized web page writer for their specific needs.

3.3.1 Tapestry 4 key components

The basic concept of “Page”, “Component” and “template” is well enough to deal with simplest web applications of Tapestry. However, in order to build more ambitious things with Tapestry it is necessary to have a good understanding of the internal components and how they operate within the Tapestry context.

ApplicatonServlet

In Tapestry, the “ApplicationServlet” is just a gateway between the stateless, multithreaded world of the HTTP protocol and the stateful, single-threaded, component-based Tapestry world. It is only used to head off web requests, find the “Engine” instance in the HttpSession scope (or create a new instance) and invoke the service() method on the instance.

Engine

The real work of Tapestry is done by the engine’s service() method, which is the place in which incoming requests are processed and results are returned to client web browsers. Inside the method, there is another layer of delegation: the Engine Service. The Engine will first do the preparation work for the Engine Service which includes initializations, creating and configuring the subsystems of Tapestry and multiple levels of exception catching and reporting [Ship, 2004] and then delegate to the Engine Service to begin the real request processing.

Engine Service

Within the context of Java web framework, the way of handling the request URL determines the framework workflow. For frameworks such as Struts, WebWork and JSF, they all utilize the similar URL format to trigger the action of the framework and then make use of the navigation configuration file to control the application working route, which is the chief reason that lead to the exclusive application life cycle.

Comparing to these frameworks, Tapestry completely abandons the way of utilizing navigation configuration, it makes use of a different, serviced-based mechanism to construct and process a URL and thus have various life cycles corresponding to URL that have different service parameters.

Tapestry includes a default roster of nine services (shown in Figure 4.6), three of which (home, page, direct) are commonly used. We will discuss this three common services as well as their life cycle in the next section.

Service	Description
action	Invokes a listener method indirectly by rewinding the entire page, not just the contents of a form. Rarely used; the direct service is more efficient.
asset	Dynamically downloads an image or other asset file stored on the classpath (inside a JAR file). Automatically used with component assets from the framework or from a component library.
direct	Allows a component to directly respond to a request; used with most links and forms.
external	Allows "bookmarkable" links to particular pages.
home	Default service, used to display the Home page of an application.
page	Responds with a particular page within the application.
reset	Discards all cached data, including all specifications and templates. Used only during development.
restart	Discards the current session and displays the Home page.
tagsupport	Special service used by the Tapestry JSP tag library to generate URLs for inclusion in a JSP.

Figure 3.6 Tapestry engine services [Ship, 2004]

Page and Page pool

Tapestry makes heavy utilization of the "Page" object since it is the footstone of the framework and the main coordinator between the application "Model" and "View" tier. However, it is also a complex entity which is very expensive and complicated to create. There are several steps the Tapestry framework needs to perform when creating a "Page", these include loading and parsing page specification (The unreleased Tapestry5 will give up the xml page specification file and use the Java Annotation instead, or we can also describe the "Component" directly inside the "Template" file), initiating customized page object, reading page template and nested parsing and loading the components inside the "Page". All of these works make the "Page" object a scarce resource and be worthy to be kept and saved for later use.

Tapestry adopts the page-pool pattern which has the same principle of the database

connection pool for its “Page”. When the clients make the requests for a “Page”, it is obtained from a central page pool. If the pool contains no such “Page”, then a new “Page” instance is created. If a usable “Page” is in the pool, it is transferred to the clients and removed from the pool for the duration of the request. All the “Page” objects will be returned back to pool when the request-response lifecycle is over. The developers do not need to worry about the page pool details, the Tapestry will take care of those in the background. However, the only thing developers need to pay attention is that since the same “Page” object and thus the same “Component” properties could be shared by multiple clients asynchronously, it is maybe not safe to save the person-related information inside the “Page”, Tapestry solves this by initializing all “Component” properties when the “Page” is about to send back the page pool.

Sometimes it is necessary to share some “Component” properties throughout the individual client’s visiting session, to address this problem Tapestry introduces the “persistent page state” concept which is another important issue of the Tapestry framework. The “persistent page state” means that Tapestry will mark the sharing properties as “persistent” and save them into the client`s HttpSession web scope in order to restore from it for the later request. The division of “page” objects and persistent page states set the Tapestry free from the saving the whole “Page” instance into the session scope to store the “Page” state, which is a disaster since one “Page” object could contain unlimited number of nested “Component”. Without this separation Tapestry couldn’t make any claim to efficiency. With it, Tapestry can manage complex server-side state simply and effectively [Ship, 2004].

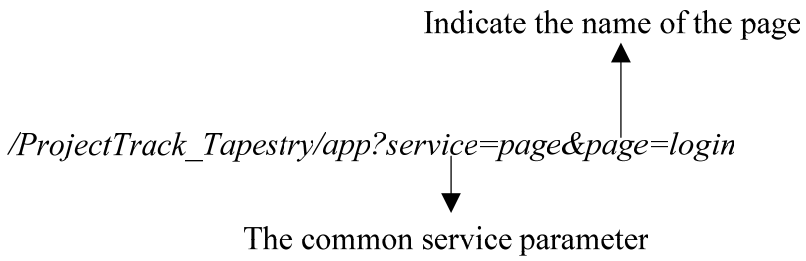
3.3.2 Tapestry 4 workflow

At the core of the Tapestry workflow is the request cycle. This request cycle is so fundamental that Tapestry utilizes a specific class, which implements the “IRequestCycle” interface, to represent it, and it is used throughout the whole HTTP request-response life cycle. Each Tapestry service makes use of the request cycle in its own way, which leads to the various web workflows in the light of services. In this section we discussed the three common services: home, page and direct.

Page service

The page service is the basic service used for rendering a page, it supplies the way of navigation between pages.

The page service URL is specified by two parts, the first part is the common service parameter which is used by all kinds of Tapestry URL to designate the name of service and thus in this case it is “page”; the second indicates the name of the page. Below is an example of the page service URL:



The page service behavior is illustrated in figure 3.7:

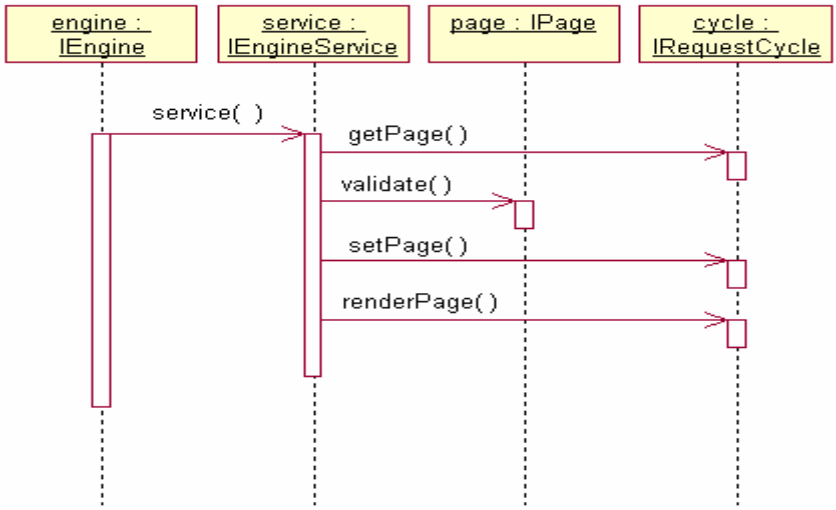


Figure 3.7 page service sequence [Apache Tapestry, 2003]

After the “Engine” discern the page service and delegate to the “Engine service”, the “Engine service” will first get the page name from the “Request cycle” object, the page is then given a chance to perform security check by invoking validate method in the “Page” object, it can throw “PageRedirectException” to stop the current “Page” processing and turn to render a different page. Otherwise, setPage() is called to tell the request cycle the page that need to render and renderPage() peforms the actual render.

Home service

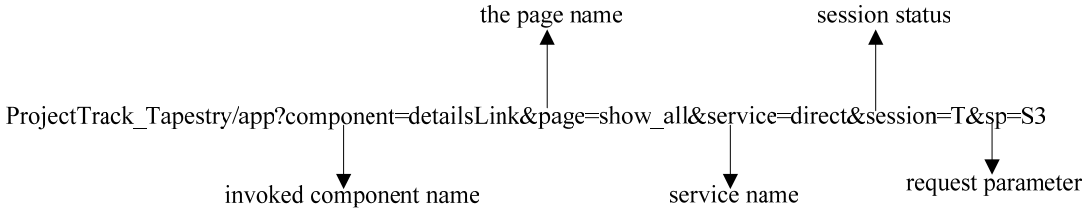
Home service is the default service of the Tapestry framework, to invoke the Home service the web clients could simply use the */web application name/app?* URL without specifying the service parameter and others. Home service is actually a

particular page service which specifies the “Home” page as the rendering page. The workflow of Home service is quite same as the page service except that page name is predefined as “Home”.

Direct service

The direct service is the most frequent service used in the Tapestry applications, it is used to trigger a action defined by “Component”, either a form component or a directlink component.

The direct service URL is a little more complicated than the one of page service. In addition to the common service parameter, the direct service URL has other four parts of parameters which respectively indicate the page name, invoked component, session status and the customized request parameters, below is the example for direct service URL:



The direct service behavior is illustrated in figure 4.8:

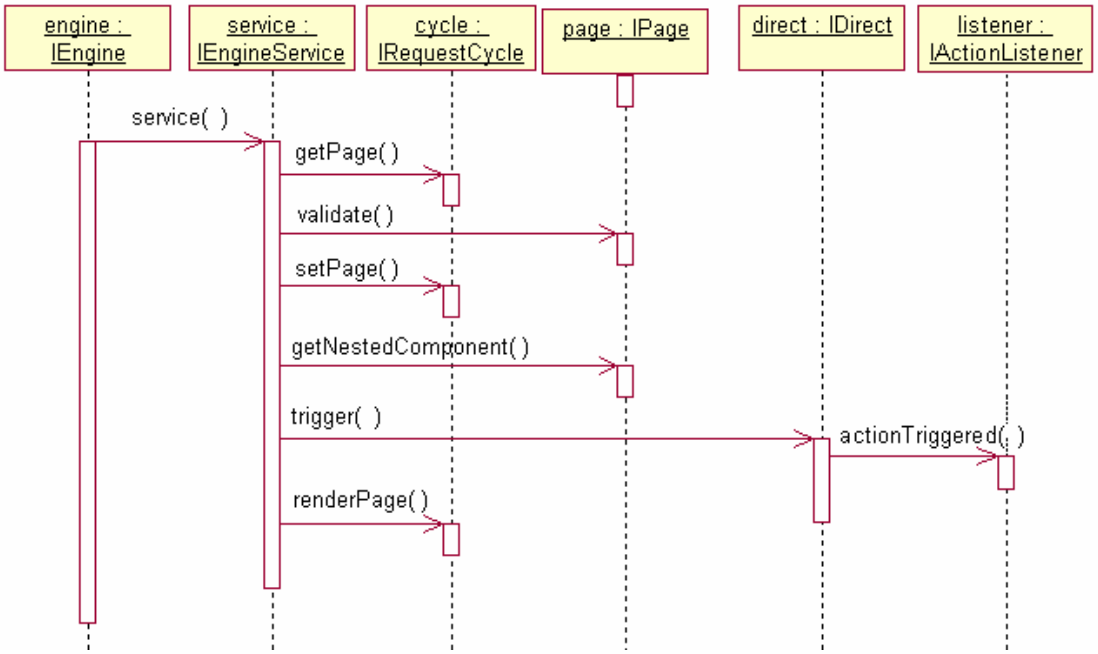


Figure 4.8 direct service sequence [Apache Tapestry, 2003]

Like the page service, the direct service begins by getting the page. The validate()

method is invoked on the page; then the component is located within the page. The component has to implement the interfaces “IDirect” (In rare cases, they should use a separate class to implements the interface and invoke the trigger method defined by it). The real action code is located in the class implementing “IActionListerner” interface, which in the normal case is still implemented by the form or directlink component. After executing the action method, the “Page” designated by the URL will be rendered by the Engine service.

3.4 JSF 1.2

Similar to Tapestry, JSF framework also supports a component-based approach to the web development, where most commonly required functionalities are encapsulated into the components and can be reused in different context. However, comparing to the Tapestry’s page-based mechanism, JSF utilizes the “managed beans” as the background support for its components. The “managed beans” is a JavaBean class where JSF components could find their dynamic properties and action execution code, and it could be shared by multiple view pages and components, which result in form-centric (one bean per view) and object-based (multiple beans per view) two development approach choices for JSF [Mann, 2005].

JSF framework is basically consists of three different parts: a standard set of UI components, a component architecture and an event-driven programming model. The standard UI components are the encapsulation of the standard HTML elements such as buttons, hyperlinks, checkboxes, text fields, and so on, they are cooperated with “managed beans” and could be configured either in the view page or in the “managed beans”. The component architecture defines a common way to build UI widgets. This architecture enables standard JSF UI components, but also set the stage for third-part components. JSF also contains all the necessary code for event handling and component organization. However, application programmers can be blithely ignorant of these details and spend their effort on the application logic in the “managed beans” class [Geary and Horstmann, 2007].

Figure 3.9 is the UML graph of the detailed composition of JSF framework:

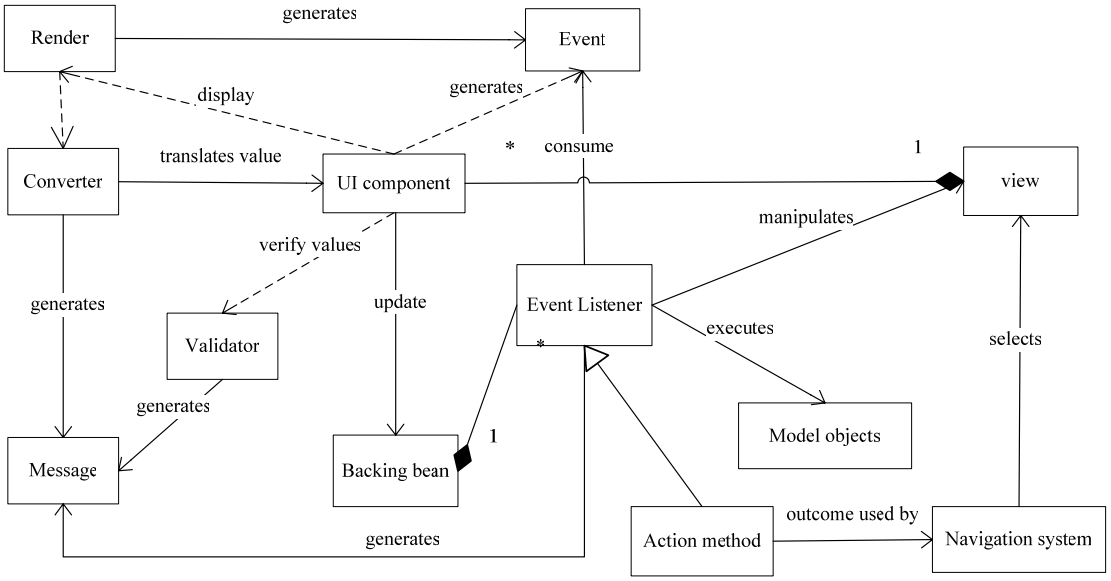


Figure 3.9 a model of how JSF component related to each other [Mann, 2005]

In figure 3.9, there are many components contributing to core features of JSF, however, the key components that control the essence of the framework are the UI component, View, Backing beans (managed beans that have special components binding in it) and Navigation system, others enhance and complement those key components in different aspects.

In the controller tier

The FacesServlet, managed beans and the navigation system constitutes the controller role of the MVC.

The “FacesServlet” is the centralized controller of the JSF framework, it communicates with the clients and control the main workflow of the framework. The “managed beans”, as mentioned before, is background support for components properties and action code. The navigation system is basically same as the ones of Struts and WebWork framework, they all make the navigation decision by judging the action return code.

In the View tier

JSF uses UI component tag library language and JSF expression language as its primary display technology. Standard compliant JSF implementations must implement a set of proscribed JSP tags to represent the core components [Phil, 2005]. JSF UI Components covers many component forms, which range form simple “outputLabel**”** component which simply displays text to complex data collection component “dataTable**”** which represent a tabular data from the datastore.

Standard JavaServer Face Reference Implementation includes two libraries of components such as the "HTML" component library which largely mirrors the standard HTML input elements along with a "Core" library which aids in common application development tasks such as internationalization, and validating/converting input data [Chris, 2005]. Besides the basic implementation, JSF component architecture also enables third-part UI component implementation to provide additional functionality above, the typical examples would be the “ADF Faces” from oracle company and the “Myfaces” implementation from Apache organization.

3.4.1 JSF key components

In this section we would briefly go through components presented in Figure 4.9, however, because most of the components contribute directly to the web features, so we would discuss them more in the next chapter.

UI components:

The UI components are stateful JavaBean classes maintained in the server side, they interact with clients in the form of properties, methods and events and they usually integrated to a component tree to constitute the view page.

Renderer

“Renderer” acts as a translator between the UI component tag and HTML tag, it is responsible for rendering the component and obtaining the component value from the user input.

Validator

The “validator” component is used for validating user input data, it normally binds with the UI component and could be shared by multiple components.

Back beans

The “Backbeans” is a special managed bean which holds references to UI component.

Converter

The “converter” is used to convert the value of component to or from the String type to display, same as “Validator”, it is also need to be registered to the UI component.

Events and listeners

JSF simulates Java swing’s event/listener mechanism, it utilizes its UI component to

generate events and use managed beans to be the event listener method carrier, the event listener method should register to the UI component as a part of the component property.

Message

The JSF framework utilizes “Message” component as the container of information displaying back the clients, which include various error messages and application message. In the front side, JSF makes use of a special tag to be the displayer of those messages.

3.4.2 JSF workflow

In general, a complete JSF Request Processing lifecycle consists of six main phases: Restore view, Apply Request Values, Process Validations, Update Model Values, Invoke Application and Render Response. Figure 4.10 is a state diagram showing what happens when JSF processes an incoming request from a client.

`

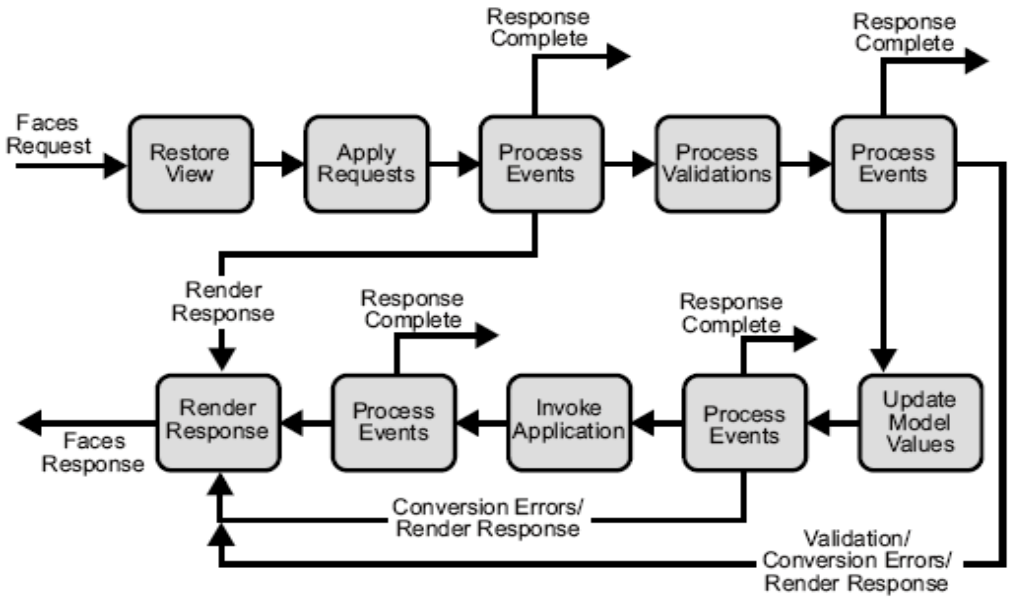


Figure 3.10 JSF standard request-response life cycle [Eric et al, 2006]

Restore view

When a client makes a request to the JSF page, such as a page link or a form submit, the JSF framework begins the “Restore view” phase.

During this phase JSF framework builds the view for the page and wire various events handler around the components, if the request is a initial request for the page

or there are no parameters nested inside the request, the JSF framework will just build the empty view and jump to “Render Response” phase directly.

Apply Requests

During this phase, corresponding components in the page first try to obtain the request parameters to update their properties, after that JSF framework checks whether the events registered in the components has been invoked, if it is, the invoked events will be put into the event line for the later execution in the “Invoke Application” phase. However, if the event-binding component has the “true” value for its “immediate” property, the JSF framework will invoke the event and execute the action code immediately and then ignore the subsequence phases and jump to the final “Render Response” phase directly.

Process Validations

During this phase JSF framework asks each component to validate itself (with the registered “Validator” of the component), if something wrong has been found, it will jump to the final “Render Response” phase and render the original request page.

Update Model Values

After determining the data is valid, JSF framework starts to update all the values of managed beans and the model objects associated to the component, if the local data can not convert to the type specified by the bean or model property, the life cycle advance to the “Render Response” phase and render the original request page with conversion error message.

Invoke Application

During this phase, the JSF implementation addresses the application events which are previously stored in the event line, those events will be broadcasted to different event listeners which subsequently invoke the business method of the domain model

Render Response

During this phase, the selected view will be rendered using the application displaying technique (normally JSP technology), if the request is the initial request, the page components will be first loaded to the empty view, otherwise the page will be sent directly to the clients.

In addition to main phases, JSF framework also reserves the space for a “Phase event” to be invoked before and after each phase. Generally the “Phase event” is

generated by JSF itself rather than by UI components, and requires developers to implement a Java interface to register event listeners. They're normally used internally by the JSF implementation, but sometimes developers could also use them to initiate managed beans' values or set test environment for their application. [Mann, 2005]

3.5 Summary

The handling process of Struts framework is comparatively easy and can be rapidly comprehended by rookies. However, the biggest shortage of Struts framework is that it does not supplies the high level abstraction well for developers, the Struts application may be full with Java Servlet API which make the system code harder to maintain and test.

WebWork framework has a similar workflow and structure with the Struts1.X framework, they both use a centralized controller to pre-handle the user's requests and they both utilize the "Command" pattern, the "Action" class, to be the business logic adapter. However, in contrast to Struts1.X, WebWork provides more flexible "Action" class which is decoupled with Servlet API and high-level web feature supports for web development. Furthermore, in addition to "JSP" technology, WebWork view tier also supports many other displaying techniques such as "Velocity", "Freemark" and "XSLT", which bring more choices and further adaptability for the application. Because of the similarity but additional advantage over Struts1.X, Struts2 have designed from the ground up with the WebWork design concept in mind, as a matter of fact, there is not much difference between these two frameworks except that they use different API name and Struts2 delete a spot of functions of WebWork.

Tapestry is large framework and provides a wealth of predefined component for handling details of object pooling, session management, and HTML components [Ford, 2004]. The nature of the framework makes developers be able to focus on coding in terms of objects, properties, and methods rather than with high awareness of URLs or query parameters, desktop UI experienced developers could easily adapt themselves for Tapestry development since the framework handles all the low-level web details of the application. However, the poor official documentations and complicated internal structure make Tapestry difficult to get started, developers always need to comprehend a great deal of details before they can safely utilize the functions supported by Tapestry, which is often not the case in other Java web framework. But once developers get familiar with Tapestry development process, they

could get pay off by tremendous improvement of development efficiency.

JSF is another component-based framework which packages up chunks of component functionality and reuses them in different contexts. From the developers` point of view, there are two major differences between JSF and Tapestry: first JSF utilize various managed beans to provide properties and event methods support for JSF component rather than a single page object in Tapestry; second it follows a fixed web life cycle to deal with web requests. These differences bring simplicity for JSF development since they steer clear of constraints on using page objects (e g. understanding page pool principle or compulsory security check) and avoid understanding multiple URL handling mechanism, yet at the price of system efficiency.

4 Methodology

To reap the full benefit of using a framework, it is necessary to understand the typical characteristics of the framework and evaluate it in the context of the application. If the framework makes the job easier without forcing us to compromise, it is a good choice. If we constantly have to code around the framework and perceive that problems caused by it are more than the benefit provided by it, we should discard it [Ford, 2004]. Based on this principle I organized several steps to conduct the framework comparison, they are listed in the following sections in this chapter.

4.1 Feature comparison

The efficiency of Java-based web development can be increased by the use of an appropriate framework, however, choosing a proper framework is dependent on several factors, this part of the thesis chooses six basic but important web features to be the yardstick of the framework, it discussed each web feature from framework to framework and gives a comprehensive presentation of each framework's web feature implementation. The six web features are listed below:

Navigation rules

The "Navigation rules" refer to the mechanism of how the framework dispatcher the view page for the web clients, it corresponds to the "Dispatcher" role of the "Front Controller" pattern.

Validation

The validation mechanism of each framework would be discussed and I evaluated them by checking whether it is easy to use and whether the framework support client-side (JavaScript) validation.

Internationalization

The I18n support and corresponding displaying technique for internationalization of different frameworks would be discussed and compared in this part.

Type conversion

Type conversion is very convenient for situations where you need to turn a "String" into a more complex object. It sets the web programmers free from the converting the raw String type by their own. In this part I would discuss and compare different frameworks` type conversion mechanism.

IoC support

Inversion of Control (IoC) is the design pattern that used to build test-oriented application, it have been popularized for year and utilized by much famous software, a representative example would be the “spring” framework that makes a huge utilization on it. In this part the IoC concept would be first discussed and then how different frameworks implement IoC feature and the easiness of using them would be presented and discussed.

Post and Redirect

This feature refers to how the framework handles the web form duplicate post problems.

Among numerous web features that can be discussed for Java web frameworks, I concluded and summarized the features mentioned above to be comparison yardstick based on my own experience and judgment. The reason for me to choose these six features is that unlike some of the fancy features that can be supported by a particular framework, these features are general and essential, and their usage scope almost covers every types of Java web application from small scale to large scale.

4.2 Case study and Conclusions of Java web frameworks

A simple “Project Track” web application was presented, this web application was originally from the book *JSF in Action* [Mann, 2005] and I revise and expend the application to make it as a practical example for this thesis. This web application has been implements with Struts1.X, webwork2.2X, Tapestry4, and JSF1.2. After each web feature’s theoretical analysis, the four versions’ corresponding code snippet of the application was provided to give a comparison in a practical way.

Because of the unimportance and less relevance, I would not present requirements and function descriptions of the “Project Track” web application, in this thesis I put more concentration on the web feature’s implementation details.

At the end, I divide the thesis comparison conclusion into two parts. In the first part the web feature implementations of four chosen frameworks are evaluated and the advantage and disadvantage of them are also concluded. The second part sums up the suitable web application types that different framework can best fit in according to the framework infrastructure and feature implementation.

4.3 Delimitations of the Method

There are several items related to this work maybe has the influence of the research result:

- Some framework features such as “Testability” and “Ajax support” would not be discussed in this thesis because of the author’s knowledge limitation.

- The sample web application’s feature discussion will be restricted by the implementation skill of the thesis author.

- The web features` shortcoming is limited by the version of the frameworks, the specific web feature could be reinforced in the later version of the framework.

However, confident to say, all of the three can not have severe impact on the correctness of research result. For the first two delimitation, as state before this thesis concentrate on six basic but essential web features, and presented code snippets of sample application focus on reflecting the real feature implementation of different frameworks and thus no fabulous programming skill is need at this point. For the third delimitation, the readers could get the newest information from the framework official web site, and easily adjust their options based on the research result of this thesis.

5 Web feature comparison

Tremendous software development achievements can be accomplished by making a great deal of minor but intelligent decisions [Lightbody and Carreira, 2005], and thus a good framework should assist developers by supplying features that restrict development from chaos but is also careful to give as many good options as possible. In this chapter we compare six important web features for four chosen framework by investigating feature implementation and presenting corresponding case study code, and then at the chapter six we make a conclusion and evaluation for each framework's web features.

5.1 Navigation rules

The navigation mechanism of Java web framework corresponds to "dispatcher" component of Front Controller pattern, it prescribes the general rules of locating response pages to users and thus marks out the overall application navigation route for the visited web request. This section describes how different frameworks make their own way to implement the page navigation rules.

5.1.1 Struts1.X

The operation-based nature and the internal implementation of the command design pattern drive Struts1.X framework utilize an "Action-oriented" navigation rule to control the route of visiting web requests. The "Action-oriented" means that the page dispatching mechanism which includes page redirecting, page forwarding and page including activities is determined by the invoked "Action" and the result codes this "Action" class produces, it does not concern with which page that generates the action invocation URL. This "Action-oriented" feature can be also reflected by the format of action mapping configuration. In the Struts configuration file, most of the dispatching information (except the global dispatching) is nested inside the "<action>" XML tag to indicate their affiliated relationship to the specific "Action" class.

Direct page navigation which means that going through different pages without invoking action classes is not recommended by Struts framework, so Struts application developers always need to build action classes to follow the Struts navigation structure even in the case that the action classes do nothing but simply offer result codes. Fortunately Struts1.X framework provides the build-in "Action" class such as "ForwardAction" to fill in the need of creating "empty" action class.

Developers could simply achieve the same direct page navigation effect by designating the “ForwardAction” action class and response page name is the same configuration unit.

Case Study:
Listing 5.1 lists parts of the “Project Track” configuration file to display the navigation rule used in this Struts application.

Listing 5.1 Navigation rules for Struts version of “Project Track”

```
<action
  path="/login"
  type="struts.projecttrack.actions.LoginAction"
  . . . . >
 <forward name="inbox" path="/protected/inbox.jsp"/>
 <forward name="show_all" path="/general/show_all.jsp"/>
</action>
 . . . .
<action
  path="/headerToInbox"
  parameter="/protected/inbox.jsp"
  type="org.apache.struts.actions.ForwardAction"
  . . . .>
</action>
```

The content nested in the first “action” tag pair is the most common configuration used in the Struts application where developers designate forward information inside the action tag. As we can see in the bold part, the Struts framework will dispatch either “inbox.jsp” or “show_all.jsp” page to the clients according to the different result code string returned by the customized “LoginAction” action. The second “action” tag example indicates the case when no business logic needs to perform during the page dispatching, all developers need to do is to configure the build-in class "org.apache.struts.actions.ForwardAction" as the type attribute and page name as the parameter attribute, Struts will automatically “Forward” to the page indicated in the parameter field without developers creating a new “Action” class or writing the forwarding information.

5.1.1 WebWork2.2X

Same as Struts1.X, WebWork2.2X framework also utilizes an Action-oriented navigation mechanism to conduct the action mapping, components correlated to the

“Action” class such as “Interceptors” and “Result” are all configured inside the “action” tag to indicate that they only serve the specific “Action”. However, in addition to configure the individual “Action” element, WebWork also introduce the “Package” and “namespace” concept to organize and group the action configurations, which can logically divide various “Action” into different function categories. The “Package” is a basic but necessary unit for the “xwork.xml” configuration file, it supply spaces for the action configuration definition and could define various global “Result” information for the nested action configuration to use. The “namespace” acts as an attribute of “Package”, it is mainly used to provide a virtual URL hierarchy for all the action mappings defined in the “Package”, for instance, if we have a request URL like:

http://localhost:8080/ProjectTrack_WebWork/protected/login.action

The WebWork will try to search an “Action” named “login” in the package with the “protected” namespace attribute.

Case Study:
Listing 5.2 presents the example snippet of the WebWork version **“**Project Track**”.**

Listing 5.2 Navigation rules for WebWork version of “Project Track”

```
<package name="default" extends="webwork-default"
namespace="/protected">
    <default-interceptor-ref name="login"/>
      <global-results>
        <result name="loginpage">/login.jsp</result>
      </global-results>
      <action name="login"
       class="webwork.projecttrack.actions.Login">
          <interceptor-ref name="validationWorkflowStack"/>
          <result name="success" type="chain">inbox</result>
          <result name="input">/login.jsp</result>
      </action>
      . . . . . .
</package>
<package name="general" extends="webwork-default"
namespace="/general">
. . . . . .
</package>
```

There are two “Packages” defined in the listing 5.2, each of which has their specific name and namespace. When executed at runtime, the WebWork will parse the “namespace” from the request URL and navigate sequentially each qualified “Package” (with the right “namespace” parameter) to find the requested action. This procedure requires developers to be careful when they configure the action mapping name, since WebWork2.2X forbids duplicated names in the same “namespace” even the action configurations are located in the different “Package”.

Listing5.2 also shows other feature support example of the “Package”, as we can see in the first package element: “default-interceptor-ref” tag defines the default executed action in the “protected” namespace, this default action will be invoked when no corresponding action mapping found in the namespace. “global-results” tag defines global “Result” information which could be shared by proprietary action configurations defined inside the package. The “Package” could also includes “Interceptors” and “Result-types” tags which are used to define customized interceptor or interceptor collection and customized result types for the package actions to use. With all the functions and features supported by “Package” and “namespace”, developers could take a more elaborate control over the action configuration. They can also have the choice of using the single “Package” and default “namespace” (without specify the “namespace” parameter in the package tag) throughout the application, but this was not recommended when dealing with large-scaled web applications, where simple maintenance and action reusability could easily be achieved by using “Package” and “namespace” to perform the module design.

5.1.3 Tapestry4

In contrast to Struts and WebWork framework, which control the navigation with the XML configuration, Tapestry determines the rules by dint of its “ILink” component such as “PageLink”, “GenericLink”, “ExternelLink” and its listener methods.

The “ILink” component listed above is used to render a “<a>” hyperlink within the page, when clients click the link generated by the components, the “PageLink” will invoke page service and render the corresponding page, the “ExternelLink” will invoke the external service which is quite similar to page service but with extra parameters, and “GenericLink” will normally lead users to pages out of application scope. However, in most of cases, listener methods are frequently used for navigation so that developers control dynamically control the rules, this process usually can be achieved by using the “Directlink” component which invokes direct

service when user click the link or designate listener method for the form submission. There are many fashions for listener methods to direct the navigation, but no matter what fashion it is used, they should always supply the response page information in the form of code. The enumeration of these fashions is listed in the case study part below.

Case Study:

Listing 5.3 Different navigation fashions used in the listener methods.

```
(1) IRequestCycle.activate(“pagename”);
(2) Return “pagename”;
(3) Return “IPage” object;
(4) Return “ILink” object;
(5) Throw PageDirectException(“pagename”);
(6) ICallback.performCallback(IRequestCycle);
```

(1) IRequestCycle.activate() is the most frequently used method for navigation, By designating the page name as parameter of this method, The Tapestry will find the corresponding page object and render that page.

(2) The developers could simply return the page name in the form of string to achieve the same effect as the IRequestCycle.activate() method. However, by using “return”, the listener method should designate “String” as its return type.

(3) Similar to the second one, but instead of return the page name, the developer should construct a “IPage” object and return it from the listener method.

(4) Similar to the second one, but instead of return the page name, the developer should construct a “ILink” object and return it from the listener method. Notice that when return the “ILink” object, Tapestry will return the page to the web clients in the fashion of “Direct”

(5) When using this sentence, Tapestry will interrupt and stop rendering the current page, and “Redirect” to new page designated as parameters.

(6) Sometimes when forwarding to a new page, we want to associate parameters with the page so that it can display dynamic content according to the parameters when rendering. To do this, we need to first let page objects implements “IExternal”

interface, which will force page objects to execute activateExternalPage() method, from which developers could setup the parameters after page activation. After that developers also need to construct a “ICallback” instance using the “IExternal” page object and required request parameters as constructer method parameters so that they could invoke performCallback() method of “ICallback” instance to active pages with parameters.

5.1.4 JSF1.2

Although the action-based navigation rule used in Struts or WebWork framework is expressive and comprehensible, it can not cover all the circumstances for web page navigation. For example if we have a template web page which will be included by several web pages and an associated web form in the template page used to change the locale of the application, the action-based navigation will be awkward when user clicks the locale submit button, and return to the current page with new locale since the input pages of the corresponding action configuration are uncertain, they could be page A, or page B or any page that includes this template page. To deal with this problem, JSF employs a page-based navigation mechanism to direct the request routes. Each navigation unit of JSF configuration file consists of three parts: a unitary input page, multiple action result codes and multiple corresponding output pages. Unitary input page is the essential part of navigation units, it is also the start point JSF searches up to orient navigation unit. Action result codes and output pages must form pairs and each pair construct the action mapping for input page, case study will show a detailed example for this.

In order to avoid the situation when one page have two separate actions with the same result string or two action method expressions that return the same result string. JSF also introduces the action methods (designated by the optional “<from-action>” XML element) in the navigation unit to refine the resource of result string code. The action mapping is only valid when the result come from the method specifies by “<from-action>” value, sometimes, for simplicity, developers could just use action methods expression as the input resources.

Case Study:

Listing 5.4 Navigation configuration file example for JSF framework

```
<navigation-rule>
    <from-view-id>*</from-view-id>
```

```
    <navigation-case>
        <from-outcome>inbox</from-outcome>
        <to-view-id>/protected/inbox.jsp</to-view-id>
    </navigation-case>

    <navigation-case>
        <from-action>#{createProjectBean.create}</from-action>
        <from-outcome>create</from-outcome>
        <to-view-id>/protected/edit/create.jsp</to-view-id>
    </navigation-case>
    . . . . . .
</navigation-rule>
```

The first navigation case exemplifies the normal situation used in JSF, the second one shows the example of using “from-action” element to refine the result code, it indicates that the “/protected/edit/create.jsp” action mapping is only valid when the “create” result code generated by create() method of “createProjectBean” managed bean. Also notice that in the “from-view-id” field, we use “*” to specify the input page which expose another feature of JSF navigation: wildcard. The wildcard is effective only in from-view-id field and it is often used to configure the global action mappings.

5.2 Validation mechanism

Validating form data is essential to preventing incorrect data from getting into the system. Having a web application that disgorges system error messages isn’t good for anybody’s professional image, so it is worthwhile to keep users from inputting invalid data in the first place. That’s precisely why validation mechanism—the ability to block bad input—is so important, and that is why different Java web frameworks have extensive support for it. This section will show different validation techniques implemented by four chosen framework and the primary technique of each framework will be exemplified by “Project Track” case study.

5.2.1 Struts1.X

The Struts1.X framework support several types of method to validate the submitted form data. The most direct method for validating values is coding the validations directly in the “Action” execute() method. However, this approach mixes validations and business logic, and thus it is not recommended by Struts framework. To separate the validations code, Struts framework utilize ActionForm`s validate () method to

carry out the validation action, it is called before Action's execute() method, and it could force the framework to stop when the form data break the validation rules (see chapter 4.1.2).

For more advanced validation requirements and validation reuse, Struts1.x has introduced David Winterfeldt`s "Validator" framework to bring a meta-data driven and XML-based validation system to the framework. From the developer's point of view, the "Validator" framework consists of two parts: the first part is the "validation-rules.xml" file which contains various validation logics defined by Struts framework. The second part is the customized "validation.xml" file which needs developers to designate the detailed form name and various field validation logics of the form. With the help of XML editing fashion and predefined validation rule supplied by "Validator" framework, developers could easily establish a stable, efficient and easy-to-maintain validation system for their application, furthermore, because of the decoupled nature of the "Validator" framework, different Struts application could make use of the same validation configuration when they have similar "ActionForm" Struture.

By defaul, all of the validations enabled by "Validator" framework are executed on the server side, However, the developers could simply add "<html:javascript formName="XXXForm"/>" sentences into the view page to enable the some validation logic in the client side.

Case Study

"Project Track" Struts version application utilizes "Validator" framework to perform the user account and password validation when a manager tries to login to the system.

Listing 5.5 shows the "validation.xml" file in which developers set the validation rules for the login form.

Listing 5.5 "Validator" framework example of Struts version "Project Track"

```
<form-validation>
  <formset>
    <form name="loginForm">
      <field property="login" depends="required,minlength">
       <arg0 key="login" resource="false"/>
       <arg1 name="minlength" key="${var:minlength}"resource="false"/>
```

```
          <var>
             <var-name>minlength</var-name>
             <var-value>5</var-value>
          </var>
        </field>
        <field property="password" depends="required,minlength">
          <arg position="0" key="password" resource="false"/>
          <arg position="1" name="minlength" key="${var:minlength}"
           resource="false"/>
          <var>
             <var-name>minlength</var-name>
             <var-value>2</var-value>
          </var>
        </field>
      </form>
    </formset>
<form-validation>
```

As we can see in the above, the XML content first designates the "loginForm", which is the name of "ActionForm" of the login page as the validated form the application, and then it sets the predefined "required" and "minLength" validation rules for the "login" and "password" form field. Other contents define what kinds of the error message should send to users when their input breaks the validation rules.

5.2.2 WebWork2.2X

The validation function of WebWork2.2X framework can fall into two categories, both of them are supported by WebWork "Interceptor". The "WorkFlow" interceptor enable the code-fashion validation, when executed, it first check whether the "Action" class implements "Validateable" interface, if it is, this interceptor will invoke the action validate() method to carry out the validation logic. Next the interceptor will invoke the action's hasErrors() method if the "Action" implements the ValidationAware interface, if this method returns true, this interceptor stops the chain from continuing and immediately returns "input" result string. This process is quite discouraging at the first glance since many interfaces and methods need to be implemented and overridden. However, WebWork framework supplies a "ActionSupport" class which has already implemented all these interfaces and defined several useful methods to simplified the development process, developers could extend this class rather than implement "Action" interface to facilitate the action validation.

For the validation irrespective to business domain, WebWork recommend developers to use its “validation” interceptor to perform a XML configuration-based validation. This validation process is quite similar to “Validator” framework used by Struts1.X framework, both of them use the external xml metadata files to describe what validations should be performed on the “Action” class and both of them make use of various predefined “validator” to designate the specific validation rules to the fields. However, the WebWork implementation supports more convenient features when deploying the validation XML configuration, these features includes *inheritance validation definition* which enables parent “Action” validation to be executed when carrying out the child “Action” validation, *Validation Short-Curcuiting* which enables short-circuiting a stack of validators if the previous validator falls and *Visitor Field Validator* which enables the customized class’s validation (see case study example.)

WebWork framework also supports the client-side validation, this function could be enabled by setting “true” to the “validate” attribute of the WebWork “form” tag and it only works when developers using the “validation” interceptor to validation the “Action” class. An example is listed below:

<ww:form action="login" ***validate="true"*** *method="post">*

Case Study
Listing 5.6 shows the WebWork’s version of login validation, which is quite similar to configutaion in the Listing 5.5.

Listing 5.6 WebWork XML configuration based validation

Login-validation.xml:

```
<validators>
    <field name="login">
        <field-validator type="requiredstring">
            <message> "Login" field is required.</message>
        </field-validator>
        <field-validator type="stringlength">
            <param name="minLength">4</param>
            <param name="trim">true</param>
            <message> "Login"  field must be at least 4 characters
            </message>
        </field-validator>
    </field>
```

```
    <field name="password">
        <field-validator type="requiredstring">
            <message>"password" field is required.</message>
        </field-validator>
        <field-validator type="stringlength" >
            <param name="minLength">2</param>
           <param name="trim">true</param>
            <message> "password"  field must be at least 2 characters
            </message>
        </field-validator>
    </field>
</validators>
```

Edit-edittoinbox-validation.xml:

```
<validators>
    <field name="project">
        <field-validator type="visitor">
            <param name="appendPrefix">true</param>
            <message>Project:</message>
        </field-validator>
    </field>
</validators>
```

The second file “Edit-edittoinbox-validation.xml” in Listing 5.6 presents an example of Visitor Field Validator, the normal WebWork field “validator” aims at the fields with Java primitive type, Visitor Field Validator extends this validation scope to let the WebWork has the ability to evaluate a customized-class type field, such as the “Project” class type in the above example. To fully enable this function, developers also need to create a particular validation file to specify the validation rules for various fields of customized classes. Listing 5.7 shows the “Project-validation.xml” file content which designates the rules for “Project” class.

Listing 5.7 Visitor Field Validator configuration example

```
<validators>
    <field name="name">
        <field-validator type="requiredstring">
        <message key=”project.namemissing”>project name is
        required!</message>
```

```
        </field-validator>
    </field>
    <field name="initiatedBy">
        <field-validator type="requiredstring">
        <message>project initiator is required!</message>
        </field-validator>
    </field>
    <field name="requirementsContact">
        <field-validator type="requiredstring">
        <message>Requirement Contact person is required!</message>
        </field-validator>
    </field>
    <field name="requirementsContactEmail">
        <field-validator type="requiredstring">
        <message>Contact email is required</message>
        </field-validator>
    </field>
</validators>
```

5.2.3 Tapestry4

The Tapestry validation system is closely bond to its form components. Unlike Struts and WebWork, which need an extra validation framework and thus various validation configuration files to enable the validation rules, Tapestry4 makes use of its build-in “validators” which are normally bond as an attribute of the form-input components to check the input value submitted by users. The “validator” parameter provides a list of validator objects, each of which provides special validation rules to the input component, currently there are ten predefined validators supplied by Tapestry4 framework which offer basic validation rules such as “email”, “date”, number scope and length of the string. To realize more complicated validation, developers could also define their own customized validator objects which must implement the “Validator” interface and associate them with the required components, the Tapestry framework will treat the new validators and execute their code just like the build-in ones.

Displaying error messages is a comparatively complex process in Tapestry framework. The main displaying task is managed by the component “Delegator” and its corresponding object “ValidationDelegate”, they supplied different choices and schemes for displaying the error messages. However, the configuration process is not as easy as it looks like, in order to display the message well, developers must have a good understanding of the “ValidationDelegate” class fields and methods

since it supplies different configuration options for the “Delegator” component, in addition to that, developers also need to write extra controlling code in the case of displaying multiple errors because the default behavior of the “Delegator” component is to display the single message and there is no convenient support for that in Tapestry framework. We will show an example of this in the case study part.

The client-side validation can also be easily enabled by Tapestry framework, to use the client-side API all developers need to do is to set the “clientValidationEnabled” parameter to true on the form components, and Tapestry framework will automatically setup the same build-in validator logic in the form of JavaScript.

Case Study

Listing 5.8 shows the Tapestry version of login validation which uses the build-in validators to enable the component-based validation.

Listing 5.8 Tapestry Build-in validators example of “Project Track”

Login.Html:

```
<form jwcid="form">
       <table cellpadding="0" cellspacing="0">
           <tr>
               <td>
                   <span jwcid="errors">
                       <span jwcid="isInError">
                           <li><span jwcid="error"/></li>
                       </span>
                   </span>
               </td>
           </tr>
           <tr>
               <td><span jwcid="logintext"/> </td>
               <td><span jwcid="passwordtext"/></td>
           <tr>
       </table>
</form>
```

Login.page

```
<component id="logintext" type="TextField">
        <binding name="validators" value="validators:required[The {0}
```

```
is missing!],minLength=3[The account should have more than 5
characterw]"/>
          . . . . . .
</component>

<component id="passwordtext" type="TextField">
       <binding name="validators" value="validators:required[The {0} is
missing!],minLength=2[The password should have more than 2 characters]"/>
        . . . . . .
</component>

<component id="errors" type="For">
       <binding name="source" value="beans.delegate.fieldTracking"/>
       <binding name="value" value="currentFieldTracking"/>
    </component>

    <component id="error" type="Delegator">
       <binding name="delegate" value="currentFieldTracking.errorRenderer"/>
    </component>

    <component id="isInError" type="If">
       <binding name="condition" value="currentFieldTracking.inError"/>
    </component>
```

Listing 5.8 presents two parts of our login view page, the first “Login.Html” file is the template file which contains several standard HTML tags and tapestry components featured by “jwcid”, and the second part is the page configuration file used to configure different components indicated in the template file. As we can see in the “Login.page” file, Tapestry framework uses various “validators” to bind with the “login” and “password” field to execute the same validation logic of Listing 5.5 (or 5.6), which saves great time and effort of validation configuration. “Login.page” also shows an example of how we can use "Delegator" component to display multiple error messages. In order to display all the error messages in the login page, we first utilize a “For” component which will hold a loop to encapsulate the "Delegator" component, and then in order to avoid showing any empty place such as:

** error message.*
***
** error message.*

we also use a “If" component to make judgment about whether any error happened for the field, if there is no error occurs, the framework will only draw the error messages without showing any “blank” between them.

5.2.4 JSF1.2

Same as Tapestry, JSF performs the validation tasks by associating various build-in or customized “validators” with its input components. Nevertheless, the “validator” in JSF is much more generalized than the one used in Tapestry since it could be applied to any standard JSF input component using the tag “<f:validateXXX =…>” nested into the component tags. Up to now, they are three build-in validators used in JSF which focus on length of string, length of Long number, length of Double number validation, more advance validation logic could be generated in a customized “Validators” which implements the “Validator” build-in interface. Validation could also be delegated to a managed bean method binding in the “validator” attribute of an input component. This mechanism is particularly useful for application-specific validation, where the submitted input values need to take account to decide if the validation is successful.

By default, JSF does not have explicit support for client-side validation. In other words, any validator methods in managed beans, as well as the various “validators” will not generate JavaScript code to check a component’s value on the client side. Although developers could write their own Javacript validation code in the view page or make use of third-part software such as apache “shale” to remedy this defect, the process run the risk of spending more effort on programming or learning new technologies.

Case Study

Listing 5.9 Build-in “validator” and validation methods example of JSF

View page:

```
<h:inputText id="username" required="true"
        value="#{authenticationBean.login}">
    <f:validateLength minimum="5" maximum="20" />
</h:inputText>
<h:message for="username" />

<h:inputSecret id="password" required="true"
```

```
        value="#{authenticationBean.password}"
        validator="#{authenticationBean.validate}">
</h:inputSecret>
<h:message for="password" />
```

AuthenticationBean validate() method:

```
public void validate(FacesContext context,
            UIComponent component,
            Object obj) throws ValidatorException
     {
           String password = (String) obj;
           if(password.length() < 5) {
            FacesMessage message = new FacesMessage(
                FacesMessage.SEVERITY_ERROR,
                "Te password length should not be less than 5",
                "The password length should not be less than 5");
            throw new ValidatorException(message);
           }
    }
```

This example presents the usage of build-in “validator” and validation methods. For the “username” field, we utilize “<f:validateLength>” to limit the length scope of the input values, any input breaks the rules will be captured by “<h:message>” tag. On the other side, the “password” field make use of managed bean method to perform the validation logic, the method signature should strictly conform to the one showed in this example and the error message should be wrapped as a “FacesMessage” instance and throwed in the form of “ValidatorException” so that “<h:message>” tag could capture. In the case of developers want to display the messages together rather than separately, JSF also supplies “<h:messages>” tag to present all messages stored in request scope in centralized layout.

5.3 Internationalization

Enabling an application to support multiple locales is called internationalization [Mann, 2005]. It is a challenging process to develop an internationalized application since many customized work for a particular locale has to be done such as translating text string, selecting a different graphic and sometimes changing the view page layout. Fortunately, Java web frameworks provide a comprehensive toolbox for building internationalized applications and managing our localized texts. This

section dissects the internationalization supporting feature into “Internal processing” and “External processing” two aspects and discusses respectively how different frameworks fulfill the functionalities of these two aspects. In the “Internal processing” part we expose different frameworks’ management mechanism of the internationalization resource bundle files and their presentation techniques for internationalized information. In the “External processing” part we concentrate on how different frameworks manage the response page locale and how they adjust the locale according to web clients’ request.

5.3.1 Struts1.X

Internal processing

Struts framework utilizes its XML configuration file “Struts-config.xml” to provide centralized management for various resource bundle files. These files will be read during the application initialization and saved as a “MessageResources” Java object into application web scope so that it can be used all over the web application. Figure 4.1 shows the relationship among XML configuration, “MessageResources” object and resource bundle files in Struts1.X framework.

In addition to reading the localized text using background Java API, Struts framework also supports for using its customized JSP tag to directly display internalization information in the front side, this includes showing-text tag, showing-picture tag and button tag. Developers just need to designate the localized key string as the special tag’s property, the Struts framework will automatically find the corresponding localized information and nest them inside the view page. This mechanism also works for Struts “Validator” framework, all of the error or exception messages could be localized if the developers configure the localized key string into the specific place.

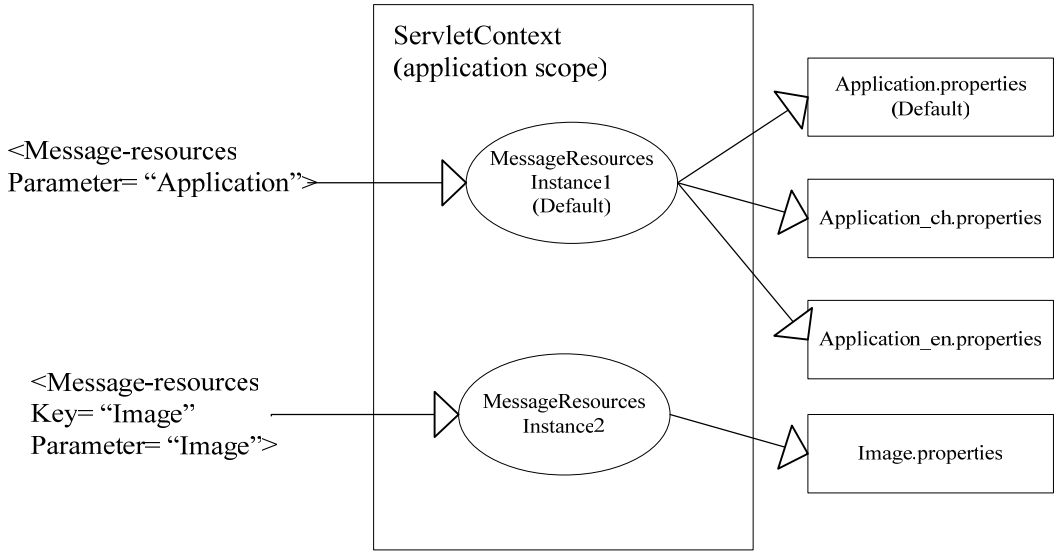


Figure 5.1 Relationship among configuration, “MessageResources” and resource bundle files.

External processing

The i18n external processing of Struts1.X framework is mainly performed inside the processLocale() method of “RequestProcessor” class which is invoked before any action methods. In the default situation, when a web client sends a request to web server, the Struts framework will parse the request locale from the parameters and save it into the client`s “HttpSession” scope, however, if the client’s “HttpSession” scope already has the locale field, the framework will ignore the request locale and go on its way. The locale information stored in the “HttpSession” is mainly used for responsing view pages, all of the view pages sent back to the clients will have the same locale as the one in the “HttpSession” scope.

Case Study

One of the functionality of “Project Track” is to change the locale of the application according to the language choice chosen by the users.

Listing 5.10 is a part of a view page example that the Struts1.X version application makes use of its specials tags to dynamically change the language locale:

Listing 5.10

```
<table cellpadding="4" cellspacing="0" border="0">
    <tr>
        . . . . .
         <td>
```

```
            <html:link page="/headerToCreate.do">
               <html:img  pageKey="/images/create.gif"
               altkey="page.create"/>
               <bean:message key="CreateNewToolbarButton"/>
            </html:link>
        </td>
         <html:submit property="submit">`
              <bean:message key="Header.submit"/>
         </html:submit>
</table>
```

As we can see in the list, the Struts customized JSP tags use the “key” or “altkey” property to designate localized key string, when rendering the page, the Struts framework will find the corresponding text information in an appropriate resource bundle file (according to the locale in the “HttpSession” scope) and fill them into the appointed place.

Listing 5.11 presents the action code that actually changing the locale of the application.

Listing 5.11 Dynamical changing the Locale in Struts framework

```
public class HeaderAction extends BaseAction
{
    public ActionForward execute(ActionMapping arg0,
    ActionForm arg1, HttpServletRequest arg2,
    HttpServletResponse arg3) throws Exception
    {
         HeaderForm header=(HeaderForm)arg1;
         Locale locale=new Locale(header.getLanguage());
         HttpSession session = arg2.getSession(false);
         session.setAttribute(Globals.LOCALE_KEY, locale);
         System.out.println(arg2.getServletPath());
         return  arg0.getInputForward();
    }
}
```

According to the operation mechanism of the Struts framework, “HttpSession” is the key point that supplies the locale information to all responding page, so in the application, all we need to do is create the “Locale” object in the light of client`s language choice and save it into the “HttpSession” field. The bold part of the Listing 5.11 presents this process.

5.3.2 WebWork2.2X

Internal processing

Unlike Struts1.X framework`s centralized management of resource bundle, WebWork splits the resource bundles per action, per package and per interface. The order WebWork framework searches the resource bundle file is listed below [Opensymphony WebWork Wiki, 2006]:

1. ActionClass.properties
2. BaseClass.properties (all the way to Object.properties)
3. Interface.properties (every interface and sub-interface)
4. ModelDriven's model (if implements ModelDriven), for the model object repeat from 1
5. package.properties (of the directory where class is located and every parent directory all the way to the root directory)
6. search up the i18n message key hierarchy itself
7. global resource properties (webwork.custom.i18n.resources) defined in webwork.properties

This policy maybe end up with duplicated messages in different resource bundles, however, it can be fixed by creating a “ActionSupport.properties” (Normally “ActionSupport” class acts as parent class for other “Action”) in the application class path and put all internationalized messages in it if all of our “Action” classes extend the “ActionSupport” class.

WebWork also offers various convenient customized tag to receive the internationalized message, such as “text” tag and “i18n” tag. By designating the requested key string of resource bundles, these tags will display the corresponding internationalized message when rendering the page. This rule also works for “message” element of WebWork validation framework, Listing 5.7 also contains a example for this.

External processing

WebWork utilizes its “I18n” Interceptor to control the locale of response pages. The “I18n” Interceptor reserves a space for storing locale information in the HttpSession web scope and makes use of it as a locale yardstick for every response page. Every time when an action request has been sent to the server, the “I18n” Interceptor (if it is configured with that action) will check whether there is any locale information associated with “request_locale” request parameter, if there is, the “I18n” Interceptor will make the locale specified by “request_locale” parameter as the new yardstick

locale and save it in the HttpSession web scope, otherwise it just hands over the request and continues to use the old yardstick locale.

If developers want, they can change the “request_locale” name by assigning a new name as the value of “I18n” Interceptor “parameterName” parameter.

Case Study
Listing 5.12 presents the WebWork version of dynamical changing the locale of web applications.

Listing 5.12 Dynamical changing the Locale in WebWork framework

View page:

```
<ww:select theme="simple"
           name="request_locale"
           list="#session.visit.localeList" />
```

Xwork.xml:

```
<action name="header_inbox"
class="webwork.projecttrack.actions.Header">
     <interceptor-ref name="i18nStack"/>
       <result name="success" type="chain">inbox</result>
     </action>
```

Listing 5.12 shows two parts that directly contribute to the locale changing, the first part refers to the content of “Project Track” view page, as we can see in the above, we use "#session.visit.localeList" to supply various locale string to the “select” tag and within the “Project Track” context these string involves “en” for English, “zh” for Chinese and “ru” for Russian, which provides three options that “i18n” interceptor could retrieve from the “request_locale” request parameters. The second part shows the “i18n” configuration for our “Action” class, here we use the predefined i18n interceptor stack which contains the “i18n” interceptor and several other fundamental interceptors to package our “head_inbox” action.

5.3.3 Tapestry4

Internal processing

Comparing to over-centralized management of Struts framework and over-dispersive management of WebWork framework, Tapestry utilize a moderate way to maintain its

resource bundle files. There are three places could subsume the localized information used in the page, the preferential place is the page or component resource bundle files which are located under the “WEB-INF” directory, and they have the same name as the page template file but with a different “properties” extension. The second place Tapestry searches for is the “namespace” resource bundle files, which share the same name of the web application and with the “properties” extension. The last place can be called as “global property source” resource bundle file which can be designated in the web application deployment descriptor (web.xml), JVM property and system property list.

The presentation support of internationalization is also straightforward in Tapestry4, developers could simply use the “” sentence in the template file to retrieve the localized message, in the case of requiring localized parameters, they could also utilize the “OGNL” and internationalization method defined in the Tapetry to perform the task, a representative example is listed below:

```
<span
jwcid="@Insert"
value="ognl:messages.format('keyStringName',param1,param2,...)"/>
```

In addition to the localized message, Tapestry utilizes the Java resource-bundle style to manage other resources in the framework as well. Take the page template file for example, if various locale-styled temple files exist in the same application, such as “Home.html”, “Home_zh.html” and “Home_en.html”, Tapestry framework will intelligently choose the right template in the light of request locale. This rule could also apply to the image file and component specification file in the framework.

External processing

As we stated before, the “Engine” component is the core processor of Tapestry framework, it receives and parses the request URL and chooses the suitable service to invoke. However, in order to cooperate with clients from different countries, Tapestry framework creates multiply engines with different locales to handle the user’s request, each engine chosen by the framework has the same locale property as the requests it handles, and each page loaded by this engine will be initialized with the engine locale. As a result, to change the locale of the response pages, we must change the locale property of the engine which load the page, this can be done by using the setLocale() method of the corresponding engine instance. Furthermore When developers call setLocale() on the engine, not only its locale is set, but also a cookie is created in the

browser to store the locale. The next time the browser sends a request to the application, it will include that cookie in the request, which will be the condition for Tapestry framework to choose the engine.

Case Study

Listing 5.13 presents the Tapestry version of dynamical changing the locale of the application.

Listing 5.13 Dynamical changing the Locale in Tapestry framework

```
public void setLanguage(IRequestCycle cycle)
    {
        cycle.getEngine().setLocale(getSelectedLocale());
        cycle.cleanup();
        cycle.activate(cycle.getPage().getPageName());
    }
```

Although the principle of dynamical changing the locale is a little bit complex in Tapestry, the code developers need to write is simple. As you can see in the Listing 5.13, we first use “cycle.getEngine().setLocale()” sentence to change the locale of our engine. However, this sentence alone can not do the deed immediately for the current page, since the current page has already been initialized with the former engine locale, so we use cycle.cleanup() to let the current page return to the pool and reload the page with new locale engine using cycle.activate() method.

5.3.3 JSF1.2

Internal processing

Setting up internationalization bundle files in JSF is fairly straightforward and very similar to the fashion that Struts framework employs, different resource bundle files should be registered as locale entries with alias name in the XML configuration file (normally face-config.xml file) so that JSF framework could initialize them as MessageResource instances at the startup stage and publicize these instances for application to use. The locale entries in the configuration file have the global effective range which means that the internationalization information contained in these entries is valid all over the application. Instead of defining resource bundle files in the form of XML configuration, developers could alternatively define them at the top of each view page using the sentence “<f:loadBundle basename="resource path and name" var=" alias "/>”. However, the “loadBundle” definition fashion

greatly restricts the effective range of resource bundle files, developers could only use the corresponding internationalized message in the current page.

In addition to the resource bundle definition, developers should also add entries in configuration file to indicate which locales the application supports, this behavior helps JSF exclude the unnecessary search of the nonexistent resource bundle file if web clients choose the unsupported locale for the application.

Once the bundle files have been defined, developers could express the internationalization message using the bundle alias and key string within the form of JSF expression language, such as:

<h:outputText value="#{aliasname.keystringname}"/>

For the messages with parameters, JSF also supplies <h:outputformat/> tag with the nested <f:param> tags to pass the parameters, the following is the example usage of these tags:

```
<h:outputFormat value="#{aliasname.keystringname}">
        <f:param value="parameter1"/>
        <f:param value="parameter2"/>
          . . .
</h:outputFormat>
```

External processing

By default, the components in a JSF view are organized into a tree structure with an instance of the “UIViewRoot” class at the root, every time when a page is about to render to the client side, JSF will check the locale attribute of the “UIViewRoot” instance to determine which locale should be used for that page. Based on this working mechanism, developers can set the locale programmatically by calling the setLocale() method of the UIViewRoot object:

```
UIViewRoot viewRoot = FacesContext.getCurrentInstance().getViewRoot();
viewRoot.setLocale(new Locale("zh"));
```

Once developers set the locale attribute of a view page, JSF would carry over the new locale property internally and apply this locale to other view pages in the application, the carrying will not be over until the application change the locale attribute again.

Case study

Considering the simple usage of JSF Internationalization feature, so we would not show any further examples in this part, the key points related to the feature has already been exemplified in the last section.

5.4 Type conversion

The type conversion is a ubiquitous problem that every web application should deal with, the web—or, more specifically, the HTTP protocol—transfers every things as a string or a array of string, no other date type can be specified in HTTP, HTML, or even the Servlet specification [Lightbody and Carreira, 2005]. Although this approach makes the transfer and specification simpler, it places the responsibility of converting input strings to a proper data type on the shoulder of developers. In this section, we'll look at how different frameworks remove all the pain usually associated with this task, allowing them to focus on the business logic and speedy development.

5.4.1 Struts1.X

Struts1.X framework internally integrates the "Commons-BeanUtils" components to perform the type conversion task. The "Commons-BeanUtils" is one of the Apache Commons sub-projects which are used to offer low-level utility classes that assist in getting and setting property values on Java Bean classes [Apache BeanUtils, 2007], it also supplies various predefined converters to convert string request parameters that were included in HttpServletRequest received by a web application into a set of corresponding JavaBean properties. With the help of the "Commons-BeanUtils" components, the newest version of Struts1.X framework could automatically convert the raw string into the java primitive types listed below:

- java.lang.BigDecimal
- java.lang.BigInteger
- boolean and java.lang.Boolean
- byte and java.lang.Byte
- char and java.lang.Character
- java.lang.Class
- double and java.lang.Double
- float and java.lang.Float
- int and java.lang.Integer
- long and java.lang.Long
- short and java.lang.Short
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

“Commons-BeanUtils” component also offers a mechanism that permits developers to define their own “Converter” to convert the string parameters into the customized Java class types. This mechanism has been used by Struts1.X framework to reinforce the functionality of its “ActionForm” classes. To enable this customized type conversion, the Struts developer should follow two steps: First write a class that implements the predefined “Convert” interface and then fill in the detailed conversion logic into the convert() interface method; Second register the converter class into the framework using ConvertUtils.register() method, this step should be done before the “ActionForm” containing the converted properties have been invoked.

Case Study:

In order to the create a new project, the project manager needs to fill in several project information supplied by creating-project page which includes project name, type, introduction and so on. Listing 5.14 and Listing 5.15 show how Struts framework utilizes “Commons-BeanUtils” to the convert project type string parameters into the background “ProjectType” class object.

Listing 5.14 Converter class that used to convert the string to “ProjectType”

```
public class ProjectTypeConverter implements Converter
{
    public Object convert(Class c, Object value)
    {
     Return(ProjectType)(TypeManager.getType(Integer.parseInt((String)
    value)));
    }
}
```

Listing 5.14 shows the customized converting method for the “ProjectType” class, as we can see in the above, the method simply uses the domain business method to construct a corresponding “ProjectType” instance with the request parameter “value”. This new instance will be transferred to the “ActionForm” as the form’s “type” field. Listing 5.15 shows the “CreateForm” class which contains the converted field and “Converter” registration code (In Bold).

Listing 5.15 The “CreateForm” class

```
public class CreateForm extends ValidatorForm
{
    private ProjectType type; // the converted field “type”
    . . . . . .
    public ProjectType getType() {
        return type;
    }
    public void setType(ProjectType type) {
        this.type = type;
    }
    public void reset(ActionMapping arg0, HttpServletRequest arg1)
    {
        ProjectTypeConverter ptc=new ProjectTypeConverter();
        ConvertUtils.register(ptc, ProjectType.class);
        // register to the “Commons-BeanUtils” component
        . . . . . .
    }
}
```

5.4.2 WebWork2.2X

WebWork2.2X framework supports a robust and full functional type-conversion mechanism, it provides more delicate type-conversion functions than the one offered by Struts1.X framework.

Same as Struts1.X framework, WebWork2.2X also supports the automatic conversion (i.e. without additional configuration) between the raw string type and java primitive types. When server and client side interact with each other, the OGNL component integrated in WebWork will automatically find the mismatch between “String” and type of action class field and perform the conversion with its primitive type converters.

The real strength of the WebWork2.2X`s type conversion reside in its handling to the customized JavaBeans classes and collection or array fields. To convert the string type to customized classes, WebWork needs developers to perform two steps: the first step is to create a conversion file to establish the relationship between the converted action class field and the “converter” class. At this stage, developer could choose the global conversion scope by creating “xwork.-conversion.properties” file

in this application classpath or choose the action conversion scope (i.e. the conversion is only limited to the special “Action” class) by creating a “*classname*-conversion.properties” file in the directory same as the “Action” class. The Second step is to build a new “converter” class which must extend the build-in “WebWorkTypeConverter” class and implement related conversion methods. This conversion process is bidirectional, which means that with the two compulsory methods implemented in “converter” class, WebWork is entitled to carry out “String->customized type” or “customized type->String” two types of conversion.

When dealing with conversion to collection or array field, there is not much extra work need the developers to do comparing to the conversion of customized class. If we have an array field in our action class for example: User[] user. WebWork will do the “String -> User” conversion several times with the normal configuration aforementioned. If we use the “List” or “Collection” type to lead the array, we just need to add an extra sentence, for instance “Element-*fieldname=classtype*”, into the conversion file to indicate the class type contained in the collection field. WebWork also supports to convert a string value into an indexed position of a collection. By doing this developers need to add “KeyProperty_*collectionname=fieldname*” sentence into the conversion file. “*collectionname*” indicates the name of collection field in our “Action” or “JavaBean” class, whereas “fieldname” means a “identifier” field of our converted class in the collection field. For example if we add the sentence “KeyProperty_mycollection=id” with the “mycollection” collection field and “int” field “id” of the class in the collection, then we may use the “mycollection(3)” expression in the view page to refer to the third value in the “mycollection”.

Case Study

“Project Track” application has the scenario that when managers submit the project to the next phase, he has the choice to submit various project documents so called “artifacts” to system so that the system can always maintain how many project documents lead by “ArtifactType[]” field have been finished. Listing 5.16 showing our conversion file and “converter” class that convert different document name (when submitted, the documents are transferred to the server in the form of an array of document string name) into the “ArtifactType” class type.

Listing 5.16 WebWork converter class and conversion file example

```
public class ArtifactTypeConverter extends WebWorkTypeConverter
{
    public Object convertFromString(Map context,
    String[] values, Class toClass)
    {
        String ikey=values[0];
        TypeManager manager=new TypeManager();
        return(ArtifactType)manager.getInstance
       (Integer.parseInt(ikey));
    }
    public String convertToString(Map context, Object o)
    {
        ArtifactType type=(ArtifactType)o;
        return  type.iValue;
    }
 }
```

xwork.-conversion.properties:

webwork.projecttrack.domain.ArtifactType=
webwork.projecttrack.actions.ArtifactTypeConverter

We want the conversion to occur throughout the application, so we use “xwork-conversion.properties” file to configure the relationship, when a submitted string value is evaluated to fields with “ArtifactType” type or in reverse order, the WebWork will find “ArtifactTypeConverter” class to the do the conversion work, correspondingly, there are two methods defined in this class to handle double-way conversion. Because the application expects the string array could be converted to “ArtifactType[]” field, so the conversion will occur several times in the light of the submitted name number.

5.4.3 Tapestry4

The type conversion support of Tapestry 4 framework is closely related to form input components, several components are associated with a “translator” attributes which are used to bind with an internal type converter to conduct the conversion between the submitted string type and user required type. For most of the situations, which is the case of Java primitive types, Typestry4 utilizes the OGNL to perform the

automatic conversion. For the more advanced Java type such as “Number” or “Date” Tapestry supplies “Date” and “number” translators with various patterns for developers to customized their needs, the example of using Date” and “number” translator is listed below:

```
<component id="numberField" type="TextField">
  <binding name="translator" value="translator:number"/>
  . . . . . .
</component>
<component id="DateField" type="TextField">
    <binding name="translator" value="translator:Date"/>
  . . . . . .
</component>
```

To create a customized translator, Tapestry framework supplies an abstract “AbstractTranslator” class for developers to extend to build the required translator, developers need to fill in the conversion logic into the formatObject() and parseText() two methods to handle the double way conversion. After the registration configuration, developers could utilize their new translators just like the build-in ones.

Case study:

The biggest shortcoming of Tapestry type conversion is its close binding relationship with the components. The conversion is bounded by the component implementation, if the input component does not possess of the “translator” attribute (i.e. no corresponding translator implementation for the component), there is no way for the submitted string value to be automatically converted to the required type of the corresponding fields. Unfortunately, in Tapestry4 only “TextField”, “TextArea” and “DatePicker” three standard components have the “translator” attribute, developers need to build their own conversion mechanism when using other input components.

In Tapestry version of “Project Track” application, customized type conversions are occurred with the “select” and “PropertySelection” component and all of corresponding conversion code is built from the beginning. Because of the irrelevance and page limit, I will not present the related code here.

5.4.3 JSF1.2

The conversion mechanism used in JSF framework is very similar to the one used in Tapestry. For instance, they all use the automatic converters (invisible to developers)

to convert the Java primitive types, and they all use the explicit converters to convert and configure the format of more complicated types such as “Date” and “Number”. However, instead of using component attribute, which is the case in Tapestry, JSF nests a “<f: converterXXX>” tag inside its form input or output components to designate the conversion rule, the following shows three examples of using date converter, number converter and customized converter which implements the “Converter” interface.

```
<h:inputText value="#{. . .}">
      <f:convertDateTime pattern="MM/yyyy"/>
</h:inputText>

<h:inputText value="#{. . .}">
      <f:convertNumber minFractionDigits="2"/>
</h:inputText>

<h:inputText value="#{. . .}">
      <f:convert    converterId=". . ."/>    // most for the customized converter
</h:inputText>
```

Comparing to the limited number of Tapestry components that support type conversion, almost every JSF output or especially input components could be associated with a converter. With this convenience, developers could design and develop application with ease without worrying too much component converter binding limitation.

Case Study

Listing 5.17 shows the example of JSF customized converter usage. The background information and scenario is same as the one in WebWork case study section.

Listing 5.17 Customized converter example for JSF framework

```
 In the view page:
<h:selectManyCheckbox id="artifactSelect" layout="pageDirection"
                    styleClass="project-input"
                    value="#{visit.currentProject.artifacts}">
   <f:selectItems value="#{selectItems.artifacts}"/>
   <f:converter converterID=”ArtifactType”>
</h:selectManyCheckbox>


In the face-configuration.xml:
<converter>
```

```
    <converter-id>ArtifactType</converter-id>
    <converter-class>jsf.projecttrack.backbeans.ArtifactTypeConverter
    </converter-class>
</converter>
```

Class definition:

```
public class ArtifactTypeConverter  implements Converter
{
    public Object getAsObject(FacesContext arg0, UIComponent arg1,
     String value)
    {
           int id=Integer.parseInt(value);
           TypeManager type=new TypeManager();
           ArtifactType artifact=(ArtifactType)type.getInstance(id);
           return artifact;
    }
    public String getAsString(FacesContext arg0, UIComponent arg1,
    Object object)
    {
           ArtifactType artifact=(ArtifactType)(object);
           return String.valueOf(artifact.getIKey());
    }
}
```

In the view page, we designate the converter with the ID to the "selectManyCheckbox" component which supported by "ArtifactType[]" type , and then we deploy the information in the configuration file to tell JSF which specific converter class relate to this convert ID. At the end, we implement the detail class that override getAsObject() and getAsString() two methods of "Converter interface" to perform the double way conversion. One thing still need to mention that if we change the "<converter-id>ID name</converter-id>" sentence to "<converter-for-class>class path and name </converter-for-class>" in configuration file, JSF will perform the type conversion once it encounters the class type deployed between the "<converter-for-class>" and there is no need to register the "converter" to a detail component.

5.5 IoC support

The essence of language design patterns is to find a harmonized relationship between different classes. In the past few years, several famous design patterns such as "Singleton" pattern and "Factory" pattern have been invented to address the redundancies and coupling problems of the class design, however, they all use a active

way to perform their tasks and still need developers to design the algorithm when deal with the complicated class relationship. In contrast to this, IoC design pattern utilizes a lazy, passive fashion to deal with the class relationship, it delegates a IoC container to help with the class binding details. With a little configuration in the IoC container, developers could blithely use different classes in the application without worrying too much about how they are bond together, since the binding details are all taken care by the background container. In this section, we discuss how different frameworks support IoC features and how they use their IoC implementation to support “Project Track” application.

5.5.1 Struts1.X

Because of the historical reason, Struts1.X framework did not support IoC feature well except binding with the external IoC framework such as “Spring”, However, looking through the Struts1.X working mechanism, we can still capture some IoC feature trace inside the framework. For instance, with specifying the name attribute of the action element (in the configuration file), the framework will automatically wire our “ActionForm” to the corresponding “Action” class without needing developers to manually setting the relationship in the code.

In order to compare the IoC feature with other frameworks, here we still present the code that how Struts1.X version “Project Track” wire the background “project”, “user status” and “user account” information into the application.

Case Study

In order to make the application workable, several predefined user data need to be inserted into the application as the background data-store support. For simplicity, in the application I used the “memory” fashion rather than “database” fashion to implement the data store class. Listing5.18 shows the detail binding code.

Listing 5.18 Data store class binding code in Struts framework

```
public class Initializer implements ServletContextListener
{
    public void contextInitialized(ServletContextEvent event)
    {
        ServletContext context=event.getServletContext();
         context.setAttribute(Constants.PROJECT_COORDINATOR_KEY,
         new MemoryProjectCoordinator());
         context.setAttribute(Constants.STATUS_COORDINATOR_KEY,
```

```
        new MemoryStatusCoordinator());
        context.setAttribute(Constants.USER_COORDINATOR_KEY,
        new MemoryUserCoordinator());
        . . . . . .
    }
}
public class BaseAction extends Action
{
    public IProjectCoordinator getProjectCoordinator()
    {
    return (IProjectCoordinator)this.getServlet().getServletContext().
    getAttribute(Constants.PROJECT_COORDINATOR_KEY);
    }
    public IStatusCoordinator getStatusCoordinator()
    {
     return(IStatusCoordinator)this.getServlet().getServletContext().
     getAttribute(Constants.STATUS_COORDINATOR_KEY);
    }
    public IUserCoordinator getUserCoordinator()
    {
     return(IUserCoordinator)this.getServlet().getServletContext().
     getAttribute(Constants.USER_COORDINATOR_KEY);
    }
    . . . . . .
}
```

Struts1.X “Project Track” first utilizes a Listener class which is initialized when the web container starts up to load the detail data store classes into the “application” web scope, and then in order to make other “Action” classes make a direct correlation to our data store classes, we define a “BaseAction” class which contains several “get” methods to obtain the data information stored in the “application” web scope, when our customized “Action” extends this “BaseAction” class, it will automatically inherit the data binding relationship from the parent “get” methods.

5.5.2 WebWork2.2X

Before the version 2.2, WebWork utilizes a “Component Architecture” framework as its internal IoC container and implementation. Each class managed by “Component Architecture” has the right to define a specific interface which name starts with the class name and end with the “Aware” string, and register its name, scope (request, session, application, none) and associated interface to the framework. If a component class wants to bind with another, it just needs to implement the specific component

interface, the “Component Architecture” framework will automatically find the corresponding class and handle the binding relationship. Although the policy adopted by “Component Architecture” is straightforward and easy to use, it always needs developers to define a great number of extra interfaces, so as of WebWork 2.2, the “Component Architecture” has been deprecated (but not removed) and the official document recommends us to use “Spring” framework instead to fulfill the IoC needs.

If the IoC support we need is only limited to the “Action” class (which is often the case in web application), the WebWork “Interceptor” could also act as a simplified IoC container, which has the same principle as “Component Architecture”, to coordinate the relationship between domain classes and the “Action” classes. Listing 5.19 presents an example of how WebWork version “Project Track” utilizes our customized “Interceptor” to bind the web scope class to the “Action” class.

Case Study:

Listing 5.19 Data store class binding with “Interceptor” of WebWork

```
public class DataInterceptor implements Interceptor
    public String intercept(ActionInvocation arg0) throws Exception
    {
        Action action = (Action)arg0.getAction();
        Map context = arg0.getInvocationContext().getApplication();
        if(action instanceof UserAware)
        {
                ((UserAware)action).setUser
                ((IUserCoordinator)context.get(Constants.USER_COORDINAT
                OR_KEY));
        }
        if(action instanceof ProjectAware)
         {
                ((ProjectAware)action).setProject((IProjectCoordinator)context.
                get(Constants.PROJECT_COORDINATOR_KEY));
        }
        if(action instanceof StatusAware)
        {

                        ((StatusAware)action).setStatus((IStatusCoordinator)con
                        text.get(Constants.STATUS_COORDINATOR_KEY));
        }
        return arg0.invoke();
```

Listing 5.19 shows the essential code of customized “DataInterceptor” which is invoked before the action to make the judgment whether our “Action” class implements the “XXXAware” interface, if the “Action” does, this interceptor will invoke various “set” method to pass the corresponding datastore class instance to the “Action” class

Listing 5.20 “Action” class that implement “UserAware” interface

```
public class Login extends ActionSupport implements UserAware
{
    IUserCoordinator user;
    . . . . . .
    public void setUser(IUserCoordinator user)
    {
        this.user=user;
    }
    . . . . . .
    public String execute() throws Exception
    {
        User user=null;
        user=this.user.get(login,password);
        . . . . . .
    }
}
```

Listing 5.20 presents our “Login” action class which implements the “UserAware” interface, the “DataInterceptor” in Listing 5.19 will automatically establish the relationship between the “IUserCoordinator” implementation class and the “Login” action class without requiring developers to concern with the binding details. Comparing to the Struts implementation in Listing 5.18, the IoC implementation of “Interceptor” are more efficient and flexible: first the WebWork framework does not need to address the inheriting relationship between the classes, which will occupy the extra resources of the system, second the developers could flexibly choose the interface to implement according to their needs rather than implementing all of them

5.5.3 Tapestry4

Tapestry4 has introduced a new “property-injection” concept to support “IoC” features. By using “<inject>” element in the page specification file or using corresponding Java annotation tag in the page object, Tapestry4 framework could

easily inject page meta information, page object, JavaScript template file object, JavaBeans, service object, web scope object and web scope flag object (i.e. a Boolean object used to judge if the web scope object exists) into the page object as a property. As for the simple injection such as injecting a page object, meta information, JavaScript template file object and normal JavaBeans, Tapestry could fulfill the tasks only with the "<inject>" elements or Java annotation tag mentioned above, no extra XML configuration is needed since enough information is provided by them.. However, for the more complicated injection such as service object, web scope object and web scope flag object injection, Tapestry internally adopts "Hivemind" framework to help with addressing details, (the more details of "Hivemind" framework can be seen in [Apache Hivemind, 2007]) in this case necessary Hivemind configuration must be done in the "/META-INF/hivemind.xml" file.

Case Study

Listing 5.21 shows the example of how Tapestry version of "Project Track" application utilize its injection "IoC" feature to deal with predefined datastore class.

Listing 5.21 "property-injection" example of Tapestry framework

/META-INF/hivemind.xml:

```
<contribution configuration-id="tapestry.state.ApplicationObjects">
       <state-object name="projectCoordinator" scope="application">
          <create-instance
class="tapestry.projecttrack.domain.MemoryProjectCoordinator"/>
       </state-object>
       <state-object name="statusCoordinator" scope="application">
          <create-instance
class="tapestry.projecttrack.domain.MemoryStatusCoordinator"/>
       </state-object>
       <state-object name="userCoordinator" scope="application">
          <create-instance
class="tapestry.projecttrack.domain.MemoryUserCoordinator"/>
       </state-object>
</contribution>
```

Login class:

```
public abstract class Login extends BasePage {
   @InjectState("userCoordinator")
   public abstract IUserCoordinator getUserCoordinator();
```

```
    @Bean
    public abstract ValidationDelegate getDelegate();
    . . . . . .
    public String onSubmit(IRequestCycle cycle)
    {
        User user=null;
        try
        {
            user=getUserCoordinator().getUser(getLogin(),
            getPassword());
        }
        catch (ObjectNotFoundException e)
        {
            getDelegate().setFormComponent((IFormComponent)
            getComponent("logintext"));
            getDelegate().recordFieldInputValue(getLogin());
            getDelegate().record("Can not find the user.",null);
            return null;
        }
        . . . . . .
     }
}
```

In the “hivemind.xml” file we have configured three datastore classes into the web “application” scope, the “name” attribute of the “state-object” element indicates the corresponding property name in the page object, and the “class” attribute of the “create-instance” element indicates the detail implementation class for the page property. In our detailed page object class, we could simply use the sentence “@InjectState("propery-name")” to inject the property into our page object, and the following method defined after this sentence will be used to get the corresponding instance of this property. “Login” class also shows an example of injecting a JavaBean “ValidationDelegate” class, which has been stated in the “Validation” section, into our page object, the injection is achieved by using the Java annotation “@Bean” tag, and the corresponding instance of the class could be obtained by “getDelegate()” method defined after the tag.

5.5.4 JSF1.2

Managed bean is the essential unit of JSF framework which is introduced to help with the separation between presentation and business logic, it contains request properties, program logic and sometimes UI instance of JSF presentation

components (i.e. referring to “Back beans”). JSF framework supports powerful configuration mechanism to deploy the managed beans, the one of which relate to IoC feature support is managed bean property configuration which is used to initialize managed bean property values and establish the relationship between bean classes, following is an example of managed bean property configuration:

```
<managed-bean>
    <managed-bean-name>user</managed-bean-name>
    <managed-bean-class>contextPath.UserBean</managed-bean-class>
     <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
        <property-name>name</property-name>
        <value>me</value>
    </managed-property>
</managed-bean>
```

In the example we defined a “user” managed bean in the “session” scope and designate its “name” property with the “me” value. The default type of the property and initialized value is “String”. However, JSF also supports “List”, “Map” and other Java primitive types by specifying the “property-class” element for the managed property. The same process could also apply to bind other managed beans or customized classes with managed bean property, but there are two differences developers need to take care: the first is wiring scope permission rules which is listed in table 5.1; the second is that developers need to utilize the JSF EL to locate the binding class instance.

Defining scope	Compatible binding scope
none	none
application	none, application
session	none, application, session
request	none, application, session, request

Table 5.1 Compatible Bean Scopes [Geary and Horstmann, 2007]

Case Study

Listing 5.22 shows the example of how JSF version of “Project Track” application utilizes its managed bean property configuration to enable the binding relationship

with predefined datastore class.

Listing 5.22 JSF managed bean property configuration example

```
<managed-bean>
    <managed-bean-name>createProjectBean</managed-bean-name>
    <managed-bean-class>
        projecttrack.CreateProjectBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
      <property-name>projectCoordinator</property-name>
      <value>#{applicationScope.projectCoordinator}</value>
    </managed-property>
    <managed-property>
      <property-name>statusCoordinator</property-name>
      <value>#{applicationScope.statusCoordinator}</value>
    </managed-property>
    <managed-property>
      <property-name>userCoordinator</property-name>
      <value>#{applicationScope.userCoordinator}</value>
    </managed-property>
  </managed-bean>
```

Listing 5.22 presents the definition of “createProjectBean” managed-bean which binds various datastore classes in the application scope (described in section5.5.1) as its properties. As we can see in the example, the configuration utilize the expression “#{applicationScope.XX}” to locate related instance in the application scope, correspondingly, the “projecttrack.CreateProjectBean” class has to define properties with the same name in the configuration and related “set” methods to receive the instances specified by the JSF EL expression.

5.6 Post and Redirect

All interactive programs provide two basic functions: obtaining user input and displaying the results. Web applications implement this behavior using two HTTP methods: POST and GET respectively. [Jouravlev, 2004]. Generally speaking, GET method aims at retrieving the resources from the server, its parameters are used to nail down the response result and thus do not change the server state. On the contrast, the parameters of POST method contains the input from the web clients, the duplicated

submission of the same input data though POST may cause elusory exceptions or unnecessary system resource consumption, since the parameters can change the server state. In the *Redirect After Post* article [Jouravlev, 2004], Micheal Jouravlev concluded three types of double submit problem resulting form POST requests, they are listed in the below:

- reloading result page using Refresh/Reload browser button (explicit page reload, implicit resubmit of request);
- clicking Back and then Forward browser buttons (implicit page reload and implicit resubmit of request);
- returning back to HTML form after submission, and clicking submit button on the form again (explicit resubmit of request)

In this section, we discuss how different frameworks utilize their internal components to solve these problems.

5.6.1 Struts1.X

Struts framework makes use of a synchronous “Token” mechanism to control the duplicate post problems. The principle of synchronous “Token” is simple but effective, it randomly create a unique value according to the web clients session ID and the current system time, and save this value into the client “HttpSession” web scope. Next it nests this value into the submitting form in the form of hidden field so that the value could be transfer back to the server along with submitting. When clients submit the web form, it will check whether the “Token” value inside the “HttpSession” web scope is equal to the one inside the request parameters, if they are equal, which means no duplicate post problem in this case, the “Token” value should be deleted from the “HttpSession”, otherwise the page should be addressed with exceptions. The Struts framework mainly utilizes its “Action” class to perform the “Token” mechanism, developers could use the build-in method to create and validate the “Token” value in their specific “Action” classes.

Case study

Listing 5.23 shows the “Token” example used in the “Project Track” Struts web application:

Listing 5.23 “Token” example of “Project Track” Struts web application

```
public class TokenAction extends Action
{
    public ActionForward execute(ActionMapping mapping,
```

```
        ActionForm form, HttpServletRequest request,
        HttpServletResponse response) throws Exception
        {
            saveToken(request);
            return mapping.findForward("ToCreatePage");
        }
    }

    public class CreateAction extends Action
    {
        public ActionForward execute(ActionMapping mapping,
        ActionForm form, HttpServletRequest request,
        HttpServletResponse response) throws Exception
        {
            if(!isTokenValid(request))
            {
                saveToken(request);
                return mapping.getInputForward();
            }
            else
            {
                resetToken(request);
            }
            . . . . . .
        }
    }
```

Listing 5.23 is related to creating-a-new-project scenario which needs the users to input project information and then submit to the server, the code presented in the above is used to prohibit users from using web browse backward and forward or refresh button to resubmit the same information to create a project. The class “TokenAction” uses the saveToken() method to create a random “Token” value and insert it into the client “HttpSession” web scope, then it returns the code to ask the framework to send the creating-project page to the web clients. “CreateAction” class is the “Action” invoked after the user submit the form, it uses the isTokenValid() method to compare the “Token” value in the session scope and in the request parameters, if they are equal, the method returns true and the class invokes resetToken() method to delete the value in the session scope, next time when user duplicate post the form to the server, the isTokenValid() method returns false since no value existed in the session scope and thus the “Action” class re-invokes saveToken() method and asks the framework to re-display the creating-project page.

5.6.2 WebWork2.2X

WebWork makes use of the same “Token” principle as the Struts adopts, However, Instead of implementing the details into the “Action” class, WebWork utilizes “interceptor” to assist with “Token” processing, developers can achieve the similar effect of the Struts “Token” with a few configuration that the “interceptor” requires.
“Token” interceptor is counterpart interceptor for the “Token” processing, to make this interceptor work, developers needs to fulfil two tasks before running the “Action” class. The first task is to add the “<ww:token/>” tag into the form of our view page so that the corresponding “Token” value could be submitted to server, the second task is to configure the “Token” interceptor to the required “Action” class, which will verify various requests before executing the “Action”. In addition to basic implementation, the “Token” interceptor also add extra features to control the addressing behaviour, for example if we designate the “excludeMethods” parameter for the interceptor, the designated action method will be free from the “Token” verification, it will behave just like there is no “Token” interceptor at all. By default, when user double click, or use back button or refresh the page after the first submitting, the “Token” interceptor will always jump to new blank page and display “Form token XXXXXXX does not match the session token YYYYYYY” string to the user. To change the default behaviour, developers could appoint their customized page which will be displayed to the clients when “Token” verification falls to match “invalid.token” result string. Following is the example for this behaviour:

<result name="invalid.token">/TokenError.jsp</result>

If developers feel tied of showing the error message to the clients, they could choose the “token-session” interceptor which providing advanced logic for handling invalid tokens. Unlike normal token interceptor, the “token-session” interceptor controls an extra “actionInvocation” object in the HttpSession field, if the user breaks the “Token” rules, this interceptor will display the same result as this “actionInvocation” object indicates, which means the clients will always see their first submitting results if they double click, or use back button or refresh the page.

Case Study

Listing 5.23 shows the WebWork version of the “Token” usage.

Listing 5.23 “Token” example of “Project Track” WebWork web application

View page:

```
<ww:form action="createtoinbox!create.action" method="post">
```

```
  <ww:token/>
  . . . . . .
</ww:form>
```

Xwork.xml:

```
<action name="createtoinbox"
        class="webwork.projecttrack.actions.Create">
          <interceptor-ref name="validationWorkflowStack"/>
          <interceptor-ref name="token">
              <param name="excludeMethods">create</param>
          </interceptor-ref>
        <result name="success" type="chain">
          <param name="actionName">inbox</param>
        </result>
        <result name="input">/jsp/Create.jsp</result>
        <result name="invalid.token">/jsp/Error.jsp</result>
</action>
```

Listing 5.23 shows the example of using the “token” interceptor in WebWork, in the view page we put the “<ww:token/>” inside the form to supply the token value for the interceptor. Also notice that because we set the “create” to the “excludeMethods” parameter, so this form will not suffer from token validation because it regards “create” as the action method according to the string “createtoinbox!create.action”. To make this interceptor work, we should delete this parameter, then when any duplicated submitting problems occur, WebWork will send the “/jsp/Error.jsp” page back to the user which indicated by the "invalid.token" result string.

5.6.3 Tapestry4

Unlike Struts and WebWork framework, Tapestry4 utilizes the POST-REDIRECT-GET (PRG) pattern to solve the duplicated submit problems. PRG pattern divides the requests into two parts. Instead of returning an HTML page directly in response to the POST request, the POST operation returns the result page with a redirection command. Next time when user reloads or refreshes the browser, the browser will resend an "empty" GET request (not POST request as before) to the server, which does not contain any input data and does not change server status, it only loads the view page again.

To fulfill the PRG pattern, Tapestry4 must set signals into the listener methods to start up the pattern service since the listener methods is the necessary way Tapestry has to pass to consummate the POST request. The signals exist into the return value

of the listener methods, if Tapestry framework detects the return value is an instance of “ILink” class, it will automatically load requested page with the link and sent the page back to clients in redirect fashion. The case study section shows an example of this process.

Although the PRG pattern successful solves the implicit resubmit of request problems, it is helpless to the explicit resubmit of request, which means the user returns back to the submission page and re-clicks the submit button. To fetch up the this limitation, developers could fix it by careful domain model design or prohibiting page caching so that user can not be backward to the submission page after first submission. However, these measures still need extra efforts from developers and run the risk of changing the existed domain model. So when dealing with application with excessive explicit resubmit of request, the “Token” method is still the first choice for duplicated submit problems.

Case study

Listing 5.24 presents a code snippet of how Tapestry framework uses the “ILink” to fulfill the PRG pattern.

Listing 5.24 “Token” example of “Project Track” WebWork web application

```
public abstract class Create extends BasePage
{
  @InjectObject("engine-service:page")
  public abstract IEngineService getPageService();
  . . . . . .
  public ILink onSubmit()
    {
        . . . . . .
       return getPageService().getLink(false, "inbox");
    }
}
```

This code snippet background is same the one indicated in Struts and WebWork part, after managers fill in the new project information and submit the page, Tapestry framework will execute the onSubmit() method to create the new project instance and save it into the domain model, finally this method created and returns a page service link instance to the “inbox” view page. If we want to associate parameters with the ILink, we should configure the “inbox” page as a “IExternal” page

(described in 5.1.3 section) use the external service to generate the “ILink” instance like below:

return getExternalService().getLink(false, new ExternalServiceParameter(“pagename”, parameters));

5.6.2 JSF1.2

Same as Tapestry, JSF framework mainly utilizes PRG pattern to solve the double submit problem. However, comparing to the great effort spent on Tapestry framework, such as understanding of background knowledge and programming with special Java sentence, all developers need to do in JSF framework is revise the default “Forward” navigation fashion to “Redirect” in the configuration file using the tag “`<redirect/>`”.

Case study

Listing 5.25 “Token” example of “Project Track” WebWork web application

```
<navigation-rule>
   <from-view-id>/protected/edit/create.jsp</from-view-id>
   <navigation-case>
     <from-outcome>success_readwrite</from-outcome>
     <to-view-id>/protected/inbox.jsp</to-view-id>
     <redirect/>
   </navigation-case>
   . . .
</navigation-rule>
```

Listing 5.25 shows the usage of “<redirect/>” tag, as you can see in the above we have inserted a “<redirect/>” tag into one of the navigation cases of the “create.jsp” view page, after that when any users send requests from this view page and triggered action return with the “success_readwrite” result code, the application will send the “inbox.jsp” page in “Redirect” fashion.

6 Conclusions of Java web frameworks

The descriptions and corresponding “Project Track” code snippets in the previous chapter already presented the implementation characteristic of six web features of four chosen Java web framework. But what are the advantages and disadvantages of different framework feature implementations and how does the framework’s infrastructure and feature implementations influence the application types that it best fit in? To provide a first answer to this question, we focus on the first half in Sections 6.1, explicitly describing different framework implementation’s pros and cons. We summaries the classification of web application and discuss their suitability with respect to different frameworks in Section 6.2.

6.1 Web feature conclusion

6.1.1. Navigation rules

The Struts and WebWork framework make use of the “Action-oriented” and XML-based configuration to deploy the different navigation rules. This dispatching mechanism enforces the controlling effect of the “Action” class and greatly reduces the needs of direct dispatching between different pages, which makes the web application strictly conform to the MVC principle suggested by JSP Model 2. Furthermore, WebWork framework adds the “Package” and “Namespace” concept into the navigation which enables the “Action” classification into a logic layer and supplies a more elaborate control to the navigation. However, there are two shortages of “Action-oriented” mechanism used by these two framework, the first is that deploying the navigation rules is a comparatively complicated job since there may be thousand of actions used in one application and developers should always be care for the name confliction when they create the action (even though the situation get better when WebWork introduce the “namespace” concept for “Action”). The second is that the “Action-oriented” mechanism can not cover all the circumstances for web page navigation (see section 5.1.4).

Tapestry framework utilizes a programmable method to perform the page navigation. Each Tapestry view page could be invoked directly by the “ILink” component in previous page or invoked directly inside the previous page form methods (see section 5.1.3) without the extra XML configuration. However, the simplicity of Tapestry navigation is achieved at the price of two main aspects. The first aspect is that any change for the navigation in Tapestry would cause the troublesome ripple effect which includes recompilation of corresponding class file and consistency

modification (i.e. if we change a page name we must change corresponding navigation code that relate to this page). The second is that there is several navigation code fashions used in Tapestry, developers should be familiar with all of them in order to choose the suitable one to fulfill their specific needs.

JSF framework, in my opinion, has the best navigation mechanism, its page-based "input page - action result code - result page" navigation structure could not only emphasis the effect of action methods but also be able to cover all of web navigation circumstances. The only demerit of this navigation structure, which is that one page have two separate actions with the same result string or two action method expressions that return the same result string, could also be remedies by "<from-action>" XML tags.

6.1.2 Validation mechanism

Both of Struts and WebWork framework utilizes a XML configuration-based validation framework to support the business-logic-irrespective validation activities. These validation frameworks supply miscellaneous predefined validation rules so that developers could easily fulfill the complicated validation without extra effort of writing code. However, the disadvantage of using validation framework is that developers must spend effort to understand the grammar and usage of different validation rules, and it is hard to track a XML configuration grammar error unless we run the application.

Another advantage of Struts and WebWork Validation mechanism is that they both use a very simply fashion to display the error messages. Any error captured by validation frameworks or validation Java methods can be easily be displayed using the special customized tags, for instance, in Struts framework we use "<html:error/>" or "<html:errors/>" tags and in WebWork framework we use "<ww:fielderror/>" or "<ww:actionerror/>" tags. JSF framework also utilizes the similar mechanism to display the error messages, the tags JSF framework used are "<h:messages>" and "<h:message>".

Tapestry framework mainly uses its build-in or customized "validator" which contains special validation logic to perform the input data validation, each "validator" is associated with Tapestry input components, which greatly saves the configuration effort since the form and input field information is automatically set to "validator" when binding to the components. Although the validation logic that Tapestry framework can be expressed by its build-in "validator" is not as mush as the ones

supported by Struts and WebWork validation framework, they are enough to handle the problems that can be meeting in small or intermediate scale web application. The biggest shortage in Tapestry is its complicated error displaying mechanism, it does not abstract the functions well and exposes too much low level details to developers, comparing to Tapestry framework, Struts, WebWork and JSF do much better in this point.

JSF framework has a similar “validator” mechanism and thus has a similar advantage of Tapestry framework, however, its limited number of build-in “validator” and non-support of client-side validation make the validation a weak part of JSF framework, we hope the situation could better in the future.

6.1.3 Internationalization

Almost every Java web framework discussed in this thesis has an outstanding function support for internationalization, they all have a specific mechanism to manage different locale’s resource bundle files, they all supply customized view tag or presentation components to efficiently retrieve the internationalized messages and they all support dynamically changing the application response page locale according to client’s request. However, besides these commonalities there are still some issues or merits that need to mention for Struts, WebWork and Tapestry framework.

Struts framework supplies poor abstraction for manipulating response page locale. Although developers could easily understand the locale changing mechanism and simply manipulate them by changing the locale value in the “Httpsession” scope, this process exposes too much Java servlet low level details and it is hard to test the internationalized applications without putting them into the real running environment.

The over- dispersed resource bundle files management of WebWork framework can also be a problem, it is very difficult to search up or maintain an internationalized message within numerous resource bundle files. To overcome this issue, managers should make the most of centralized management fashion supplied by WebWork framework (e.g. using “ActionSupport.properties” resource bundle file described in section 5.3.2).

A preponderant advantage of Tapestry framework for internationalization is that apart from internationalized message, Tapestry utilizes the Java resource-bundle

style to manage other resources in the framework as well. The resources includes HTML template files, component specification files, image files, and Tapestry framework will intelligently choose the suitable files with the correct locale subfix according to the current engine locale.

6.1.4 Type conversion

Because all of the frameworks discussed in this thesis support automatic conversion well between string type and Java primitive types, so this part focuses on their implementation of customized class type conversion.

Struts framework utilizes “Commons-BeanUtils” to perform the customized class type conversion, however, the “Commons-BeanUtils” component only supports one way which is “String-> customized class” conversion, and it has no way to directly transfer the type conversion error message to Struts framework. Another problem of using “Commons-BeanUtils” component is that it does not have conversion scope control, all the conversion registered in “Commons-BeanUtils” component will be marked as global conversion.

WebWork framework has the most elaborate control of customized class type conversion, it has measures that could almost deal with every conversion situation in web development. The only problem of WebWork type conversion is that the configuration process is comparatively complex, several files and concepts need developers to create and understand, but once developers get familiar with the process, it will become much more efficient.

Tapestry utilizes the component-based “translator” perform the type conversion, each “translator” is closely bond with components and perform the conversion for the value associated with components. However, Tapestry has limited number of components that support its “translator” and the global conversion function can not be fulfilled in Tapestry.

JSF framework has a similar conversion mechanism as the one in Tapestry framework, but it overcomes all of the disadvantage that Tapestry framework has and supports both component scope (i.e. conversion is limited to specific component) and global scope conversion that WebWork framework supports. We can regard it as the second best framework that supports the type conversion.

6.1.5 IoC support

Struts framework does not have internal support for IoC feature, it must integrate with the third part software such as “Spring” to perform the functions.

Before the version 2.2, “Component Architecture” framework has been utilized as WebWork framework’s IoC container and implementation. However, because of the complexity and capability limitation, now it is deprecated, and WebWork suggests developers to use “Spring” IoC feature instead.

Tapestry framework utilizes the “property-injection” concept to support “IoC” features, different resources such as page meta information, page object, JavaScript template file object, JavaBeans, service object, web scope object and web scope flag object could be injected into the page object as a property, which is the simplest and most direct way to bind with class relationship. The only problem for “property-injection” is that the usage convenience is only limited to page object, the injection feature can not be used in any other class forms in Tapestry.

Among the four chosen frameworks, JSF is the only one that fully support the IoC features on its own. Based on the managed bean property configuration in the XML configuration file, JSF could easily establish classes` binding relationship no matter whether it is a managed bean class or it is a normal class, the only thing developers need to concern is the “Compatible Bean Scopes” which is described in table 5.1.

6.1.6 Post and Redirect

Struts and WebWork framework utilize a synchronous “Token” mechanism to tackle the duplicated submit problems. The advantage of using “Token” mechanism is that it could solve all of the three problems described in section 5.6 with little system resource occupied. Comparing to code fashion used in Struts framework which place all of the responsibility on the shoulder of developers, WebWork capsulates the common “Token” logic into a specific interceptor so that developers could simply reuse the “Token” mechanism by binding the Token interceptor with the required “Action”.

Tapestry and JSF framework adopts the “Post and Redirect” pattern as the duplicated submit problem’s solution, although the principle and the implementation of “Post and Redirect” pattern is much easier than “Token” mechanism, it can not solve the explicit resubmit of request problem as mentioned in section 5.6.3, we suggest

developers to create their own “Token” components if they choose JSF and Tapestry framework to build the application that is extremely sensitive to the explicit resubmit of request problem.

6.2 Recommended web application types for Frameworks

Struts1.X

In the past few years, Struts1.X has stopped been the favourite choice of web developers because of its non-support of high level abstraction and web feature limitation (e.g. type conversion, IoC and etc.). If you are going to build a web application require portlets or complex pages with lots of things going on [Raible, 2006], or more to the point, has the extremely needs of off-the-runtime-environment test, type conversion or IoC feature. We highly suggest using other framework instead. However, if you application does not belong to any cases described above and most of your crew are beginners or not familiar with other Java web frameworks, Struts framework is still the best choice since it is easy to train on and has immense community and documents supports.

WebWork 2.2X

WebWork framework which acts as the fundamental parts of Struts2 has gained more and more attention recently, its comprehensive web feature supports and well-designed interceptor architecture makes it a great choice when coming to the action-based framework. If you are not involving in an agile web project which always needs drag-and-drop UI development, if you project will constantly utilize plug-in and extensions and if you need various presented techniques to be used or altered, WebWork is absolutely the first choice.

Tapestry 4

The real strengths of Tapestry come through on medium to large sized projects [Raible, 2006], since in that case the high-efficiency characteristic (either the system running efficiency or development efficiency) brought by page pool mechanism and component-based nature could be highly embodied. Furthermore, because of excellent support of Internationalization and IoC web feature, Tapestry framework make huge bonus when the application concern with multiple locales and project teams that want to produce strong, testable code.

The only problem developers need take care is that Tapestry does not support the type conversion and Post and Redirect feature well. However, the problem is not

fatal and could be fixed by careful software structure design and additional programming.

JSF 1.2

Similar to Tapestry, JSF framework is also extremely suitable for a desktop-like agile project. Everything deployed by developers in JSF is contributed directly to the UI component property or UI component events and most of complicated UI component handling details are carried out by framework, which is the reason why JSF framework more understandable and has a much shorter learning period than Tapestry.

Because JSF takes care of managing the state of the UI for developers and there is no such performance optimizing mechanism (e.g. Tapestry page pool mechanism) existed in the framework, the overhead of which maybe make JSF not ideal for large, read-only web sites [Raible, 2006].

7 Summary

Java web framework is a set of related classes and other supporting elements that make Java web application development easier by supplying pre-built parts [Ford, 2004]. Frameworks become popular because they ease the complexity and enable web developers to write at a high level of abstraction without compromising the application content. However, to take full advantage of frameworks` benefit, necessary studies must be done to find out the optimum framework applied to the application. The main purpose of this thesis was to help web developers or technique managers gain deep insight of four popular Java web framework: Struts1.X, WebWork2.2X, Tapestry 4 and JSF1.2 through the comparison conducted in this these and try to conclude the best suited web application types of these frameworks. In order to achieve the research result several steps are taken:

First, the four chosen framework's infrastructure were investigated separately, the content includes framework introduction, framework key components and the lifecycle of the framework. The aim of investigating infrastructure of different frameworks was to reveal the general view of each framework and lays the understanding foundation for the web features comparison. After the research on the four chosen frameworks I concluded different framework's typical characteristics and some advantage and disadvantage at the end of chapter three.

Second, I selected six basic but essential web features, which are Navigation rules, validation mechanism, Internationalization, Type conversion, IoC support and Post and Redirect, as the comparison yardstick for different frameworks. In the chapter five, I first briefly introduced the concept of the six web features and then described detailedly the six feature implementations of each framework. Meanwhile in order to provide practical support, I also combined the theoretical discussion with presenting a "Project Track" case study web application.

Third, I carried out a conclusion on framework researches. The advantage and disadvantages of each framework feature implementation were summarized and based on the research results of framework infrastructure investigation and feature comparison, the suitable web application types for four chosen frameworks were also deduced at the end.

Choosing a suitable framework that best match the application from numerous peer software products is a time-consuming and complicated process since there is no one

web framework that will be best for all projects. Though the research result of this thesis, web developers or managers have the great opportunities to rapidly master the essential of the four popular frameworks and make the accurate framework choice according to the recommendation application types showed in this thesis. They may also make the judgment by themselves based on their project requirements and presented framework web feature implementation analysis.

Because of the constant web feature improvement and introduction of new features of frameworks, the future work of this thesis may focus more on feature analysis amendment, and framework recommended application types in this thesis should also be adjusted to make the research result consistent with framework status.

Reference

[Apache BeanUtils, 2007] *Apache Commons-BeanUtils user guide.* 2000-2007, The Apache Software Foundation.
http://commons.apache.org/beanutils/v1.8.0-BETA/apidocs/org/apache/commons/beanutils/package-summary.html#package_description

[Apache Hivemind, 2007] *Introduction to Hivemind.* 2005 Apache software foundation. Available as
http://hivemind.apache.org/hivemind1/index.html

[Apache Tapestry, 2003] *Tapestry developer guide.* 2000-2003 Apache software foundation. Available as
http://tapestry.apache.org/tapestry3/doc/DevelopersGuide/DevelopersGuide.html

[Apache Tapestry, 2006] *Tapestry Introduction.* 2006-2007 Apache software foundation
http://tapestry.apache.org/tapestry4.1/usersguide/index.html

[Chris, 2005] Chris Schalk, Oracle Corporation. Introduction to JSF, what is JSF? Unpublished manuscript, April 2005 Available as
http://www.jroller.com/cschalk/entry/online_introduction_to_jsf_presention

[Eric et al, 2006] Eric Jendrock, Jennifer Ball, Debbie Carson, Ian Evans, Scott Fordin, Kim Haase. *The Java™ EE 5 Tutorial Third Edition For Sun Java System Application Server Platform Edition 9.* Sun Microsystems, Inc 2006.

[Ford, 2004] Neal Ford, *Art of Java Web Development.* Manning Publications Co, 2004.

[Geary and Horstmann, 2007] David Geary, Cay Horstmann. *Core JavaServer™ Faces, Second Edition.* Prentice Hall 2007.

[Jouravlev, 2004] Micheal jouravlev, Redirect After Post. Unpublished manuscript, Auguest 2004 Available as
http://www.theserverside.com/tt/articles/article.tss?l=RedirectAfterPost

[Lightbody and Carreira, 2005] Patrick Lightbody and Jason Carreira, *WebWork in Action.* Manning Publications Co, 2005.

[Mann, 2005] Kito D. Mann. *JSF in Action.* Manning Publications Co, 2005.

[Marinescu et al, 2006] Floyd Marinescu, Frank Cohen, Doug Bateman, Adib Saikali, and Joseph Ottinger**. In:** *Sun's push of open source on a lot of levels in the Java stack, the rebranding of J2SE and J2EE, and the presence of two major technologies: JBI (Java Business Integration) and AJAX.* Also available as http://www.theserverside.com/tt/articles/article.tss?l=JavaOne_Day1.

[Opensymphony WebWork Wiki, 2006] *WebWork Wiki document.* 2000-2006 Opensymphony. http://www.opensymphony.com/webwork/wikidocs/WebWork.html

[Phil, 2005] Phil Zoio, JavaServer Face vs Tapestry –A head to head comparison http://www.theserverside.com/tt/articles/article.tss?l=JSFTapestry

[Raible, 2006] Matt Raible, Java Web Sweet Spot. Unpublished manuscript, March 23-25 2006 Available as http://www.virtuas.com/files/JavaWebFrameworkSweetSpots.pdf

[Raible, 2007] Matt Raible, Comparing Java Web frameworks. Unpublished manuscript, July 25 2007 Available as http://static.raibledesigns.com/repository/presentations/ComparingJavaWebFrameworks-OSCON2007.pdf

[Ship, 2004] Howard M. Lewis Ship. *Tapestry in Action.* Manning Publications Co, 2004.

[Sun, 2004] Weiqing Sun. *Master Struts: Java Web Design and Development Based on MVC.* Dian zi gong ye chu ban she 2004 (China).

[Sun J2EE Blueprint, 2005] *Core J2EE patterns* 1994-2007 Sun Microsystems, Inc. http://java.sun.com/blueprints/corej2eepatterns/index.html

[Tong, 2005] Ka Iok 'Kent' Tong. *Enjoying Development with Tapestry.* TipTec Development, 2005.

[Westkäper, 2004] Timo Westkäper. Architectural models of J2EE Web tier frameworks. University of Tampere, Dept. of Computer Science, Master Thesis 2004. Also available as
http://www.cs.uta.fi/research/theses/masters/Westkamper_Timo.pdf

[Zoio, 2004] Phil Zoio, JavaServer Face vs Tapestry A Head to Head Comparison Unpublished manuscript, Auguest 2005 Available as
http://www.theserverside.com/tt/articles/article.tss?l=JSFTapestry