CHARLES UNIVERSITY, PRAGUE
FACULTY OF MATHEMATICS AND PHYSICS

MASTER THESIS

Tomáš Mihalčin

Web Frameworks Comparison Concerning the Efficiency of Development

Department of Software Engineering
Advisor: RNDr. Tomáš Bureš, Ph.D.
Study Program: Computer Science

I would like to thank my advisor Tomáš Bureš for his valuable comments and advice. I would also like to thank my consultant Tomáš Krátky. His help and professional experience in the web frameworks area helped me to improve the quality of this thesis.

Názov práce: Porovnanie webových frameworkov s ohľadom na efektivitu vývoja
Autor: Tomáš Mihalčin
Katedra: Katedra softwarového inžinierstva
Vedúci diplomovej práce: RNDr. Tomáš Bureš, Ph.D.
e-mail vedúceho: bures@dsrg.mff.cuni.cz

Abstrakt: Skoro všetky podnikové webové projekty sú dnes implementované v nejakom webovom frameworku. Frameworky sa líšia vo filozofii, vlastnostiach a podpore pre ostatné populárne frameworky a technológie.
Cieľ tejto práce bolo porovnať frameworky Struts 2, Tapestry a Spring WebFlow a dať aplikačným programátorom a architektom prehľad o procese implementácie a zložitosti každého frameworku, aby sa mohli rozhodnúť, ktorý framework si vybrať pre svoj projekt..
Práca definuje kritériá pre porovnanie frameworkov. Porovnanie bolo vykonané na štúdii, ktorá bola implementovaná v Struts 2 a Tapestry a bol vykonaný výskum o možnostiach integrácie Spring WebFlow so Struts 2 a Tapestry. Proces implementácie bol detailne popísaný so všetkými problémami, ktoré sa objavili v implementačnej fáze. Nakoniec boli vykonané merania podľa definovaných kritérii a boli navrhnuté konečné odporúčania.
Kľúčové slová: webové frameworky, Struts 2, Tapestry, Spring WebFlow, porovnanie

Title: Web Frameworks Comparison Concerning the Efficiency of Development
Author: Tomáš Mihalčin
Department: Department of Software Engineering
Supervisor: RNDr. Tomáš Bureš, Ph.D.
Supervisor's e-mail address: bures@dsrg.mff.cuni.cz

Abstract: Almost all enterprise web projects today are implemented in some kind of web framework. The frameworks differ in philosophy, features and support for other popular frameworks and technologies.
The aim of this thesis was to compare Struts 2, Tapestry and Spring WebFlow frameworks and give the application developers and architects an overview about the implementation process and complexity of each framework to help them decide, which framework to choose for their projects.
The thesis defines criteria for the frameworks comparison. The comparison is done on a case study which has been implemented in Struts 2 and Tapestry and research on the integration possibilities of Spring WebFlow with Struts 2 and Tapestry has been done. The implementation process has been described in detail with all the problems that occurred during the implementation phase. Finally, the measurements have been made according to the defined criteria and final recommendations have been proposed.
Keywords: web frameworks, Struts 2, Tapestry, Spring WebFlow, comparison

Contents

1. Introduction

Web applications have gained much popularity in the past decade. More and more applications are being developed with web user interface due to the qualities these kinds of applications provide. Users may access them through a web browser anywhere in the world, without any kind of special software required. All they need is a connection to the Internet.

Web framework is a software framework designed to make development of the dynamic web applications easier by providing means that allow software developers to focus on the important parts of development. Tedious tasks such as session handling, localization, user input validation, etc., are taken care of by the framework with minimum amount of configuration. This makes use of web frameworks popular in web development environment, because developers do not have to reinvent the wheel with every new project, this is being done for them by the framework.

There are many Java frameworks available today. Some are designed for a specific purpose and others are meant for general use. Some are better for smaller applications with best performance in mind, some are better for large scale systems, where there is the need for more added functionality. Support for additional technologies like Spring framework, AJAX, Hibernate, iBATIS, EJB or others vary.

Focus of this thesis are new Java frameworks – Apache Struts 2, Tapestry 4.1 and Spring WebFlow. All of them have a different approach to development of web applications. Struts 2 is an action based framework opposing component based Tapestry, whereas WebFlow is designed as a support framework for capturing page flow of web applications.

All of the frameworks (or versions of frameworks) are relatively new and there has not been done a comparison on a common case study for all of them. Therefore, this thesis will use a case study, the same for all the frameworks, as basic means of comparison between them.

1.1. Goals

The main goal of this thesis is the comparison of above mentioned frameworks on a defined case study. The outcome of this thesis will be information that may be useful for developers, software engineers or software architects, when they are trying to decide which framework will be suitable for their intended purpose.

The sub-goals of this thesis include:

- **Definition of comparison criteria** – detailed list of criteria with their justification will be defined. Criteria will include time needed for development and modification of web applications, amount of source code written, size of generated HTML, response times for loading pages, support for other technologies such as AJAX, etc.
- **Definition of case study** – the case study will consist of 15 screens. It will contain several types of page flow. The case study will be implemented and later modified by adding, removing and modifying pages as well as page flows, therefore all of the common tasks of application modification will be included in measurements.
- **Case study implementation** – the case study will be implemented in Struts 2 and Tapestry. Research will be done on the possibility of integration of Struts 2 with WebFlow and Tapestry with WebFlow. Measurements of time spent developing and debugging will be collected during this phase.
- **Final measurements and comparison of results** – measurements of time spent developing and modifying the case study implemented in all of the frameworks will be done. Other measurements will include the size of source code files, size of generated HTML files, compliance with HTML standards, several measurements of frameworks response times under different conditions, etc.
- **Evaluation of the results** – the results will be evaluated and the final recommendations on the selection of an appropriate framework for a project will be presented.

1.2. Structure of the text

The Chapter 2 describes Struts 2, Tapestry and Spring WebFlow frameworks. The descriptions include framework history, architecture, example of use, requirements for framework usage and chapter listing supported technologies. All of the descriptions are aided by several examples of source code and framework configuration.

In Chapter 3, all of the comparison criteria are presented. Criteria are divided in three groups – benchmarks, features and subjective judgement. Each criterion is defined along with justification of its selection.

The Chapter 4 introduces the case study. Basic and modified GUI model, domain model and all the page flows will be presented here.

The Chapter 5 deals with the implementation of the case study in all of the chosen frameworks and frameworks combinations. Description of the implementation as well as the problems that occurred during the implementation are mentioned in this chapter.

Comparison results are presented in Chapter 6. The results of all measurements are provided with the comments on the outcome.

The Chapter 7 compares the results of this study with some relevant related studies which are available.

The final conclusion is included in Chapter 8. The evaluation of goals met are provided here. Recommendations on choosing a suitable framework for software developers are presented.

2. Frameworks description

In this chapter, Struts 2, Tapestry and Spring WebFlow frameworks will be closely described starting with some background information, continuing with architecture, examples of use and listing supported technologies and frameworks.

2.1. Struts 2

Struts 2 is an extensible open-source web framework that supports the full development cycle from building to deploying and maintaining applications [1]. It is a new web framework with GA release released in January 2007 and as all new frameworks it reflects progress in the development community by supporting most of the new technologies and frameworks and by using new methodologies and design patterns in design and implementation of the framework.

2.1.1. History

Apache Struts is an open-source framework originally developed by Craig R. McClanahan that was taken over by the Apache Software Foundation in 2002 [4]. Since then, it became one of the most popular web frameworks for Java and is currently the most used framework for Java web projects. Later on, a group of programmers separated from Struts development community and created a framework called Web Work. Now, the communities have joined together to create the Struts 2 framework, which is based on Web Work 2.2 version [1].

2.1.2. Architecture

Struts 2 frameworks follows the MVC Model 2 design pattern. This design pattern divides an application into 3 parts – Model, View and Controller. Struts 2 implements the Controller. The framework support various technologies in the Model and View parts.

- Model – application backend – Struts 2 allows to use many technologies including JavaBeans, Spring, Hibernate, iBATIS, EJB,etc. in the Model part.
- View – presentation layer – JSP, JSF, Velocity templates, Jasper Reports and other may be used in the View part. Struts 2 has a native support of JSP in this layer.
- Controller – layer implemented by Struts 2. This layer ties the Model and View together, processes events and responds to them with possible changes to the Model.

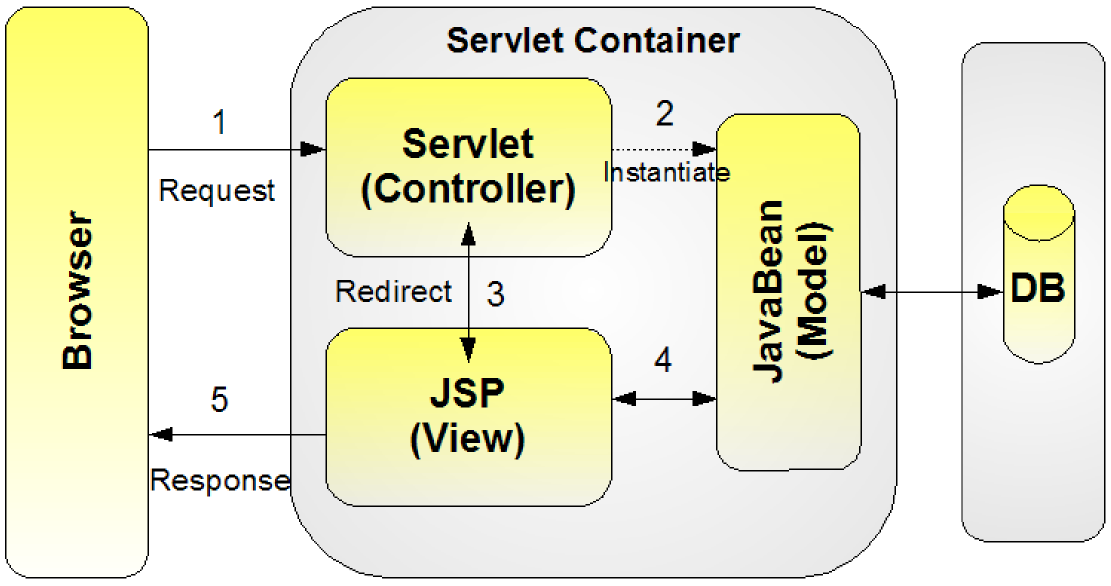


Figure 2.1 MVC Model 2 architecture

Typical request cycle is as follows: Web browser sends a request to a web server, where it is passed to the Controller. Controller may execute some business logic. Execution is then passed to the View layer, which creates response in interaction with Model (data transfer objects) and the response is finally sent back to the client.

Struts 2 follows this design pattern, however its architecture is much more complex (see figure 2). Initial request goes to the Servlet container, where it may be passed through several filters. Next, the required Filter Dispatcher is called, which consults the Action Mapper, which determines whether this request should invoke an action [3].

If the Action Mapper decides that some action should be invoked, the control is delegated by the Filter Dispatcher to the Action Proxy, which consults the Configuration Manager (initialized from struts.xml, the main configuration file) to find out which action class should be used to handle the request. Action Proxy then creates Action Invocation, which implements the command design pattern. Action Invocation invokes all required interceptors (defined in struts.xml. Interceptors are classes, which provide certain functionality such as logging, profiling, providing access to request or session, etc.) and finally invokes the action itself [3].

Upon the action return, the Action Invocation looks up the proper result in struts.xml. The result is executed, which typically involves execution of a template written in JSP, or in some other templating engine. During the rendering phase, the templates can make use of the Struts Tags provided by the framework. Interceptors are executed in reverse order and finally the response is returned to the client through filters configured in web.xml [3].

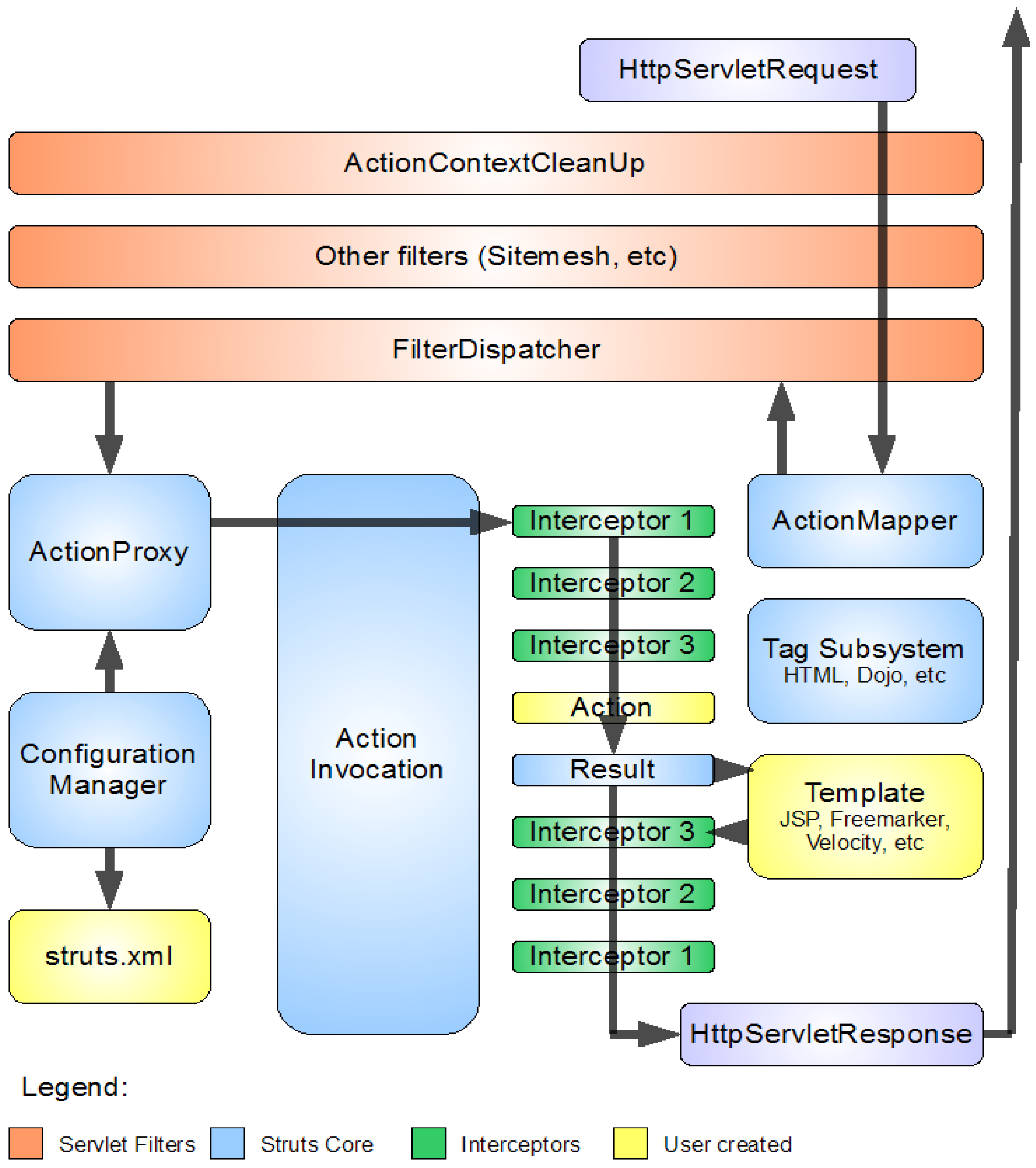


Figure 2.2 Struts 2 architecture

2.1.3. Example of use

There are 2 main configuration points. First is the deployment descriptor web.xml, which is common to all web applications. The other is struts.xml, where all configuration related to Struts 2 application takes place.

This example shows configuration of web.xml deployment descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>Struts2</display-name>

  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

</web-app>
```

Basic deployment descriptor configuration is short. All that is really needed is the Filter Dispatcher configuration, so the Servlet container knows, which class is responsible for handling requests. Additional information is required in case when advanced functionality is demanded (for example using Spring as an object factory for Struts 2).

Struts.xml configuration is more complex. This is the place where user defines Struts 2 configuration properties, interceptors, actions and exception handling behavior.

Struts.xml configuration is presented in this example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name="struts.devMode" value="true" />
  <constant name="struts.objectFactory" value="spring" />
  <constant name="struts.custom.i18n.resources" value="webcomparison.main" />

  <package name="default"  namespace="/“ extends="struts-default">
    ...
  </package>
</struts>
```

Struts.xml file includes definitions of constants for Struts 2 (such as object factory, main resource files, etc.), include files definition (so struts.xml can be broken to several files in larger projects) and finally package definitions.

Packages are the most interesting part of the configuration. That is the place where the action configuration takes place. Packages group actions together and define particular behavior for them.

This example shows a package configuration in Struts 2:

```
<package name="default" namespace="/" extends="struts-default">
        <global-results>
                <result name="exception">Error.jsp</result>
        </global-results>

        <global-exception-mappings>
                <exception-mapping result="exception" exception="java.lang.Exception" />
        </global-exception-mappings>

        <action name="Login" method="login" class="example.LoginAction">
                <result type="chain">Main</result>
                <result name="input">/pages/Login.jsp</result>
        </action>

        <action name="Main" class="example.MainAction">
                <result>/pages/Main.jsp</result>
        </action>
</package>
```

This package configuration defines behavior for two actions – Login and Main. The package extends the struts-default package, which is provided by the framework and defines several useful interceptors and interceptor stacks (groups of interceptors).

Namespace for this package is "/". Namespace tells which package will be used to find mapping for the client request. If the request URL is "/foo/bar.action", Action Proxy will try to locate action "bar" in a package with namespace "foo".

Global results are common for all actions, unless overridden in action configuration. Result from example tells that whenever result "exception" is returned from an action, Error.jsp will be returned to the client.

Global exception mappings define exception handling behavior. Whenever an action throws java.lang.Exception, the exception will be caught and result "exception" will be returned, which will in this case lead to Error.jsp shown to the client.

Last part of package configuration is the action configuration. Actions define name, class and method. Name is used to locate the action by the Action Proxy. The attribute class tells which class will handle the request and the optional attribute method specifies the method to execute (default value for the method is "execute"). After action finishes its execution, it has to return a result. The result determines, what will happen after the action execution. Typically an action will result in a template execution (such as JSP template), or in forwarding to another action. This is called action chaining.

Action classes are simple POJOs (Plain Old Java Objects). They do not need to extend any particular class, however extending ActionSupport class is recommended. ActionSupport provides access to resource bundles, message handling system and some other useful functionality.

Basic actions are not aware of Java Servlet API. Access to request, session and request parameters is done through implementation of the associated interfaces. IoC container (such as Spring) takes care of injection of the requested objects.

Action class implementation is shown in this example:

```
public class LoginAction extends ActionSupport implements SessionAware{
	private String login;
	private String password; //properties correspond with form input fields on associated JSP page
	private UserService userService; //business object injected by IoC container
	private Map session; //injected by IoC container due to implementing SessionAware interface

	public void setUserService(UserService service){
		this.userService = service;
	}

	protected UserService getUserService(){
		return userService;
	}

	public void setSession(Map arg0) {
		session = arg0;
	}

	protected Map getSession(){
		return session;
	}

	public String getLogin() {
		return login;
	}

	public void setLogin(String login) {
		this.login = login;
	}

	public String getPassword() {
		return password;
	}

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

	public String login(){
		User user = getUserService().getUserByLogin(login, password);

		if(user == null){
			addActionError(getText("badLogin"));
			return INPUT;
		}

		getSession().put(SessionAttributes.USER, user);

		return SUCCESS;
	}
}
```

Action handling code is implemented in method login. All methods, which are used for action handling must not have any parameters and must return object of type String. The result will determine the next action of the framework.

Action properties correspond with form controls. If the user inserts a text in the form input fields login and password these properties will be automatically populated during the action initialization. There is no need to look them up in any kind of map used in the request object. All of the work is done by the framework.

Same procedure can be applied for the preparation of form data. Action prepares its data, stores it in the action properties and forwards to the JSP page. All of the form controls will be filled with data stored in the properties.

This is an example of JSP page and written with Struts 2 tags:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
    <title><s:text name="title" /></title>
    <link rel="stylesheet" type="text/css" href="<s:url value="/theme/main.css" />" />
</head>
<body>
  <div class="loginForm">
        <h2><s:text name="welcome.key" /></h2>
        <s:actionerror />
        <s:form action="Login!login">
            <s:textfield key="login" />
                    <s:password key="password" />
                    <s:submit cssClass="button" align="left"/>
        </s:form>
  </div>
</body>
</html>
```

Struts 2 provides a large set of custom JSP tags, which make JSP development easier. All that is needed in order to use them is the taglib declaration on the top of the page. The tags include support for form input fields, collections iteration, localization, URL creation and others. All tags make use of OGNL, which is very powerful and flexible expression language.

Validation may be done in action classes or in XML validation files. Struts 2 provides all common validators including integer, string, double, required, range check, email, url and custom validators defined through regular expressions.

This example shows XML configuration of Struts 2 validation:

```
<!DOCTYPE validators PUBLIC  "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

<validators>
   <field name="login">
        <field-validator type="requiredstring">
             <param name="trim">true</param>
            <message key="requiredString" />
        </field-validator>
   </field>
   <field name="password">
        <field-validator type="requiredstring">
             <param name="trim">true</param>
            <message key="requiredString" />
        </field-validator>
   </field>
</validators>
```

2.1.4. Requirements

Basic Struts 2 distribution requires:

- Servlet API 2.4
- JSP API 2.0
- Java 5

Other requirements may be necessary in case of use of some specialized plug-in or technology.

2.1.5. Supported Technologies

Struts 2 integrates well with Spring, Hibernate, iBATIS and some other technologies and frameworks in the Model layer.

In the View layer, integration options include JSP, Velocity, FreeMarker, JFreeChart, SiteMesh, Tiles, Jasper Reports, JSF, AJAX (dojo toolkit or GWT) and others.

Struts 2 provides simple plug-in architecture which makes making of plug-ins for yet unsupported technologies easy.

2.2. Tapestry

Tapestry is a dynamic, robust and highly scalable open-source framework for creating Java web applications. Tapestry builds upon the Java Servlet API, but shields developers from direct interaction with it (URL construction, persistent state storage, validation, etc. are handled by the framework), therefore Tapestry provides higher level of abstraction than some other popular frameworks [7].

2.2.1. History

Tapestry was created by Howard Lewis Ship in the year 2000, and it has gone through a lot of development and through major code and functionality changes. In 2003, Tapestry has been adopted by Apache Software Foundation. Current stable version, described and used in this thesis, is version 4.1. Development version 5 has been rewritten from scratch and provides much more functionality and integration support for other popular frameworks and technologies [11].

2.2.2. Architecture

Tapestry follows the same MVC design pattern as Struts 2, although Tapestry is present in the Controller and the View part (with its native HTML templates). In the Model part, many technologies including Spring, Hibernate, iBATIS, EJB and others may be used.

Tapestry is a component based framework and as such components are its main focus. Every web page is generated from an HTML template file, which has an associated page class that provides

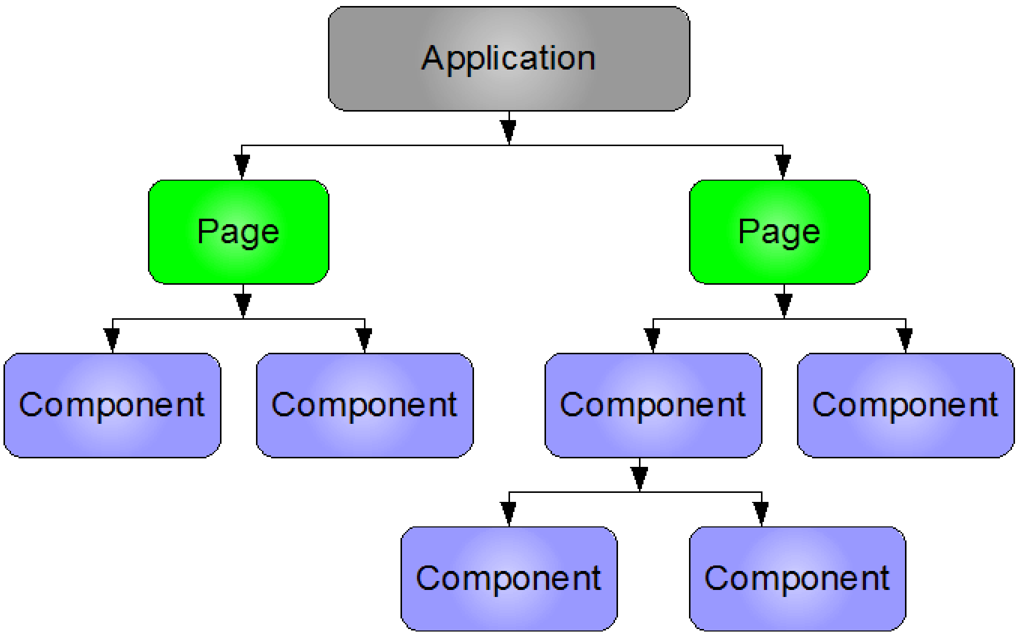


Figure 2.3 Tapestry scheme

business logic and data for the template. Pages are special types of components and may be built from yet another components (which may be also built from components).

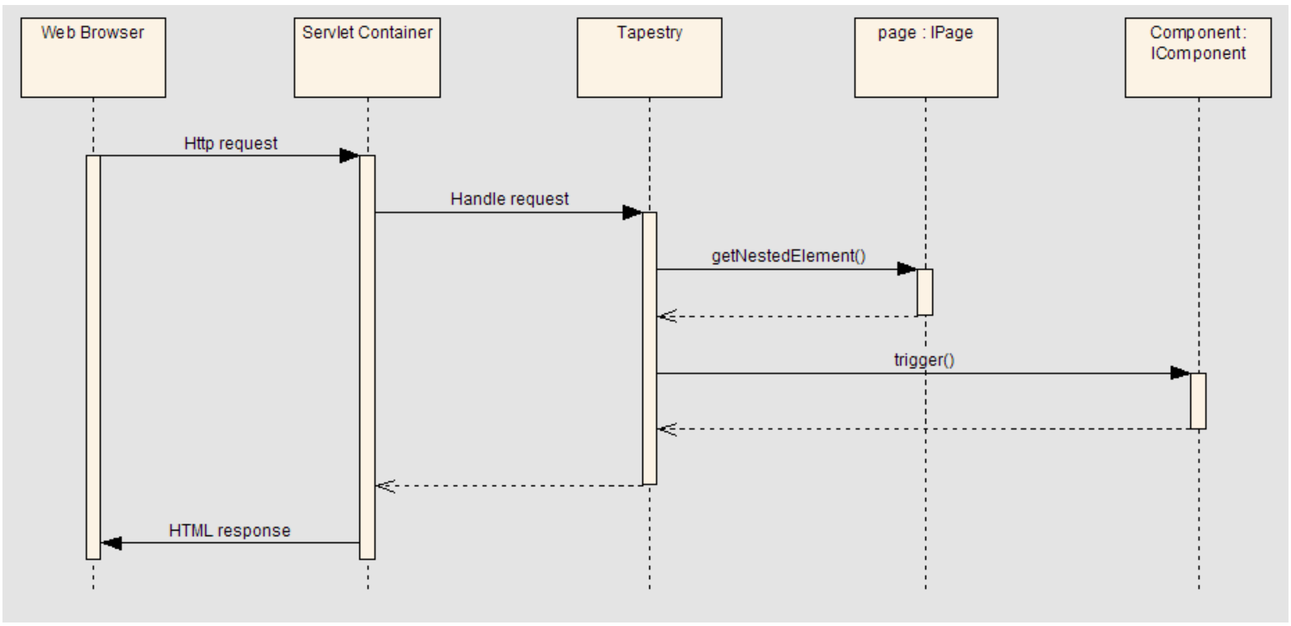


Figure 2.4 Tapestry request cycle

Request cycles in Tapestry are following: request from a web browser is handled by the Servlet container, which passes it up for processing by Tapestry. Tapestry looks up the page responsible for handling the request. The page may execute some business logic, prepares data and proceeds to rendering output according to the associated HTML template. The output generation is handled by the page, except on occasions when a component is used in the template. The component handles its own output processing and afterwards the execution returns to the page. The resulting output is finally sent back to the client.

2.2.3. Example of use

Tapestry has, unlike Struts 2, distributed configuration. There are 2 main configuration files: Web.xml, which is common for all Java web applications and a Tapestry specific app.application file. Tapestry is provided with native IoC container HiveMind designed and implemented specifically for Tapestry. Users should include hivemodule.xml, the HiveMind configuration file, in case of need of its services beyond the common use by Tapestry.

This example shows web.xml deployment descriptor in Tapestry:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
```

```
<web-app>
  <display-name>Tapestry</display-name>
  <servlet>
      <servlet-name>app</servlet-name>
      <servlet-class>org.apache.tapestry.ApplicationServlet</servlet-class>
      <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
      <servlet-name>app</servlet-name>
      <url-pattern>/app</url-pattern>
  </servlet-mapping>
</web-app>
```

Configuration of the deployment descriptor is short. Required configuration consists of definition of the ApplicationServlet, which handles the client requests and of the URL mapping for the defined servlet. Default value for the servlet mapping is "/app", which means the application URL will look like "http://www.host.domain/MyTapestryApp/app". This default value may be overridden, but it requires additional changes in the Tapestry configuration file.

The app.application file contains definitions of Tapestry configuration properties and page and component definitions. Pages and components are configured separately using required page/component file, optional properties files, optional class file and optional HTML template.

Main Tapestry configuration file – app.application – is shown in this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC  "-//Apache Software Foundation//Tapestry Specification 4.1//EN"
  "http://jakarta.apache.org/tapestry/dtd/Tapestry_4_1.dtd">

<application name="Tapestry">
    <meta key="org.apache.tapestry.page-class-packages" value="webcomparison.pages" />
    <meta key="org.apache.tapestry.component-class-packages" value="webcomparison.components" />

    <!-- Pages -->
    <page name="Home" specification-path="pages/home/Home.page"/>
    <page name="Main" specification-path="pages/main/Main.page"/>

    <!-- Components -->
    <component-type type="Menu" specification-path="components/menu/Menu.jwc"/>
    <component-type type="Language" specification-path="components/language/Language.jwc"/>

    <!-- Libraries -->
    <library id="contrib" specification-path="classpath:/org/apache/tapestry/contrib/Contrib.library"/>
</application>
```

The app.application configuration file contains configuration of Tapestry application such as specification of page class and component class packages. All of the pages and components must be specified along with the paths to their specification files. Additional component libraries may be specified. The contrib library, available with the framework, contains lots of useful components written by the Tapestry community.

This is an example of HiveMind configuration file – hivemodule.xml:

```
<?xml version="1.0"?>
<module id="webcomparison" version="1.0.0" package="webcomparison">

        <contribution configuration-id="tapestry.state.ApplicationObjects">
                <state-object name="context" scope="session">
                        <create-instance class="context.ContextObject"/>
                </state-object>
        </contribution>

        <service-point id="UserService" interface="services.UserService">
                        <create-instance class="services.impl.UserServiceImpl"/>
        </service-point>

        <service-point id="FlightService" interface="services.FlightService">
                        <create-instance class="services.impl.FlightServiceImpl"/>
        </service-point>
</module>
```

Hivemodule.xml manages the configuration of the IoC container. It allows the definition of objects with different scopes of validity. Important for web applications are objects with the session scope. Tapestry does not allow direct access to the session object, therefore the only way to keep session data is through definition of objects with the session scope. Other uses for HiveMind include definition of service objects, which may provide access to business logic/database and their direct injection to the page or component classes.

Page configuration in Tapestry is shown in this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE page-specification PUBLIC "-//Apache Software Foundation//Tapestry Specification 4.1//EN"
  "http://jakarta.apache.org/tapestry/dtd/Tapestry_4_1.dtd">

<page-specification class="webcomparison.pages.Home">
   <inject object="context" property="co" type="state"/>
   <inject object="Main" property="mainPage" type="page"/>
   <inject object="service:webcomparison.UserService" property="userService"/>

   <asset name="stylesheet" path="theme/main.css" />
</page-specification>
```

The default initial page for Tapestry has to be the Home page (this behavior may be overridden in the app.application configuration file). Page specification must contain path to the associated class file, otherwise the BasePage class will be used as the page class. The specification may contain definition of injected objects (such as the session scope object, other page classes or service classes), definition of page properties (these may be defined here or in the page class itself), configuration of components used in the HTML template and definition of asset files (alias for defined file in HTML template).

Page classes, if specified in the page configuration, must extend the BasePage class or any of its subclasses. Page class must be abstract. Class properties do not have to be specified, only abstract getters and setters must be provided. Finally, the listeners, which handle page events signaled by the client must be implemented.

Page classes are subclassed at runtime and their properties and corresponding getters and setters are supplied so they match the page class and the page configuration file.

This example shows page class in Tapestry:

```
public abstract class Home extends BasePage{
        public abstract String getLogin();
        public abstract String getPassword(); //page properties used in handling user input
        public abstract boolean getError();
        public abstract void setError(boolean error);

        public abstract ContextObject getCo();
        public abstract Main getMainPage();
        public abstract UserService getUserService(); //injected objects

        /**
         * login listener
         * validates login and password and forwards to main page on success
         *
         * @return MainPage
         */
        public IPage login(){
            User user = getUserService().getUserByLogin(getLogin(), getPassword());
                   if(user != null){
                        getCo().setLoggedUser(user);
                        return getMainPage();
                   }

                  setError(true);

                  return null;
        }
}
```

Listeners handle client actions. Listener may return object which identifies the page to forward to. The listener method may define parameters. One of them is the RequestCycle object which provides access to the Tapestry framework. Other parameters, if specified on the page, must match the number and types of parameters defined on the page.

Page class may implement interface which enables the class to handle events such as the beginning or the end of page rendering.

The HTML templates are files written in plain HTML. Tapestry components, which provide the dynamic output generation, are identified with special jwcid attribute, which stands for Java Web

Component id. Tags without this id are copied to the output when processed. Tags with jwcid specified are handled by corresponding components.

This example shows Tapestry HTML template:

```
<span jwcid="@Shell" title="message:loginPage" stylesheet="asset:stylesheet">

<body>
  <div class="loginForm">
     <h2><span key="welcome_message" /></h2><br />

     <span jwcid="@If" condition="ognl:error">
                <span key="badLogin" style="color: red;"/>
     </span>

     <span jwcid="@Form" listener="listener:login">
           <table class="loginTable">
              <tr>
                 <td>
                      <b><span jwcid="@FieldLabel" field="component:login" style="font-family: Verdana;
font-size: 14px;"/></b>
                 </td>
                 <td>
                     <span jwcid="login@TextField" value="ognl:login" validators="validators:required"
displayName="message:login"/>
                 </td>
              </tr>
              <tr>
                 <td>
                     <b><span jwcid="@FieldLabel" field="component:password" style="font-family: Verdana;
font-size: 14px;"/></b>
                 </td>
                 <td>
                     <span jwcid="password@TextField" value="ognl:password" hidden="ognl:true"
validators="validators:required" displayName="message:password"/>
                 </td>
              </tr>
              <tr>
                 <td colspan="2">
                     <span jwcid="@Submit" value="message:button.submit" class="button"/>
                 </td>
               </tr>
            </table>
     </span>
  </div>
</body>
</span>
```

Every component has a unique id and a defined set of attributes. These attributes may be bound to the page properties. This is usually the case of form components or iterators. Tapestry, like Struts 2, makes use of the OGNL expression language in the HTML templates.

Field validation is handled by the FieldLabel component. FieldLabel component is linked to other form component. It provides a label and a label decoration for the component (decorations may vary depending on the result of validation. For example if the validation fails, the label will be generated in different color) as well as the possibility of specifying certain validator(s). Tapestry provides standard set of validators and users may implement their own so there are no constraints on client input validations on the Tapestry side.

Component specification and implementation is very similar to the page specification and implementation. The main differences include: components have to define required attributes (these will be linked to components data structures, so the component may use them in some calculations) and the components must subclass the BaseComponent class, otherwise their use remains the same.

2.2.4. Requirements

Tapestry distribution requires:

- Java 1.2.2 (Java 1.4 is recommended)
- a microkernel such as HiveMind
- Java Servlet API 2.2 (version 2.3 is recommended)

2.2.5. Supported Technologies

Tapestry supports several of the most popular technologies including Spring, EJB, Hibernate, HiveMind and AJAX. There are several Tapestry extensions which aim to make the integration easier.

2.3. Spring WebFlow

Spring WebFlow is an open-source framework designed for capturing the application page flow. The main goal of this framework is to collect all the information about an application page flow in one place so the developer will not have to look it up in several places spread across multiple files.

2.3.1. History

Spring WebFlow has been developed as a part of Spring Framework mainly to support Spring MVC framework, but has been designed to support any of the Java web frameworks. The first public preview release of Spring WebFlow appeared in March 2005 and the first official release was released in October 2006.

2.3.2. Architecture

In the MVC design pattern, Spring WebFlow addresses the Controller part. The View and Model parts support the technologies of the main framework, which is aided by Spring WebFlow.

Spring WebFlow divides an application into smaller units called flows. The flow is a set of states, beginning with exactly one start state and ending with several end states. The flow may contain other flows as subflows.

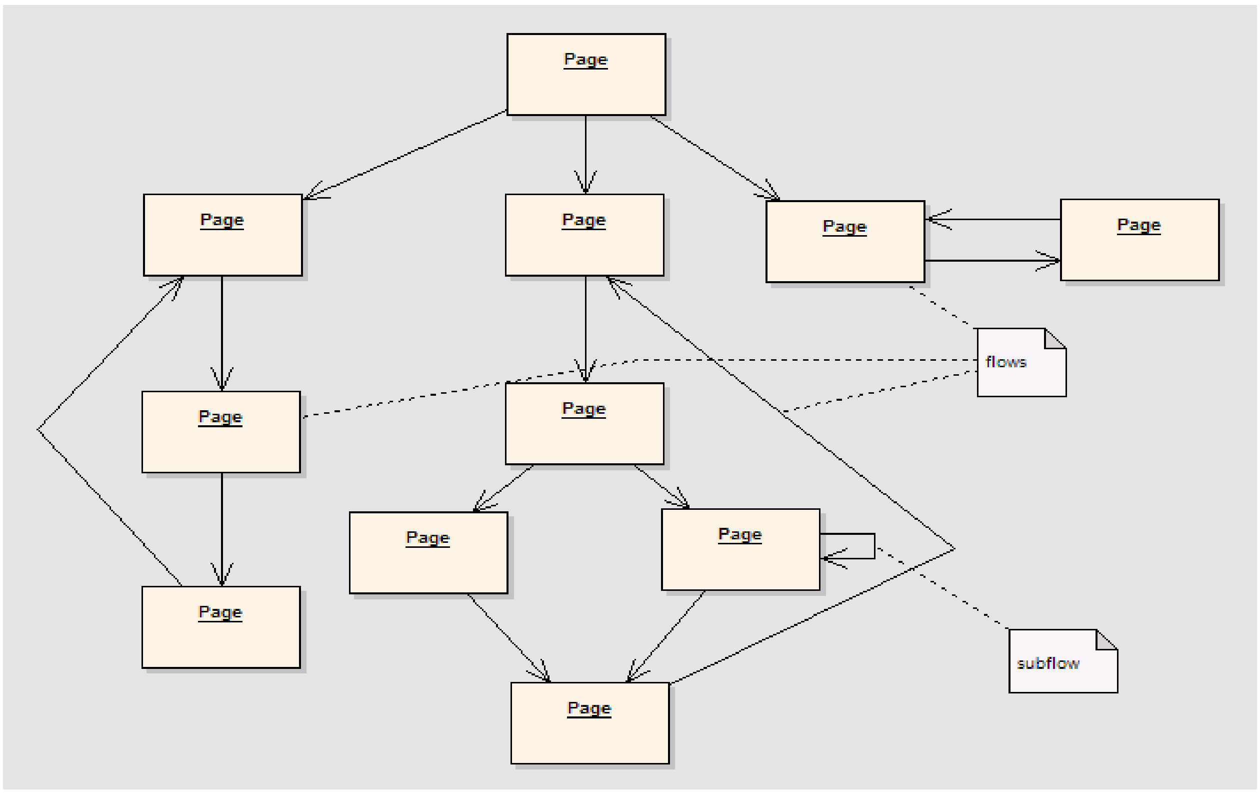


Figure 2.5 Application consisting of flows and subflows

States can be of two types: action states and view states. The action state executes some business logic while the view state interacts with user typically through a web page by processing events such as a form submit. The navigation between states is done through transitions. Each state defines its own set of transitions (global transitions for the whole flow may be defined as well) and depending on the outcome of the state (event signaled by the user in the view state or the result of a business logic execution in an action state) a transition is identified and the next state is chosen and executed. The framework keeps track of the current state of the execution and allows to execute only valid, defined transitions.

A state may define a set of entry actions and a set of exit actions so the source code may be broken down into small highly reusable pieces. Spring WebFlow also supports exception handling by allowing each state to define its own exception handler.

This example shows template of a state in Spring WebFlow [12]:

```
<?xml version="1.0" encoding="UTF-8"?>
   <flow xmlns="http://www.springframework.org/schema/webflow"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="
           http://www.springframework.org/schema/webflow
           http://www.springframework.org/schema/webflow/spring-webflow-1.0.xsd">
      <start-state idref="myStateId"/>

      <xxx-state id="myStateId">
         <attribute name="..." value="..."/>

         <entry-actions>
            ...
         </entry-actions>

         <transition on="..." to="..."/>
         <transition on-exception="..." to="..."/>

         <exit-actions>
               ...
         </exit-actions>

         <exception-handler .../>
      </xxx-state>

      </flow>
```

2.3.3. Example of use

Spring WebFlow has more complicated setup than the previous frameworks. The configuration is split into several files including configuration of Spring Framework. First to configure is the obligatory web deployment descriptor web.xml. Then follows the configuration of Spring MVC and Spring WebFlow in the Spring Framework configuration files and the last to come is the configuration of the flows in the Spring WebFlow configuration files.

Example of web.xml deployment descriptor configuration for Spring WebFlow [13]:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">

         <context-param>
                  <param-name>contextConfigLocation</param-name>
                  <param-value>
                           classpath:org/springframework/webflow/samples/phonebook/stub/services-
config.xml
                  </param-value>
         </context-param>

         <listener>
                  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
         </listener>

         <servlet>
                  <servlet-name>phonebook</servlet-name>
                  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                  <init-param>
                           <param-name>contextConfigLocation</param-name>
                           <param-value>
                                    /WEB-INF/phonebook-servlet-config.xml
                                    /WEB-INF/phonebook-webflow-config.xml
                           </param-value>
                  </init-param>
         </servlet>

         <servlet-mapping>
                  <servlet-name>phonebook</servlet-name>
                  <url-pattern>*.htm</url-pattern>
         </servlet-mapping>

         <welcome-file-list>
                  <welcome-file>index.jsp</welcome-file>
         </welcome-file-list>
</web-app>
```

The context-param and listener elements configure the Spring Frameworks usage. The following lines configure the Spring MVC web framework. The servlet element provides the name of the servlet class, which will handle all of the incoming requests, and supplies the servlet with the location of the configuration files.

This example shows configuration of Spring MVC for Spring WebFlow [13]:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.springframework.org/schema/beans
                                    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
```

```
          <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
                    <property name="prefix" value="/WEB-INF/jsp/"/>
                    <property name="suffix" value=".jsp"/>
          </bean>
</beans>
```

The Spring MVC configuration tells the framework where to locate the view pages. Each view in the view states from Spring WebFlow flow definitions will be located in the "/WEB-INF/jsp" directory and will be provided with suffix jsp. That means that when the Spring MVC will want to render a view with name login, it will use "/WEB-INF/jsp/login.jsp" page to render it.

This example shows configuration of Spring WebFlow in the Spring Framework configuration files [13]:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:flow="http://www.springframework.org/schema/webflow-config"
     xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
        http://www.springframework.org/schema/webflow-config
        http://www.springframework.org/schema/webflow-config/spring-webflow-config-1.0.xsd">

          <flow:executor id="flowExecutor" registry-ref="flowRegistry"/>

          <flow:registry id="flowRegistry">
                    <flow:location path="/WEB-INF/flows/**-flow.xml"/>
          </flow:registry>
</beans>
```

This configuration properly sets up the internals of the Spring WebFlow. By changing the flow:location path attribute, it is possible to modify the location, where will the Spring WebFlow search for its flow definitions.

This is an example of flow with a view state configuration in Spring WebFlow [13]:

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/webflow
                    http://www.springframework.org/schema/webflow/spring-webflow-1.0.xsd">

          <start-state idref="enterCriteria"/>

          <view-state id="enterCriteria" view="searchCriteria">
                    <transition on="search" to="displayResults"/>
          </view-state>

          <view-state id="displayResults" view="searchResults">
                    <transition on="newSearch" to="enterCriteria"/>
```

```
		<transition on="select" to="...detail module..."/>
	</view-state>
</flow>
```

This flow example contains two view states. A flow must define a start state, so the enterCriteria state will also be our start state. Upon entering the view state a searchCriteria view will be rendered (that means "/WEB-INF/jsp/searchCriteria.jsp" will be executed). Spring WebFlow will wait for users response "search" and then it will invoke a transition to the displayResults view state.

Spring WebFlow offers more configuration options including execution of actions upon entering or exiting the view state or upon rendering the view. All of the configuration options may be found in Spring WebFlow documentation [14].

This is an example of a JSP page written using Spring WebFlow tags [13]:

```
<form:form commandName="searchCriteria" method="post">
<table>
	<tr>
		<td>Search Criteria</td>
	</tr>
	<tr>
		<td colspan="2">
			<hr/>
		</td>
	</tr>
	<spring:hasBindErrors name="searchCriteria">
	<tr>
		<td colspan="2">
			<div class="error">Please provide valid search criteria</div>
		</td>
	</tr>
	</spring:hasBindErrors>
	<tr>
		<td>First Name</td>
		<td>
			<form:input path="firstName" />
		</td>
	</tr>
	<tr>
		<td>Last Name</td>
		<td>
			<form:input path="lastName" />
		</td>
	</tr>
	<tr>
		<td colspan="2">
			<hr/>
		</td>
	</tr>
	<tr>
		<td colspan="2" class="buttonBar">
			<input type="hidden" name="_flowExecutionKey" value="${flowExecutionKey}"/>
			<input type="submit" class="button" name="_eventId_search" value="Search"/>
		</td>
	</tr>
```

```
</table>
</form:form>
```

The JSP page is a standard JSP page written in Spring MVC framework. The only difference is the need to include two special values into every form and every link on the page. Flow execution key tells Spring WebFlow the state of the application and the eventId parameter tells which event will be signaled and which transition to invoke.

This example shows a flow with an action state configuration [12]:

```
<?xml version="1.0" encoding="UTF-8"?>
  <flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
         http://www.springframework.org/schema/webflow
         http://www.springframework.org/schema/webflow/spring-webflow-1.0.xsd">

    <start-state idref="executeSearch"/>

    <action-state id="executeSearch">
      <action bean="searchAction" method="executeSearch"/>
      <transition on="success" to="displayResults"/>
    </action-state>

  </flow>
```

Action state definition resembles the view state definition. The difference is that the action state must define an action to be executed and does not declare a view page. Action state will execute an action and upon the result of the action the framework will decide what will be the next state.

2.3.4. Requirements

Spring WebFlow distribution requires:

- Spring Framework 1.2.7
- Java 1.3
- Java Servlet API 2.3

2.3.5. Supported Technologies

Spring WebFlow is a support framework for other web frameworks and by itself does not provide support for any technologies. Supported technologies will be those, which the main web framework supports.

3. Criteria description

This chapter contains detailed list of the comparison criteria along with the justification of their selection. Criteria are divided into three chapters – benchmarks, features and subjective judgement.

Some of the benchmarks have been made during the implementation of the case study in all of the selected frameworks. The rest of the benchmarks, features and subjective judgement criteria has been described for every implementation of the case study after the implementation has been done.

3.1. Benchmarks

This chapter contains a list of the measurable criteria. All of the criteria here will be precisely measured and the results will be shown in the chapter 6.

- **time spent developing pages** – it is very important to know the approximate time spent creating and debugging pages. The more time the project needs to be finished the more expensive the project will be, therefore this has been one of the most important criteria measured. The measurements include time spent developing and debugging each single page as well as the average of time needed to develop 1 page. The measurements have been done during the implementation of the case study.

- **time spent modifying existing application** – time spent developing the basic application is important, but maintenance has to be accounted for as well. The time spent modifying existing application (adding, removing, joining pages) tells how good is the framework configuration organized and how expensive will be the maintenance of the application in the future. The measurements include the time spent modifying pages and the number of (configuration) files changed to accomplish the task.

  Four modifications to the case study implementations have been done. The modifications comprise of joining pages, removing pages and adding new pages. The measurement of time spent modifying the applications has been done during the modification process. The modifications are more closely described in the next chapter.

- **the size of written code** – this criterion is closely related to the first criterion (time spent developing pages). It will help to estimate the overall time spent developing the application. If the results will not be approximately the same as the first criterion results, it may indicate that one framework may be more complicated to understand and use than the other. This is important for developers new to the framework. It may take more time to be fully productive with framework with steeper learning curve. This criterion may also show the benefits of using the component based framework over the action based, because in the component based framework, the code reuse is expected to be on much higher level. The measurements will be divided into the size of written configuration, class pages and view templates (either JSP in Struts 2 or HTML templates in Tapestry). The results consist of measurements of the size (in bytes) of all of the files used to implement each page and of average size of files per one page.

- **size of generated HTML code –** nowadays, most people have access to high speed Internet connection, but people need to access information systems from various places at different time, so the amount of generated HTML code is still important. Less code means less time spent waiting for the page to load. People do not like waiting for a long time for the application response and if the time is too long, they will search for alternatives.

  Every framework generates the web pages in a different way and includes various metadata and JavaScript in the resulting code, therefore the resulting size of the generated HTML pages may vary. The size of generated HTML code has been measured in bytes on a per page basis. Results also include the average size of 1 page.

- **frameworks performance** – frameworks performance along with the size of generated HTML code affect the page loading time. Even if the HTML code is small in size, it may still take long time to load the page due to low performance of the framework.

  The measurements consist of two tests. One has been measuring repeated requests for one page, the other has been cycling through all of the pages multiple times.

  The JMeter [22] tool has been used for this benchmark.

- **typical amount of files per 1 page** – this criterion tells how complicated is the page configuration. Less files means less configuration and better orientation in the source code for the developer. If the page needs too many files to configure, changes take more time and are more confusing to do.

  The measurements include the amount of files needed to implement every page as well as the average amount of files needed to implement one page.

- **page correspondence with W3C standards** – it is important to know if the HTML generator follows W3C standards for HTML. Different people use different web browsers and in order for pages to display correctly they have to follow the standards.

HTML 4.01 Transitional and XHTML 1.0 Transitional standards have been checked using the W3C validator [20] and the resulting number of validation errors has been included in the results chapter along with the average number of errors per one page.

3.2. Features

This chapter contains criteria that focus on various features, which are provided by most of the current frameworks. The comparison results include yes or no answer to support of a certain feature along with more information on the quality of the support.

- **localization support** – most of the current frameworks provide some kind of support for localization. All of the localization possibilities have been mentioned in this comparison.
- **validation support** – like in the previous point, most frameworks support this feature. Differences are in the quality of the support. Some frameworks provide only field validation, some allow definition of complex form validations, client side validations or even specifying own validators. The quality of validation support has been discussed in this comparison.
- **AJAX support** – AJAX is relatively new technology that gained popularity very fast upon its introduction. In many projects, AJAX is used to minimize the traffic between the client and the server. New frameworks usually support AJAX in a native way, although not all do and it may require a good deal of configuration to make it work. Also supported AJAX toolkits vary. The quality of AJAX support has been discussed in this comparison.

3.3. Subjective judgement

This chapter contains criteria, which represent personal opinions about the experience with development with the selected frameworks.

- **technology complexity** – this criterion is closely related to the amount of time spent on the development and the size of written code. This is a personal opinion that sums up the previous two mentioned criteria and may be useful in deciding which framework to pick up.
- **complexity of orientation in existing project** – this criterion sums up two of the previous criteria – typical amount of files per page and time spent modifying existing project along with personal opinion about the frameworks complexity.

- **complexity of setting up a new project** – describes whether all requirements for the project setup can be easily found and acquired, how long did it take for the project from downloading requirements to running the project and whether there were any problems during the setup.
- **quality of support** – states the quality and the amount of documentation provided by the community which supports each framework along with the support provided by the community upon solving framework specific problems.

4. Case study

This chapter covers the case study. The case study is an application consisting of 15 screens and represents a part of an aviation company management software. The application deals with the management of users, flight crews, aircrafts, destinations and flight planning. The application is divided into several modules. User with different roles may access different modules, therefore a login screen with password checking will be implemented. The application is localized in the English and Czech language.

At first, the GUI model will be described, following with domain model and the list of used page flows along with their justification. At last, the case study modifications will be presented.

4.1. GUI model

The application is divided into several modules. First is a simple login page, where the user is required to enter his login name and password. All pages and user actions pass through access privileges checking routine to ensure that the user may access only those pages for which he is authorized. After the login page follows the main page with menu, which gives access to all of the applications modules.

Pages and access rights are divided into 4 groups (modules): administration, aircraft, planning and crew.

Users with administration rights have access to the administration module. They may add, modify and delete all users in the system. Administration module consists of 2 pages. The first contains list of all users with filtering option enabled. The second page serves for editing users.

Aircraft module is used for aircraft administration purposes – it enables the users to add, modify or delete the aircrafts in the system. The first of 2 pages contains a simple list of all aircrafts and the second is used for editing aircrafts.

Planning module serves the crew members. It allows them to look up the information about their planned flights for several time periods.

The largest is the crew module. It allows privileged users to add, modify or remove special types of users – pilots and stewards. Module consists of 8 pages starting with the list of crew members. The next is the page where may be the common data for pilots and stewards edited. Other pages include pages for editing specific data for pilots and stewards, pages for entering stewards language skills and pages for entering crew members contacts.

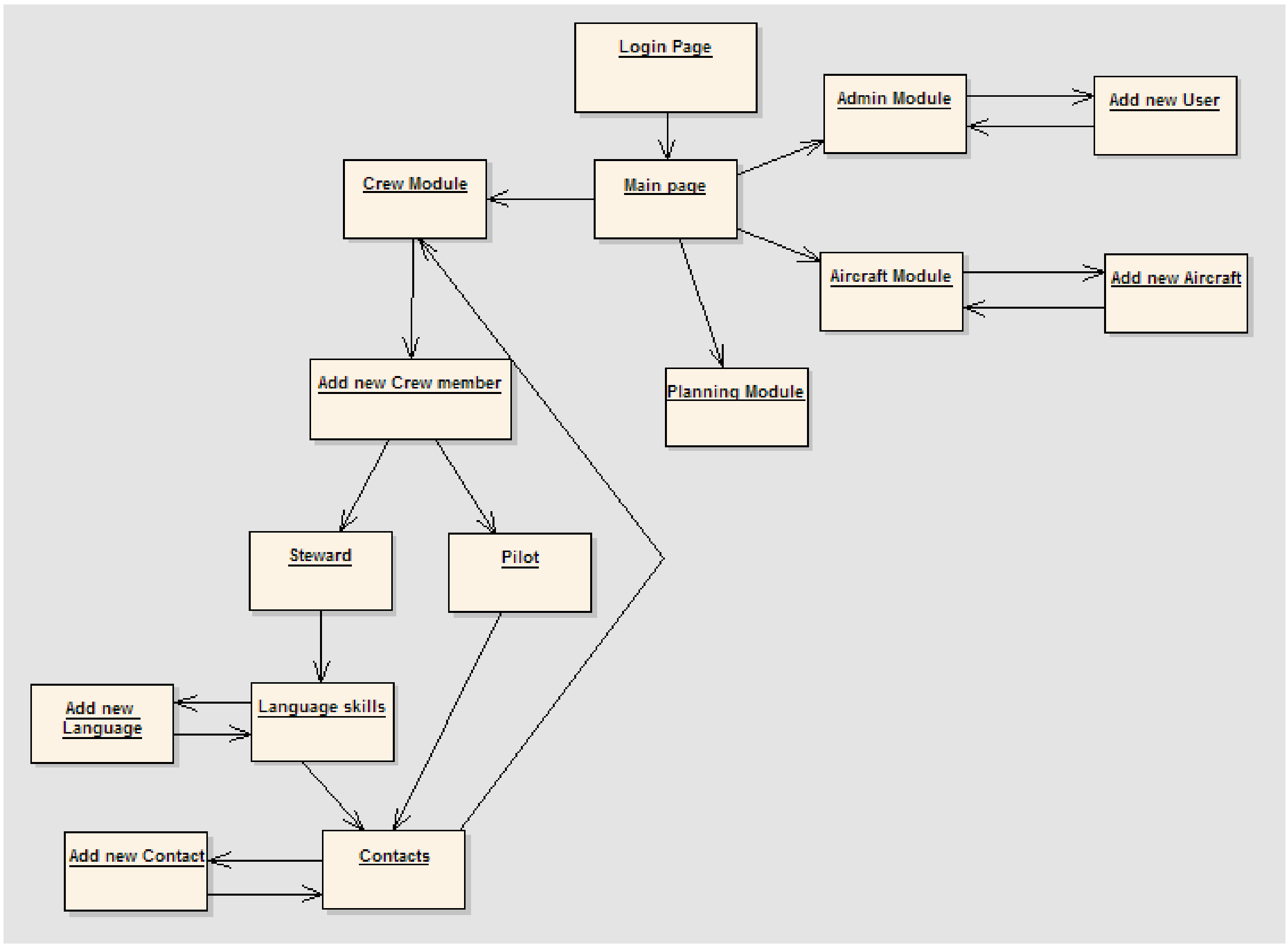


Figure 4.1 Case study page flow model

4.2. Domain model

The domain model consists of 9 JavaBeans and 2 service classes, which provide the business logic. The domain model provides the infrastructure upon which the application is built. It remains the same for all of the implementations to ensure that it does not affect the performance benchmarks.

Four of the classes represent the users in the system. User class is the base class for users and contains basic information about a person, along with login, password and access rights information. This class is used for the basic user.

Two other kinds of users exist in the system: Pilot and Steward. Both subclass the Crew class, which subclasses the User class. Crew class holds data common for both Pilot and Steward. Pilot and Steward classes add their own specific information.

Crew class contains a list of Contacts and a list of Flights. Flights are connected to the Destination class to identify the departure and arrival destinations.

Pilot class holds the information about pilots current Aircraft certifications, and at last, the Steward class contains a list of references to Language class, which represent his language proficiencies.

There are 2 service classes: FlightService and UserService. FlightService contains Aircraft and Flight operations while the UserService provides methods for working with Users.

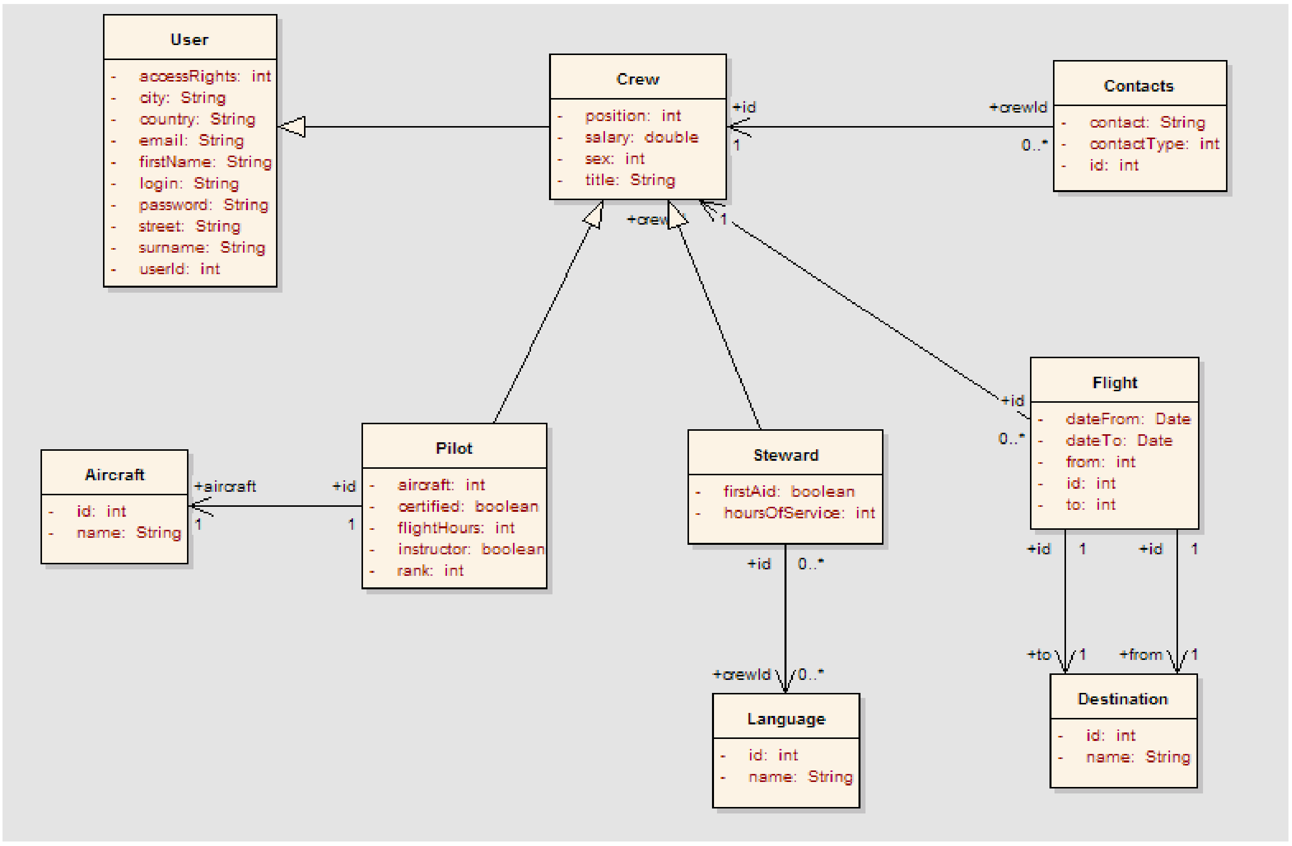


Figure 4.2 Case study domain model

4.3. Page flows

As can be seen on Figure 6, there are several types of page flows present in the case study:

- **star page flow** – menu is a typical example of the star page flow. It is directly connected to several different pages. In this case study, the star page flow connects the Main page to all of the main module pages – Crew, Aircraft, Planning and Administration.

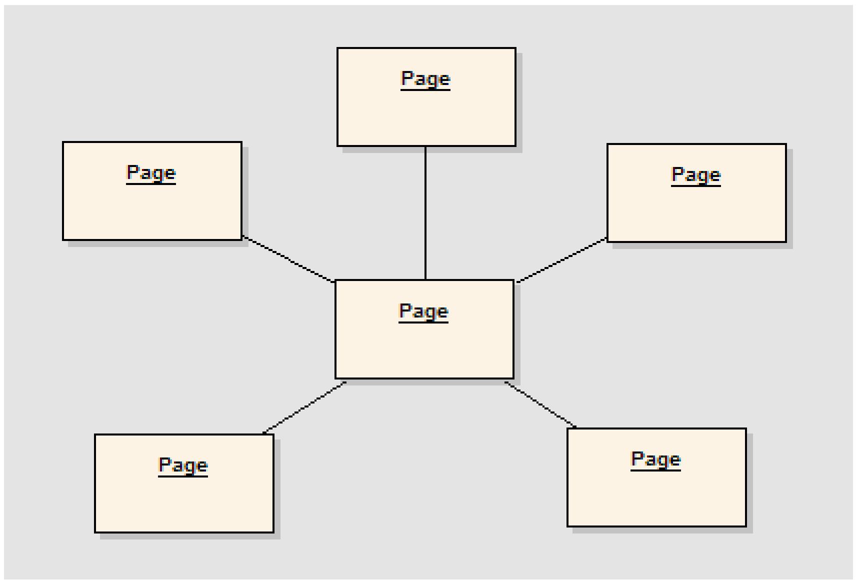


Figure 4.3 Example of star page flow

- **cycle page flows** – these can be found in the Administration module, Aircraft module and two cycles which act as subflows may be found in the Crew module (Language and Contacts pages).

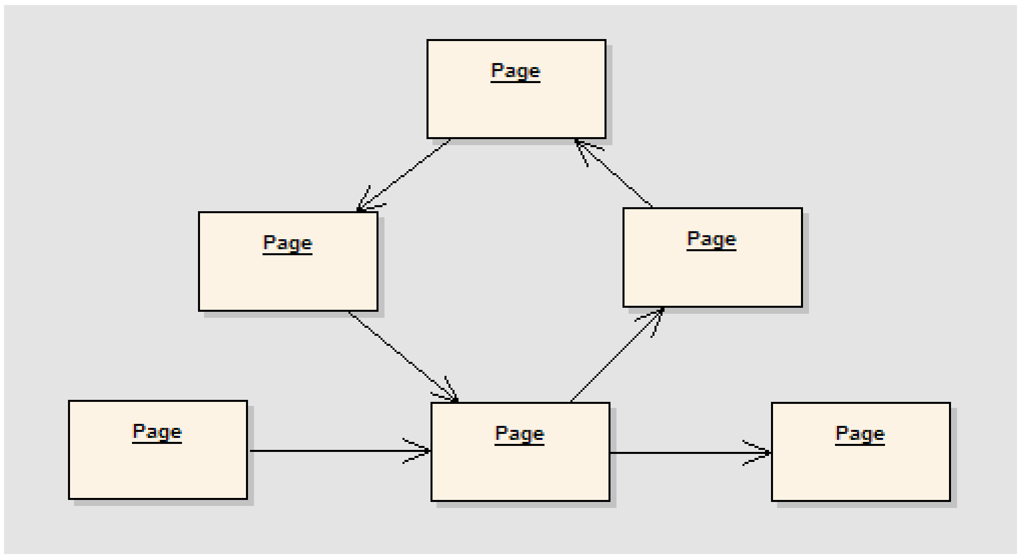


Figure 4.4 Example of cycle page flow

- **master-detail page flow** – all of the main module pages contain master-detail page flow. Master-detail contains some filtering options which are followed by displaying a list of objects with the option of viewing detail of one selected object.

- **fork and join** – the flow divides in one place and joins in another. This is shown in the process of adding new Crew member.

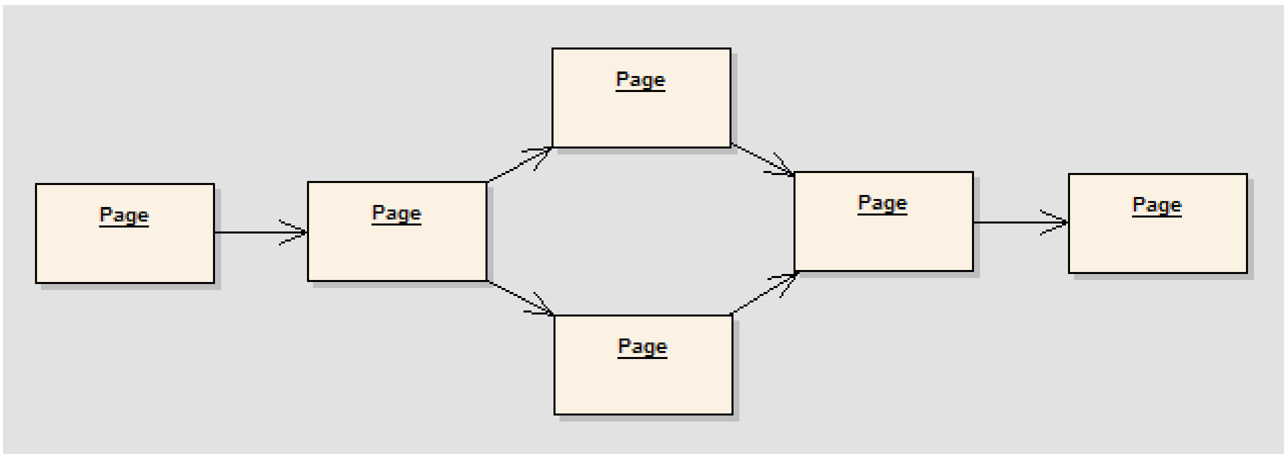


Figure 4.5 Example of fork and join page flow

- **wizard** – it is a logical sequence of related pages. It can be found in the Crew module.

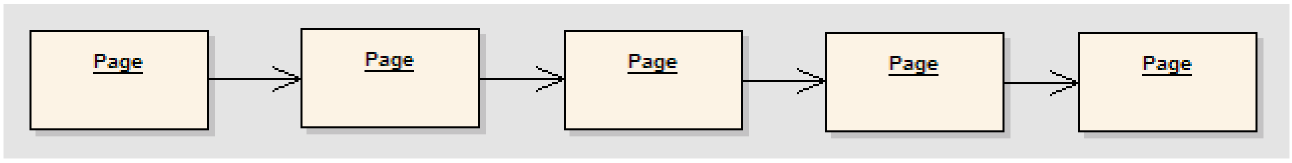


Figure 4.6 Example of wizard page flow

4.4. Modifications

The modifications of the case study are the subject of significant part of the measurements. Following modifications have been implemented:

- **joining two pages** – Steward and Language skills pages have been joined together. The Steward page went through major modification.
- **removing a page** – the Planning page has been removed. The menu component had to be modified.
- **adding a page** – new page for Destinations management has been added to the Administration module. The Administration page has been modified as well.
- **adding a page** – new page has been added to the Crew module. A Summary page has been inserted before the Contacts page. The two forked flows join in this page, so it has been a major modification to the flow itself.

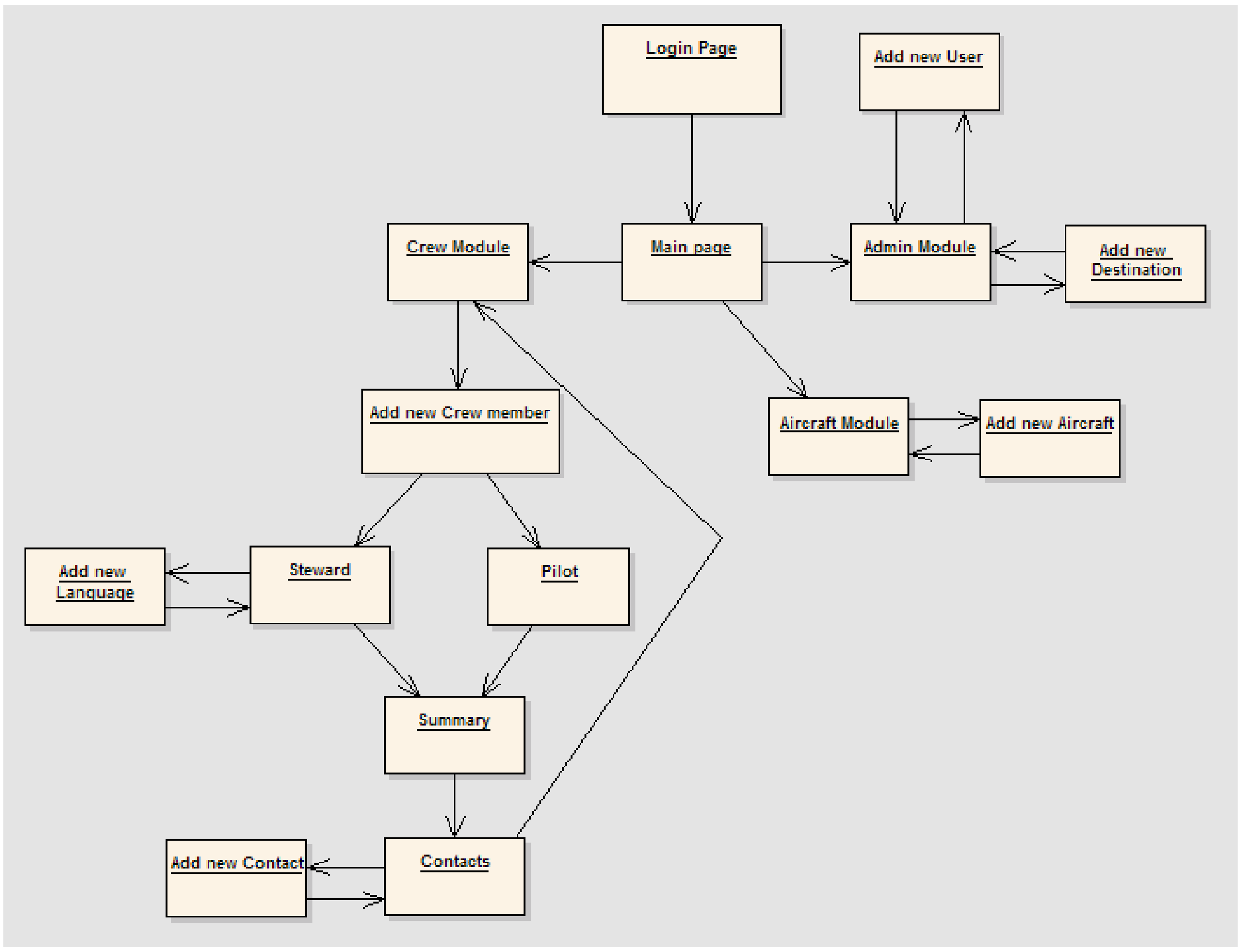


Figure 4.7 Modified page flow of the case study

5. Case study implementation

This chapter covers details about the implementation of the case study in Struts 2 and Tapestry and the integration of Struts 2 with WebFlow and Tapestry integration with WebFlow. All information about the progress and problems with implementation are mentioned here.

5.1. Struts 2

5.1.1. Setup

Setting up a new Struts 2 application was painless and straightforward. On the Struts 2 download page [21] are presented all of the required and optional libraries. Struts 2 community even provides a distribution with example blank application, which can be deployed and launched as provided.

The only minor problem was setting up Spring as the IoC container for Struts 2. After adding logging configuration for Log4J library, Spring informed about the missing configuration in web.xml and misplaced Spring configuration file. When the corrections were done, the application ran without problems.

5.1.2. Implementation

There were only few minor problems during the implementation phase. Spring was used as the IoC container for Struts 2 and the Struts 2 action classes were initialized by Spring. The only problem in configuration was that Spring initializes its beans by default as Singletons. Problems with user input validation occurred until this bug was fixed. Springs default behavior had to be overridden by setting singleton attribute in bean elements to false.

Other problems involved proper setup of interceptors. The documentation was very resourceful on this subject, so the issues were quickly fixed.

The last problem was with validation of field with value of type double. There were problems with setting customized validation error messages to display to the user (there were no problems with any of the other types of validations).

Overall, the implementation problems were a minor ones and did not cause larger delay in the implementation phase.

5.2. Tapestry

5.2.1. Setup

The configuration of Tapestry was more complicated than Struts 2. In Tapestry, only the core libraries are provided and the dependencies like Javassist and OGNL have to be downloaded separately and the project structure needs to be created from scratch.

There were some problems configuring HiveMind, where the application would not launch due to problems with HiveMind configuration files, although the files were placed on the required place in the project.

One of the main problems was configuration of services in HiveMind, so they could be injected to the Page and Component classes. There was a lack of documentation on the integration of Tapestry and HiveMind, therefore it took some time to configure it properly.

5.2.2. Implementation

There was only one major problem during the implementation phase. It was the validation of user input. Validation was implemented on the login page for the first time without any problem. Another attempt at implementing validation on other pages brought unpleasant behavior. Although the validation configuration followed the same pattern, only the validation on the login page worked as expected. The validations on the other pages did not work the way they were configured to and after their failure the login page validation failed as well. It took a server restart for the login validation to start working properly again.

Besides problems with validation, there was only one small problem with the change of locale for the application. Tapestry is unable to change the locale for currently selected page. The page needs to forward to different page before the change of locale is reflected in the application. There exist one workaround for this problem. Page needs to be redirect to a temporary page which redirects back to the original page before rendering its output.

5.3. Struts 2 with WebFlow

The base for this integration was taken from the application written in Struts 2. All that was needed was the addition of the Spring WebFlow libraries and the Spring WebFlow plug-in library for Struts 2. Additional configuration of WebFlow in Springs configuration file was required as was mentioned in chapter 2.3.3 along with the definition of special Spring bean called struts2FlowAdapter.

The plug-in provides very limited support for the possibilities WebFlow offers. There has to be defined a new Struts 2 action for every flow. The action must define a flowId parameter, which identifies the flow (it must be the same as the name of the file where is the flow defined).

This example shows flow action configuration in Struts 2:

```
<action name="AircraftFlow" class="com.googlecode.struts2webflow.FlowAction">
        <interceptor-ref name="myStack" />
                <param name="flowId">aircraft-flow</param>
                <result name="aircraftListAction" type="redirect">Aircraft_input.action</result>
                <result name="aircraftEditJSP">/pages/AircraftEdit.jsp</result>
</action>
```

The actions must define the FlowAction in the class attribute. The flowId parameter identifies the flow. The results have a special meaning for the view states (this will be described later in this chapter).

The action states are the only states that may execute some business logic. They are limited to execute one action and return a result. No support for entry or exit actions is provided.

This example shows action state configuration in Struts 2:

```
<action-state id="Aircraft_edit">
        <action bean="struts2FlowAdapter" />
        <transition on="edit" to="aircraftEdit" />
</action-state>
```

The id of the action state must match a name of an action in struts.xml. The action is invoked upon entering the action state and the actions result is mapped to the transition with corresponding name. All other types of actions or exception handling mechanism WebFlow provides support only Spring beans and are unusable in a Struts 2 application.

View state configuration in Struts 2 can be found in this example:

```
<view-state id="aircraftList" view="aircraftListAction">
        <transition on="edit" to="Aircraft_edit" />
        <transition on="new" to="Aircraft_addNew" />
        <transition on="delete" to="Aircraft_delete" />
```

```
</view-state>
```

The id has no special meaning for the view states. When a view state is executed, the value of the view attribute is matched against result names in the associated Struts 2 flow action and the matching result is executed. As in the action states, no support for the entry or exit actions is provided.

The plug-in uses an interceptor to store the flow execution key in the session object on the server. This works very well until the user decides to hit the browsers back button (the application will not update its state) and resubmit the form or click another link. The state of the application on the server and on the client will not be synchronized, what will cause an application error.

The solution proposed by the author of the plug-in involves leaving the interceptor out of the action configuration and use the flowExecutionKey directly like in any other WebFlow application, but the key does not update properly without the interceptor being used. This renders the plug-in unusable and as the result this application was not fully implemented. Moreover, when the Spring WebFlow is used with the Spring MVC, the original action configuration for the Spring MVC is not needed, so the size of the configuration does not increase much as it does when the WebFlow is used in integration with Struts 2. The original action configuration must be kept, new actions for flows must be defined and the flows themselves must be defined. The resulting configuration is more than doubled, what makes the orientation in the configuration much harder.

5.4. Tapestry with WebFlow

When the first official release of Spring WebFlow was released, the authors wanted to support most of the popular frameworks that were available including the Tapestry framework. As of today, no working solution for the integration of Tapestry with WebFlow exists. The only support which can be used to a higher degree is provided for Struts and JSF. Therefore this integration could not be implemented.

6. Results

In this chapter, the comparison results are presented. The results are organized in chapters, which correspond with criteria defined in chapter 3. The comparison has been done on Struts 2 and Tapestry frameworks. The integrations with WebFlow are not included because the implementations were not feasible (see previous chapter).

My previous experience with the frameworks should be mentioned here so it may be possible to obtain the information about frameworks learning curves from the results.

I have had about 9 months experience with Struts and 2 months of experience with Tapestry. Struts 2 is similar to Struts in many ways, what made learning Struts 2 easier, but I had no previous experience with Struts 2.

The pages have been implemented in the order they are shown in the tables so the increase of productivity is plainly visible by the reducing amount of time needed to implement successive pages.

6.1. Benchmarks

6.1.1. Time spent developing pages

The approximate time spent developing pages has been measured on a per page basis. Results are divided into 3 groups: time spent developing, time spent debugging and the sum of previous two. Time was measured in minutes.

	Development time	*Debugging time*	*Total time*	*Comment*
Login	50	30	80	problems with configuring Struts 2 actions in Spring to validate correctly
Main	40	0	40	20 minutes – creating menu, 20 minutes – change of locale
Planning	40	0	40	
Aircraft list	35	0	35	
Aircraft edit	30	20	50	problems with validations and interceptor configuration
Administration list	30	0	30	20 minutes – user list, 10 minutes filtering form
Administration edit	30	5	35	
Crew list	10	0	10	
Crew edit	25	0	25	problems with customizing error message for validation of field with type double
Pilot	25	0	25	
Contacts list	20	0	20	
Contacts edit	10	0	10	
Steward	10	0	10	
Languages list	10	5	15	
Languages edit	10	0	10	
SUM	**375**	**60**	**435**	

Figure 6.1 Time spent developing pages – Struts 2

	Development time	*Debugging time*	*Total time*	*Comment*
Login	80	0	80	
Main	105	0	105	45 minutes – menu component, 60 minutes – locale change component
Planning	60	30	90	
Aircraft list	25	5	30	
Aircraft edit	25	0	25	
Administration list	25	0	25	15 minutes – filtering form
Administration edit	60	0	60	
Crew list	15	5	20	
Crew edit	60	15	75	
Pilot	50	5	55	5 minutes – propertySelection for dropdown list
Contacts list	25	20	45	
Contacts edit	50	0	50	
Steward	10	0	10	
Languages list	30	0	30	
Languages edit	10	0	10	
SUM	**630**	**80**	**710**	

Figure 6.2 Time spent developing pages – Tapestry

The approximate average times spent developing 1 page (given the circumstances such as my previous experience):

Struts 2 – **29 minutes,** Tapestry – **47 minutes**.

6.1.2. Time spent modifying existing application

Four modifications to the existing application have been done: removing of Planning module, adding Destination management to the Administration module, joining Steward page with Language list page and adding Summary page before Contact list in Crew module.

The results show the approximate time needed to accomplish each modification and the number of files changed for each modification. Time was measured in minutes.

	Time needed	*Number of files changed*
Removing Planning module	5	10
Adding Destination management	30	10
Joining Steward and Language list page	15	11
Adding Summary page	15	7
SUM	65	38

Figure 6.3 Time spent modifying existing application – Struts 2

	Time needed	*Number of files changed*
Removing Planning module	3	9
Adding Destination management	22	11
Joining Steward and Language list page	8	14
Adding Summary page	13	10
SUM	46	44

Figure 6.4 Time spent modifying existing application – Tapestry

The approximate average time needed for one modification in Struts 2: **16 minutes**.

The approximate average time needed for one modification in Tapestry: **11 minutes**.

6.1.3. The size of written code

The size of written code is divided into size of configuration files, size of class file and size of JSP/HTML template. The results are shown in bytes for each page separately as well as an average for all of the pages.

The Struts 2 results do not show the size of configuration files (there is only one struts.xml in whole application). Instead, they show the size of validation files (where the validation files are present).

	Validation	*Class*	*JSP*	*SUM*
Login	591	1427	816	**2834**
Main	0	352	553	**905**
Administration list	0	2555	1978	**4533**
Administration edit	1984	2608	1632	**6224**
Aircraft list	0	3505	1483	**3235,5**
Aircraft edit	391		1130	**3273,5**
Planning	0	2304	1564	**3868**
Crew list	0	1745	1760	**3505**
Crew edit	2256	6242	1752	**10250**
Pilot	697	2779	1046	**4522**
Steward	444	1468	911	**2823**
Languages	0	3796	1621	**3519**
Languages edit	386		1129	**3413**
Contacts	0	4122	1612	**4213**
Contacts edit	385		1120	**4106**
Average	**475.6**	**2193,53**	**1340.47**	**4208.85**

Figure 6.5 The size of written code – Struts 2

	Configuration	*Class*	*HTML*	*SUM*
Login	549	1121	1246	**2916**
Main	500	577	131	**1208**
Administration list	673	1749	2871	**5293**
Administration edit	673	2049	3019	**5741**
Aircraft list	686	1550	932	**3168**
Aircraft edit	941	1733	896	**3570**
Planning	612	2036	1570	**4218**
Crew list	670	1452	1090	**3212**
Crew edit	747	4137	3372	**8256**
Pilot	684	1810	1189	**3683**
Steward	602	1203	749	**2554**
Languages	762	2005	1045	**3812**
Languages edit	685	1994	780	**3459**
Contacts	679	2067	1038	**3784**
Contacts edit	680	3254	1351	**5285**
Average	**676.2**	**1915.8**	**1418.6**	**4010.6**

Figure 6.6 The size of written code – Tapestry

The action based orientation of Struts 2 made it possible to implement only one action for more pages as can be seen in the table for Aircraft, Languages and Contacts pages.

The overall results between Struts 2 and Tapestry are almost equal giving Tapestry slight advantage.

6.1.4. Size of generated HTML code

The size of generated HTML code has been measured in bytes on a per page basis. Requests for pages have been sent to the server and the size of the received pages has been measured. The average size of 1 page is included as well.

	Struts 2	*Tapestry*
Login	1181	2640
Main	1353	2821
Administration list	4415	8450
Administration edit	5500	5442
Aircraft list	3407	5147
Aircraft edit	2142	3716
Planning	2021	3334
Crew list	2664	3886
Crew edit	5955	6004
Pilot	3295	4282
Steward	2407	3628
Languages list	1811	3044
Languages edit	2075	3562
Contacts list	1801	3027
Contacts edit	2061	3931
Average	**2805.87**	**4194.27**

Figure 6.7 Size of generated HTML code

The results of this measurement favor Struts 2. Tapestry includes additional meta information and JavaScript code in the generated HTML. The form handling mechanism also works differently and Tapestry stores a lot of information about form fields on the page. This results in a larger HTML code generated by Tapestry.

6.1.5. Frameworks performance

Two benchmarks for each of the frameworks have been done. One benchmark simulated 50 users requesting single page 100 times each (5000 requests) and the other simulated 100 users cycling through all of the pages 10 times (15000 requests together – 100 users * 15 pages * 10 cycles). The Crew Edit page has been chosen for this measurement, because it is the largest page in the case study.

The benchmarks have been done using JMeter benchmarking tool and the applications have both been deployed on the same Tomcat web server. The measurements have been done after a warm up phase of 5000 requests to eliminate the effects of JIT (Just In Time compiler) and the "warming up" of the frameworks.

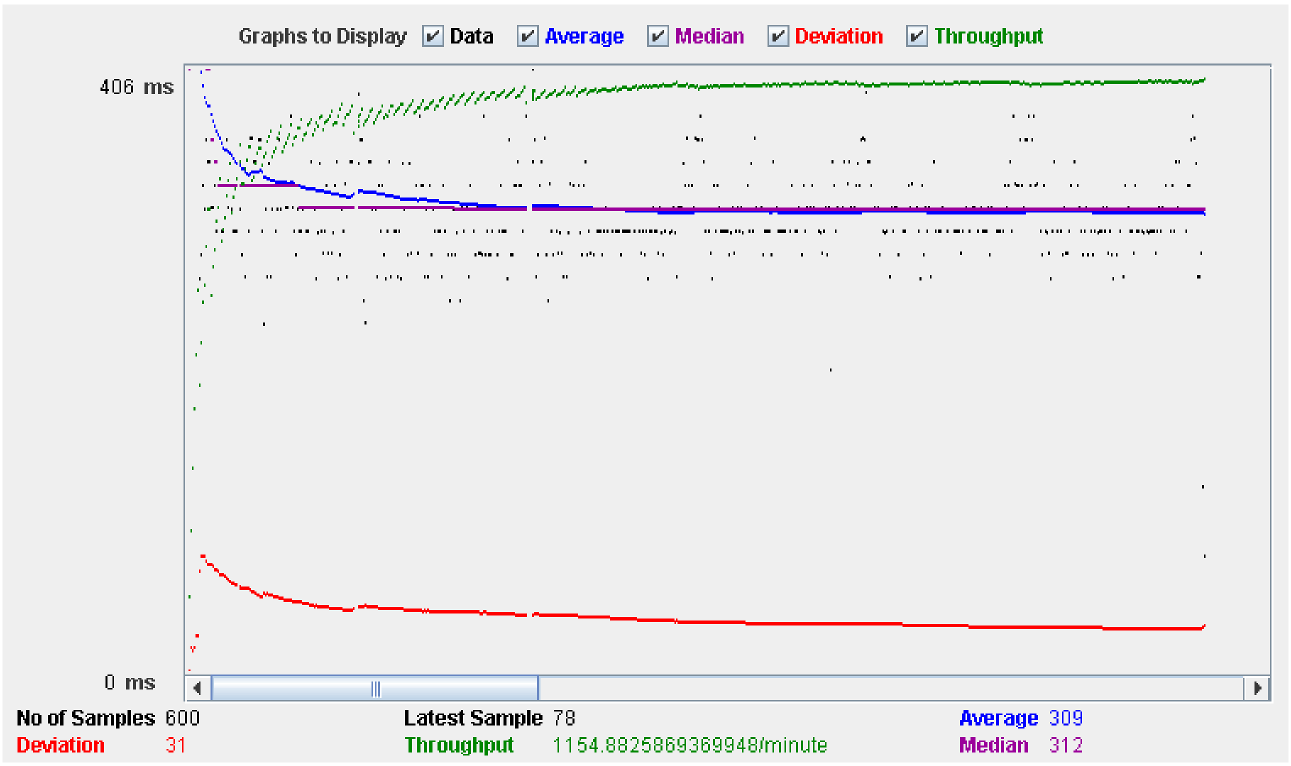
The benchmarks recorded response times (in milliseconds on the graphs and hundreds of microseconds in the tables), median and average times, standard deviation and throughput (number of pages processed per second/minute) for each request.

The graphs always show the results for first 600 requests, because JMeter can not correctly handle graphs with more than 2000 samples.

All results are shown in the color of their label. The X axis represents index of a sample.

Data values represent the times of each request, average values represent the average time for all previous samples (for example value for sample 10 shows the average for first 10 samples), the median and deviation are calculated the same way as the average.

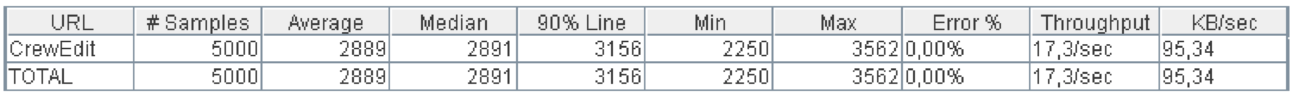
Throughput curve does not correspond with the Y axis. It shows the progress of throughput in time.



No of Samples 600
Deviation 31
Latest Sample 78
Throughput 1154.8825869369948/minute
Average 309
Median 312

Figure 6.8 Struts 2 – single page benchmark

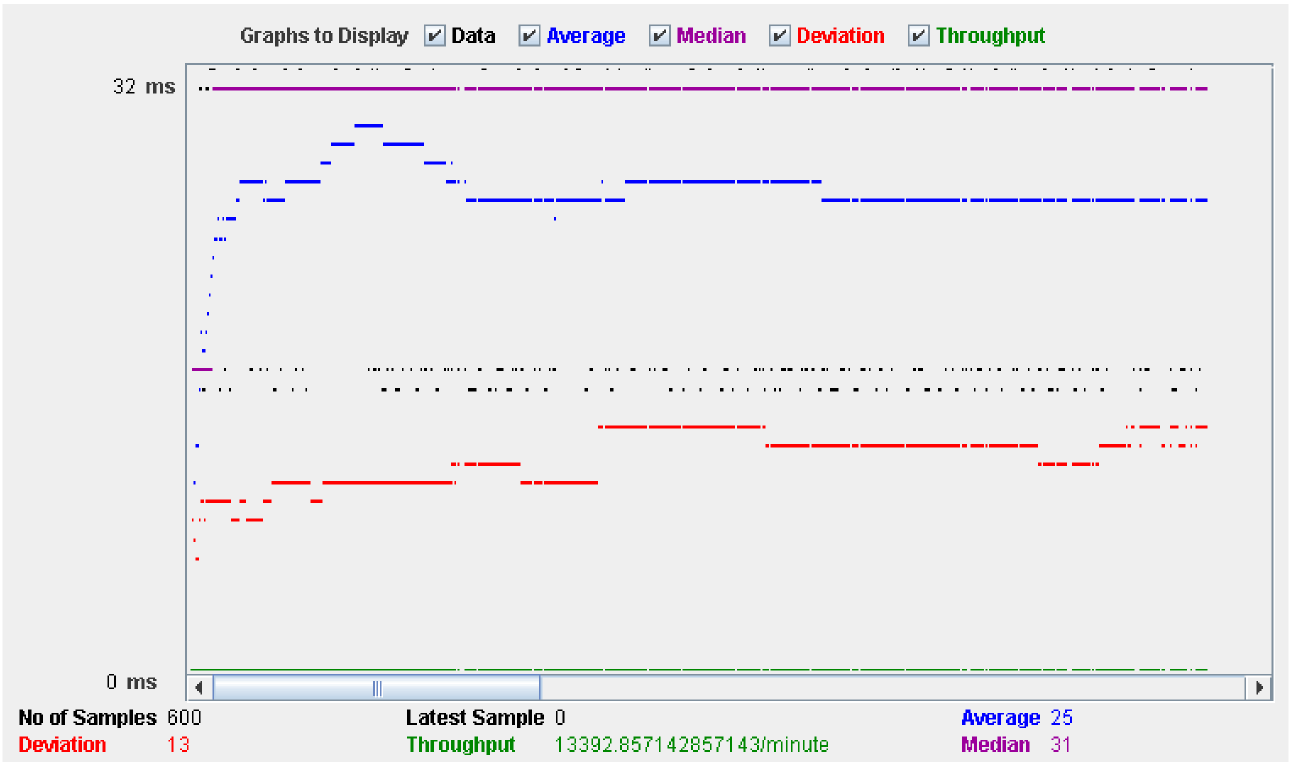
As can be seen on this graph, the average time lessens in time until it settles at the value of about 300ms, which probably indicates the peak value for this measurement.



URL	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
CrewEdit	5000	2889	2891	3156	2250	3562	0,00%	17,3/sec	95,34
TOTAL	5000	2889	2891	3156	2250	3562	0,00%	17,3/sec	95,34

Figure 6.9 Struts 2 – single page benchmark

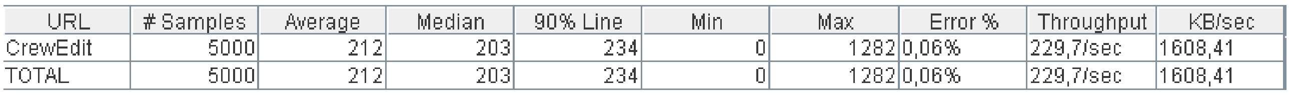
The final results for 5000 requests are a little lower at the approximate value of 290ms, which gives the final throughput of approximately 17 pages per second.



No of Samples 600
Latest Sample 0
Average 25
Deviation 13
Throughput 13392.857142857143/minute
Median 31

Figure 6.10 Tapestry single page benchmark

The results of Tapestry in this benchmark are approximately 10 times better than those of Struts 2. The average response time is 25ms coming after rise in the beginning and fall at the end. This may be caused by creation of page classes, which are pooled by Tapestry. This may cause the overhead in the beginning and good performance towards the end.

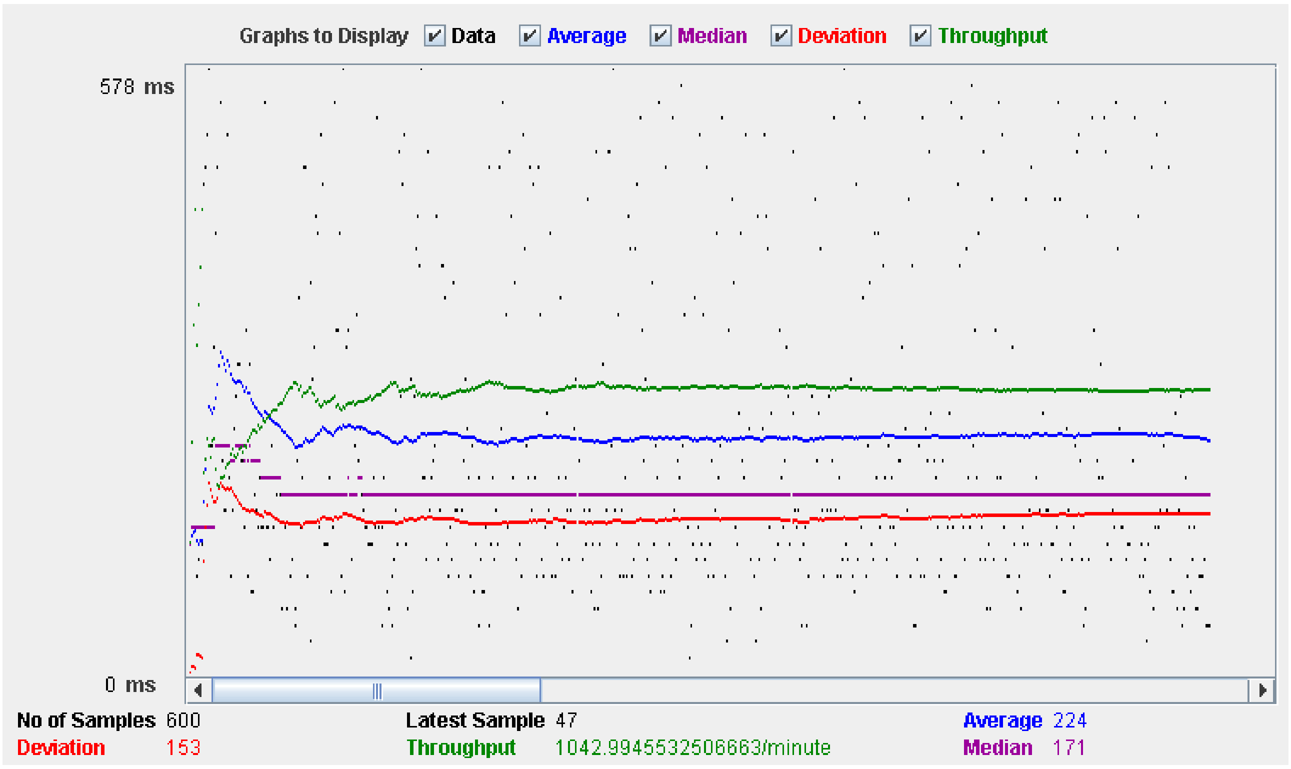


URL	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
CrewEdit	5000	212	203	234	0	1282	0,06%	229,7/sec	1608,41
TOTAL	5000	212	203	234	0	1282	0,06%	229,7/sec	1608,41

Figure 6.11 Tapestry – single page benchmark

The final results for Tapestry, as was the case for Struts 2, are still lower than results for 600 requests. The average time is about 21ms and the final throughput rounds up to almost 230 pages per second. The pooling strategy of Tapestry showed its better face in this benchmark.

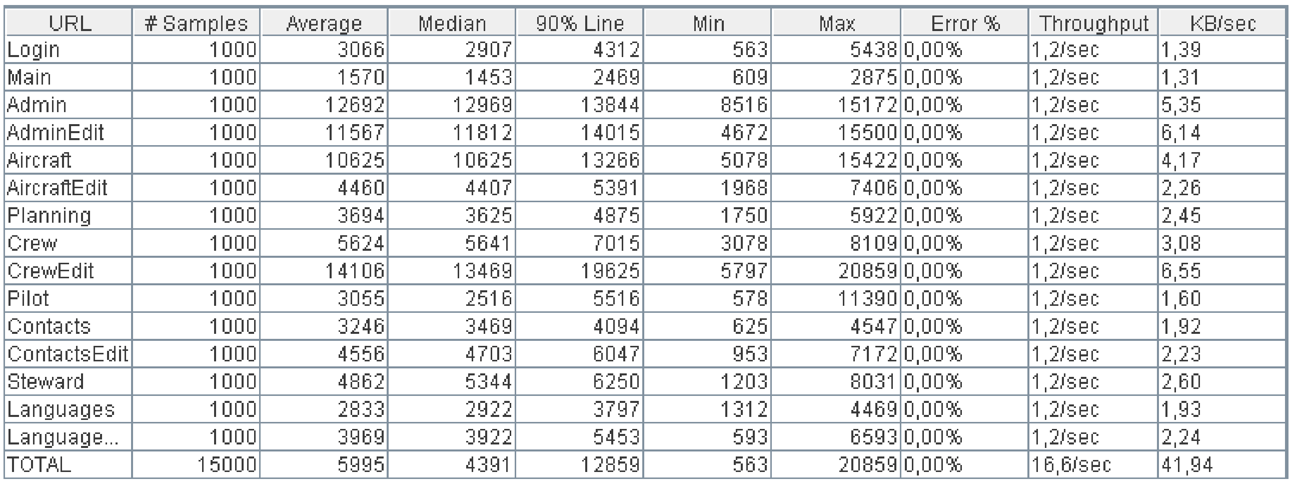
Now follow results of the second benchmark:



No of Samples 600
Deviation 153
Latest Sample 47
Throughput 1042.9945532506663/minute
Average 224
Median 171

Figure 6.12 Struts 2 cycle benchmark

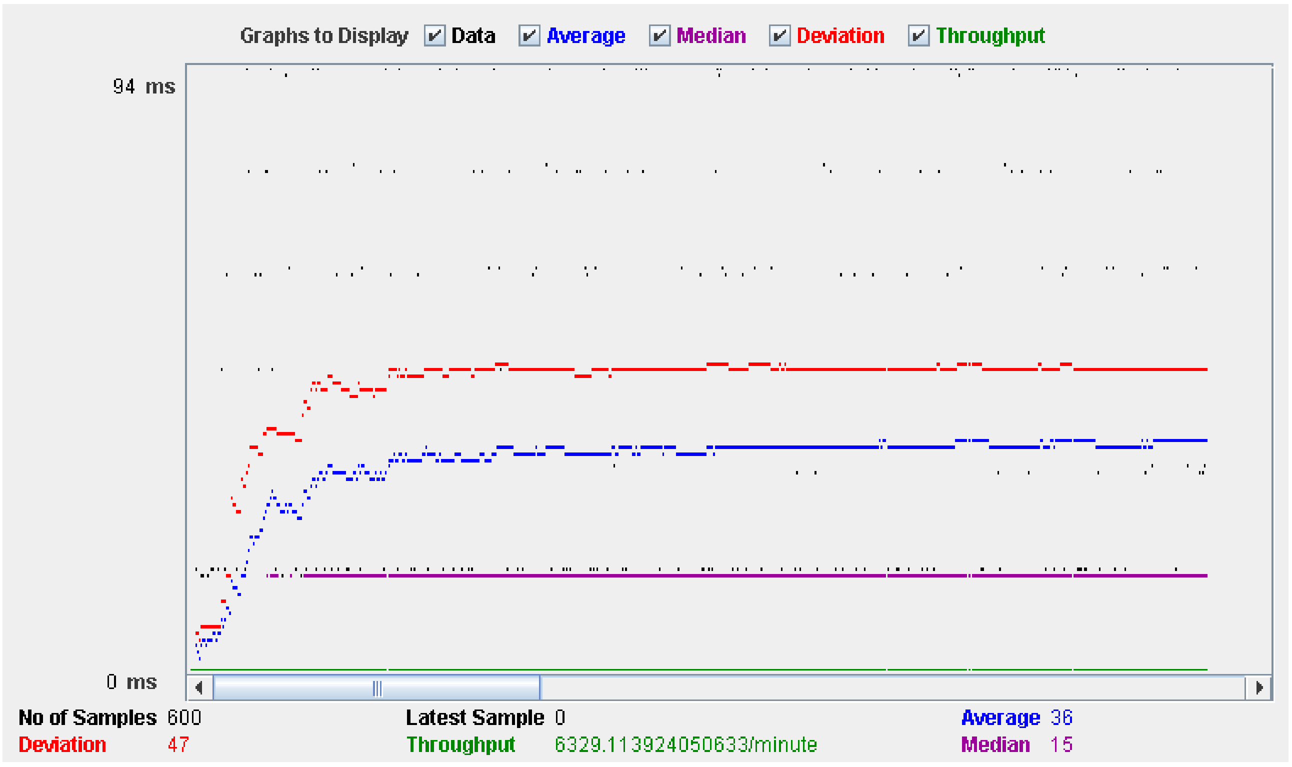
The average response times for first 600 requests show the average time of about 220ms with almost constant throughput after about first 200 requests.



URL	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
Login	1000	3066	2907	4312	563	5438	0,00%	1,2/sec	1,39
Main	1000	1570	1453	2469	609	2875	0,00%	1,2/sec	1,31
Admin	1000	12692	12969	13844	8516	15172	0,00%	1,2/sec	5,35
AdminEdit	1000	11567	11812	14015	4672	15500	0,00%	1,2/sec	6,14
Aircraft	1000	10625	10625	13266	5078	15422	0,00%	1,2/sec	4,17
AircraftEdit	1000	4460	4407	5391	1968	7406	0,00%	1,2/sec	2,26
Planning	1000	3694	3625	4875	1750	5922	0,00%	1,2/sec	2,45
Crew	1000	5624	5641	7015	3078	8109	0,00%	1,2/sec	3,08
CrewEdit	1000	14106	13469	19625	5797	20859	0,00%	1,2/sec	6,55
Pilot	1000	3055	2516	5516	578	11390	0,00%	1,2/sec	1,60
Contacts	1000	3246	3469	4094	625	4547	0,00%	1,2/sec	1,92
ContactsEdit	1000	4556	4703	6047	953	7172	0,00%	1,2/sec	2,23
Steward	1000	4862	5344	6250	1203	8031	0,00%	1,2/sec	2,60
Languages	1000	2833	2922	3797	1312	4469	0,00%	1,2/sec	1,93
Language...	1000	3969	3922	5453	593	6593	0,00%	1,2/sec	2,24
TOTAL	15000	5995	4391	12859	563	20859	0,00%	16,6/sec	41,94

Figure 6.13 Struts 2 cycle benchmark

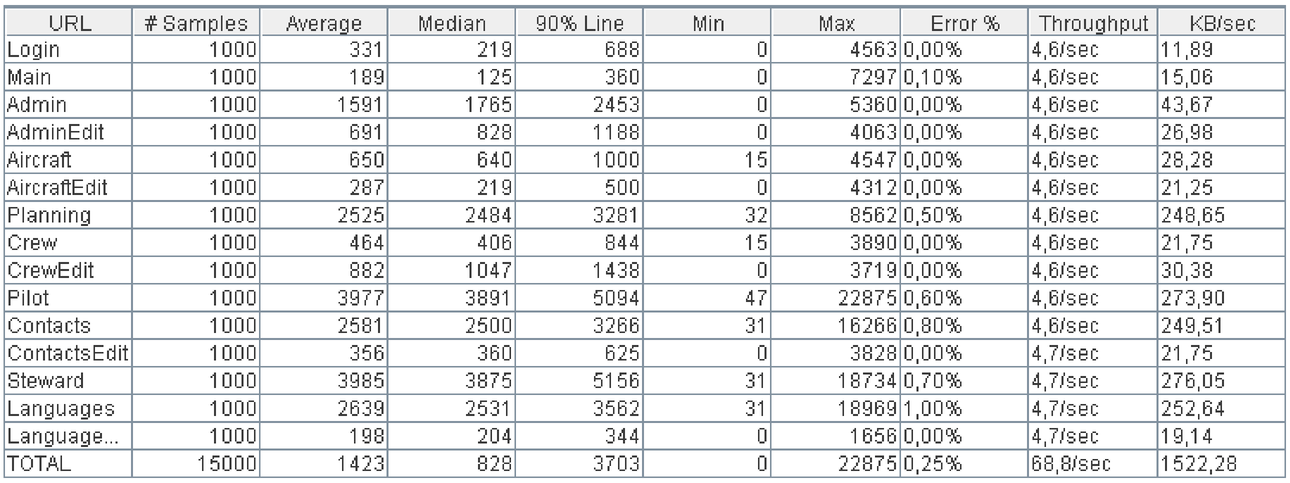
The average time for all 15000 requests comes up to 600ms. The higher values were expected due to variation of pages. The approximate final throughput is almost 17 pages per second.



No of Samples 600 Latest Sample 0 Average 36
Deviation 47 Throughput 6329.113924050633/minute Median 15

Figure 6.14 Tapestry cycle benchmark

The average times for first 600 requests are about 36ms. The throughput curve is constant at the bottom of the graph.



URL	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
Login	1000	331	219	688	0	4563	0,00%	4,6/sec	11,89
Main	1000	189	125	360	0	7297	0,10%	4,6/sec	15,06
Admin	1000	1591	1765	2453	0	5360	0,00%	4,6/sec	43,67
AdminEdit	1000	691	828	1188	0	4063	0,00%	4,6/sec	26,98
Aircraft	1000	650	640	1000	15	4547	0,00%	4,6/sec	28,28
AircraftEdit	1000	287	219	500	0	4312	0,00%	4,6/sec	21,25
Planning	1000	2525	2484	3281	32	8562	0,50%	4,6/sec	248,65
Crew	1000	464	406	844	15	3890	0,00%	4,6/sec	21,75
CrewEdit	1000	882	1047	1438	0	3719	0,00%	4,6/sec	30,38
Pilot	1000	3977	3891	5094	47	22875	0,60%	4,6/sec	273,90
Contacts	1000	2581	2500	3266	31	16266	0,80%	4,6/sec	249,51
ContactsEdit	1000	356	360	625	0	3828	0,00%	4,7/sec	21,75
Steward	1000	3985	3875	5156	31	18734	0,70%	4,7/sec	276,05
Languages	1000	2639	2531	3562	31	18969	1,00%	4,7/sec	252,64
Language...	1000	198	204	344	0	1656	0,00%	4,7/sec	19,14
TOTAL	15000	1423	828	3703	0	22875	0,25%	68,8/sec	1522,28

Figure 6.15 Tapestry cycle benchmark

Tapestry outperformed Struts 2 again. The average time of about 1400ms is more than 4 times better than that of Struts 2, although the difference is not as significant as in the single page benchmark. The throughput values are also approximately 4 times better.

The caching system of Tapestry showed its strength in this benchmark. The overwhelming result came at the cost of higher memory consumption. The allocatable heap memory for Tomcat had to be increased for this benchmark to run successfully, otherwise Tomcat threw OutOfMemory exceptions.

Overall, Tapestry proved to be the more efficient framework in this benchmark.

6.1.6. Typical amount of files per 1 page

Configuration of one action in Struts 2 requires a record in struts.xml, action class, localization properties file (one per each supported language), validation XML file (if page has any form, which requires validation) and JSP page. Overall, it takes about 5 files per one action (two localization properties files were used in this case study. Struts.xml is common for all actions). Actions in these examples were grouped in classes to map one class to one page.

Several other configuration files, base classes and actions have to be accounted for. Total of 69 files (domain model classes are not included) were required to implement 15 pages. The approximate result is **4,5 files per one page.**

Tapestry page configuration requires a record in app.application file, page class, page configuration file, localization properties files and an HTML template. This sums up to 5 files (again two properties files and app.application file is one in whole application) per one page.

Total sum of files needed to implement 15 pages, including all other configuration files and files needed to implement reusable components, is 96 files. The approximate result is **6,5 files per one page.**

This criterion shows that the configuration of Tapestry is little more complicated requiring almost 2 files more than Struts 2 to implement one page.

6.1.7. Page correspondence with W3C standards

Two standards have been chosen for this comparison. HTML 4.01 Transitional and XHTML 1.0 Transitional. The pages were generated and uploaded to W3C validator to check the correspondence to both of the standards. There were done no modification to the pages between measurements. The results show number of errors on each page according to the definition of the standard.

	HTML	*XHTML*
Login	0	3
Main	0	2
Administration list	0	3
Administration edit	0	3
Aircraft list	0	2
Aircraft edit	0	3
Planning	0	3
Crew list	0	2
Crew edit	0	3
Pilot	0	3
Steward	0	3
Languages	0	2
Languages edit	0	3
Contacts	0	2
Contacts edit	0	3
Average	0	2.67

Figure 6.16 Page correspondence with W3C standards – Struts 2

All Struts 2 pages were completely error free according to the HTML 4.01 Transitional standard. XHTML standard generated errors due to unclosed meta and link tags (closed tags generate errors in HTML standard) and with uppercase letters in the form method type (the only error that is not possible to fix because it is rendered by the Struts 2 form tag).

	HTML	*XHTML*
Login	6	5
Main	12	11
Administration list	12	11
Administration edit	13	12
Aircraft list	22	21
Aircraft edit	13	12
Planning	8	7
Crew list	16	15
Crew edit	13	12
Pilot	12	11
Steward	12	11
Languages	12	11
Languages edit	13	12
Contacts	12	11
Contacts edit	13	12
Average	12.6	11.6

Figure 6.17 Page correspondence with W3C standards – Tapestry

Tapestry generated a lot more errors than Struts 2. This is caused mostly by the extra meta information and JavaScript inserted by the framework. Also in Struts 2, the developer has more control over the resulting HTML code. Tapestry components are on a higher level and do not provide that much control over the code generation.

The overall results favor Struts 2 with approximately 0 : 12 errors for HTML and 3 : 12 errors for XHTML standard.

6.2. Features

This chapter covers several features like localization, validation and AJAX support. This criteria could not be precisely measured, therefore the results are yes or no answers with closer description of the level of support.

6.2.1. Localization support

Both of the frameworks support localization in a native way. The difference is in the quality of support.

Struts 2 defines a hierarchy of localization files. Localization messages can be defined for each action, base class, interface and package separately. Furthermore, it allows definition of extra resource bundles, which can be accessed through special tags and global resource bundle may be defined as well in the main configuration file. The localized message is searched for through a hierarchy of bundles (precise hierarchy is defined in the Struts 2 documentation) and the closest match is returned.

Changing of the locale is a matter of a definition of a special URL tag. Overall, the localization system is sufficient for most of the projects.

Tapestry allows to define localization file for each component, each class and a global resource bundle. The shortage of possibilities is balanced by the possibility to define a localized version of the page itself (for example, countries where people read from right to left may have different HTML template).

Change of the locale is more complicated in Tapestry. The change itself is simple – the only thing needed is the change of the locale for the page in the page listener. However, the change of the locale does not apply to the currently loaded/processed page. Therefore, in order to instantly change the locale for the current page, the application must redirect to a different page and redirect back again. This issue is solvable by defining a new page, which redirects back to original page right after it loads itself, although it is not very clean solution.

6.2.2. Validation support

Struts 2 and Tapestry provide means for user input validation. Both of them allow the definition of input field validators and the definition of more complex validation, which may include more fields or complex expressions.

Both frameworks provide standard validators like required, integer, double, date, email, url, etc. More complex validations may be defined through the regexp (regular expressions) validator. User defined validators may be defined as well.

The major difference is the place of the definition of the validation. Tapestry allows to define validations on the HTML template or in the page configuration file. Struts 2 has only its validation XML files.

Both frameworks support client side validations as well.

The validation support from both frameworks is on a satisfiable level. All common validators are provided and extra validators may be defined by the user.

6.2.3. AJAX support

AJAX is a relatively new technology, which makes dynamic web pages more interactive. Both frameworks support AJAX by providing special tags or components. DOJO toolkit is supported by both frameworks. Struts 2 supports Google Web Toolkit through a plug-in as well.

No extra configuration is needed in any of the frameworks for AJAX to work properly. By using special tags in Struts 2 or components in Tapestry, users are given direct access to AJAX features.

6.3. Subjective judgement

This chapter covers personal opinion about the frameworks in several areas.

6.3.1. Technology complexity

The size of the written code for both frameworks is almost equal. The size of the code written in Tapestry is actually smaller, although the development time has been more than 150% of the development time in Struts 2.

I have had a lot of experience in working with Struts, which is similar to Struts 2 in many ways and I have had about two months of experience working with Tapestry. Even thought I had no experience working with Struts 2, I still found Struts 2 as easier to learn and comprehend. It may be a better choice for programmers who have experience with Struts and want to use a new framework with new features and support for new technologies, but who do not have time (or simply do not want) to learn a completely different framework.

6.3.2. Complexity of orientation in existing project

Although Tapestry requires more configuration files to configure one page, the configuration is spread in three places, the same as in Struts 2. The overall time spent modifying the application came up better for Tapestry, which may indicate better productivity once the framework is learned.

Overall, both frameworks are well designed and with good project structure. The orientation in existing project should not pose big problems.

6.3.3. Complexity of setting up a new project

Struts 2 came out better in this comparison as the community provides all dependencies in one place along with a sample blank application, which is ready for deployment and modification. The setup phase took 40 minutes.

Tapestry does not provide all dependencies by itself. Third party libraries have to be downloaded separately and the web project structure has to be created from scratch. There were minor problems configuring HiveMind and the overall setup time was 90 minutes.

6.3.4. Quality of support

The weakest place in both frameworks is the quality of the documentation. The documentation is incomplete, contains few examples and is badly organized. It does not cover all the functionality and the information has to be looked up in other sources on the Internet.

The troubleshooting support is on a lot better level. All questions regarding problems encountered by users are answered promptly by some of the developers involved in the frameworks.

7. Related work

This chapter discusses other studies that deal with Java web frameworks and their results.

There are several comparisons to be found on the Internet. Most of them cover more than two frameworks (Struts 2 is included in only one of the studies, although more deal with WebWork, the Struts 2 predecessor), but none of them compares data acquired from serious measurements.

Studies *Comparing Web Frameworks*([15] and [16]) done by Matt Raible are popular, but they are based just on a personal experience of the author. No case study has been implemented and the results reflect authors personal opinion on the usage of selected frameworks.

The studies describe several frameworks with examples of source code. The author defines comparison criteria, some of which are the same as in this thesis like internationalization, validation and AJAX support. The author included some reasonable criteria like testability and the number of tools, which are available to ease the development process, however he did include several criteria, which are not professional enough and change in time like the number of jobs available for programmers with knowledge of certain framework.

Overall, the study did not include any measurable criteria and the results are based solely on the personal opinion of the author, therefore it cannot be compared to this thesis.

The study *Comparison of Java Web Frameworks* [17] covers 6 different Java web frameworks. The study starts with description of MVC design pattern and continues with description of all of the frameworks.

The architecture and main feature of each framework are shortly described and discussed. No examples aid the descriptions, no criteria for comparison are defined.

The author provides simple evaluation at the end of the study, however the evaluation is short and it does not compare the frameworks from any point of view, which really matters to most of application developers. The evaluation criteria are the following: transparent infrastructure, innovative ideas and high cohesion and low coupling. These criteria do not tell anything about the development process or the performance of the frameworks, therefore this study does not give much useful information.

The work *Web Framework Comparison* [18] starts with the definition of comparison criteria. Some, such as community support, internationalization and performance, are the same as criteria defined in this study, other are completely different, but reasonable enough to include in serious study.

The criteria definition is followed by the evaluation of each framework according to the criteria. The evaluation is based on a personal opinion without any real measurements being done.

The summary gives the result in form of one chosen framework without much justification of the selection.

The only study, where the comparison has been done on a real implementation of some case study is *Comparing webapp frameworks* [19]. The case study consists of 3 pages and contains very simple domain model.

The study is implemented in a large number of frameworks/technologies, but to be able to accomplish this, the author chose very simple case study, which is incapable of showing many features or capabilities of each framework.

The study provides examples of code for each of the implementations, but it does not define any real criteria for comparison. It just gives an overview of the development process in each framework but does not provide any evaluation that could be of any use to the developers.

8. Conclusion

The main goal of this thesis has been to provide information about several Java web frameworks for application developers and architects, when they are trying to choose suitable framework for their projects.

The sub-goals of this thesis have been to define criteria for frameworks comparison, define a case study, implement the case study in selected frameworks and measure and evaluate the results based on the defined criteria.

The goals of this thesis have been fulfilled. The comparison criteria were defined and described in detail.

The basic case study along with its modifications has been defined and implemented in both Struts 2 and Tapestry frameworks.

The research on the possibility of integration of Spring WebFlow with Struts 2 and Tapestry has been done. The result of this research showed that the integration support for Struts 2 is not in a functional state and the support of Tapestry has not been implemented yet.

The measurements have been successfully accomplished and showed the strengths and weaknesses of both frameworks.

The final recommendations are:

- both frameworks are capable of providing services, which most of current projects demand.
- both frameworks are suitable for projects from small to large scale.
- Struts 2 may be the better choice for developers with the knowledge of Struts as their design is very similar. Therefore, Struts 2 is easy to learn with the knowledge of Struts.
- Tapestry may be better for larger projects where the pages are divisible into highly reusable components, where it may save a lot of development time.
- Tapestry showed a lot better performance so it may be the better choice if one of the most important criteria is the performance.

9. References

[1] Struts 2 homepage. http://struts.apache.org/2.x/

[2] Using Struts 2, Matt Raible, 2007. http://appfuse.org/display/APF/Using%20Struts%202

[3] Struts 2 Architecture, Patrick Lightbody, 2007, http://struts.apache.org/2.x/docs/big-picture.html

[4] Struts 2 features. http://www.roseindia.net/struts/struts2/struts-2-features.shtml

[5] MVC pattern, Sang Shin, 2007. http://www.javapassion.com/j2ee/MVCPatternAndFrameworks.pdf

[6] Wikipedia. http://en.wikipedia.org/wiki/

[7] Tapestry homepage. http://tapestry.apache.org/

[8] Tapestry 4.1 homepage. http://tapestry.apache.org/tapestry4.1/

[9] Tapestry users guide. http://tapestry.apache.org/tapestry4.1/usersguide/index.html

[10] Introduction to Tapestry, Neal Ford , 2006. http://www.nealford.com/downloads/conferences/2006_nfjs_canonical/Neal_Ford-Introduction_to_Tapestry-slides.pdf

[11] Tapestry 101, Warner Onstine, 2006. http://sourcebeat.com/titles/tapestrylive/public/Rev_1/TapestryLive_SampleChapter.pdf

[12] Spring WebFlow flow definition. http://static.springframework.org/spring-webflow/docs/current/reference/flow-definition.html

[13] Spring WebFlow – A practical introduction, Erwin Vervaet, 2007. http://www.ervacon.com/products/swf/intro/index.html

[14] Spring WebFlow homepage. http://www.springframework.org/webflow

[15] Comparing Web Frameworks, Matt Raible, 2006. https://equinox.dev.java.net/framework-comparison/WebFrameworks.pdf

[16] Comparing Web Frameworks, Matt Raible, 2007. http://static.raibledesigns.com/repository/presentations/ComparingJavaWebFrameworks.pdf

[17] Comparison of Java Web Frameworks, Neal Ford. http://bdn1.borland.com/article/borcon/files/6000/paper/6000.html

[18] Web Framework Comparison, Pieter Hartsook, 2006. http://chandlerproject.org/bin/view/Projects/WebFrameworkComparison

[19] Comparing webapp frameworks, Simon Brown, 2005. http://weblogs.java.net/blog/simongbrown/archive/2005/11/comparing_webap.html

[20] W3C validator, http://validator.w3.org/

[21] Struts 2 download page, http://struts.apache.org/download.cgi

[22] JMeter homepage, http://jakarta.apache.org/jmeter/