

Evaluation of Future JEE Technologies

Nicolas Lanquetin
0604918@abertay.ac.uk



UNIVERSITY
of
ABERTAY DUNDEE

University of Abertay Dundee
School of Computing & Creative Technologies

May 2007

University of Abertay Dundee

Permission to copy

Author: Nicolas Lanquetin

Title: Evaluation of Future JEE Technologies

Degree: BSc (Hons) Web Design & Development

Year: 4

- i) I certify that the above mentioned project is my original work.
- ii) I agree that this dissertation may be reproduced, stored or transmitted, in any form and by any means without the written consent of the undersigned.

Signature:

Date:

Abstract

The internet is an ideal medium for enterprises to strengthen their brand and communicate their services. Many enterprises also require an intranet web application for internal processes or simply to increase communication inside the company. For large-scale enterprise web applications, a strong environment such as JEE is required. However, in order to develop such a web application, a good solution is to use a framework. The latter offers a bundle of common functionality and will, therefore, act as the foundation for a new web application. Many frameworks exist, but a recent one in particular stands out: Apache Struts 2.0. Together with a project lifecycle management tool, Maven 2, this technology make professional and modern web development possible. This research will test a variety of features that Maven 2 and Struts 2.0 offer and evaluate whether they meet specific criteria required for successful web development. The results will clarify if it is worth using these tools in production environment and for which target group they will be of interest.

Contents

1. Introduction	1
2. The Technologies	3
3. Evaluation of the Technologies	5
3.1. Maven 2	5
3.1.1. The Default Folder Hierarchy	6
3.1.2. The Project Object Model	7
3.1.3. Goals in Maven 2	8
3.1.4. The WAR File	9
3.1.5. Summary of Maven’s Evaluation	10
3.2. Struts 2.0	11
3.2.1. The Framework Architecture	11
3.2.2. Data Binding	16
3.2.3. Validation	17
3.2.4. Double Submits	20
3.2.5. Internationalisation (i18n)	21
3.2.6. Struts 2.0 Tag Library	22
3.2.7. Testing Framework	23
3.2.8. Support & Resources	24
3.2.9. Tool Support	27
3.2.10. Performance	28
3.2.11. Flexibility & Extensibility	33
3.2.12. Revolutionising Features	34
3.2.13. Summary of Struts’ Evaluation	35
4. Conclusion	36
A. The JMeter Test Plan	38

Contents

Glossary	42
Bibliography	44
Index	48

List of Figures

2.1. Deployment of a web application using Maven 2	4
2.2. The different layers of used technologies	4
3.1. Folder Structure of the 'hp' project	6
3.2. Folder Structure of the exploded hp.war file	9
3.3. Diagram of the Model-View-Controller pattern	11
3.4. Interceptor Architecture	13
3.5. UML class diagram of the Person JavaBean	16
3.6. Screenshot of the rendered Login form	22
3.7. Messages posted in the mailing lists from January to April 2007	26
3.8. Proportion of Struts 1.x and Struts 2.0.x threads in the Struts mailing list	26
3.9. Number of messages posted monthly to the mailing lists	26
3.10. Single user load-test: Aggregate Graph and Graph Data	30
3.11. Multi-user load-test: Aggregate Graph and Graph Results	32
3.12. Multi-user test destructively: Graph Results	33
A.1. Screenshot of the Welcome screen	38
A.2. Screenshot of Login screen	38
A.3. Screenshot of the main menu	38
A.4. Screenshot of the registration screen, showing the subscriptions	39
A.5. Screenshot of the screen to add a new subscription	39
A.6. Screenshot of registration screen with a new subscription	40
A.7. Screenshot of the screen to delete a subscription	40
A.8. Screenshot of a submitted form where validation failed.	41

List of Tables

- 3.1. Single user load-test: Summary report 29
- 3.2. Multi-user load-test: Summary report 31

Listings

3.1. pom.xml of the hp project	7
3.2. Execution of Maven's 'package' goal	8
3.3. An Action example: Login.java	12
3.4. Extract of the struts.xml	14
3.5. Accessing the personList Collection from JSP	15
3.6. Accessing JavaBeans from a JSP	16
3.7. Registration-validation.xml belonging to the Registration Action	18
3.8. The Registration Action with annotations	19
3.9. Accessing message resources in JSP	21
3.10. Login form demonstrating Struts 2.0 tags	22
3.11. LoginTest.Java	24
3.12. Continuations demonstrated by the GuessAction	34

Organisation

This dissertation is separated into four chapters, which introduce the research, present the technologies and their evaluation, and conclude the dissertation by summarising the major results.

Chapter 1, Introduction (p. 1), goes through the background of this research project and places the subject into its context.

Chapter 2, The Technologies (p. 3), gives an overview of the different technologies required to run a web application and explains how they work together.

Chapter 3, Evaluation of the Technologies (p. 5), evaluates the two most important technologies: Maven 2 and Struts 2.0.

Chapter 4, Conclusion (p. 36), summarises the major results found in this research, concludes if the technologies are of interest for web application development and proposes future solutions to better Struts 2.0.

Chapter 1.

Introduction

Since the internet has become accessible to everyone, having an internet presence is crucial for corporations, as they can represent their brand and services, and therefore do business. It is however required to have a web application to communicate with the internet users. Many ready to use applications are already available, often for free. The latter are for instance out-of-the-box content management systems or e-commerce applications which just need to be adapted to the look and feel of the company's brand. However, for large enterprises, it is necessary to build a web application tailored to their specific needs. Because there are no resources to start an entire web application from scratch, developers use frameworks. The latter offer basic functionality required in almost every web application. They ensure: code reuse, prevention of duplication, incremental development, easier and faster maintenance, and the separation of concerns, enabling better and more focused team work.

In practice, JEE applications have proven to be one of the best solutions for large-scale enterprise applications which need to be extended frequently and require constant maintenance. There is a large choice of available frameworks for the Java environment. The most famous one are Spring, Java Server Faces, Webwork, Seam, Tapestry and the Struts Action Framework 2.0. Previous research (Lanquetin, 2007) has shown, that the Struts Action Framework 2.0 is of greatest interest for a research project. Not only it is very new, but it also incorporates most of the modern ideas on how to implement web applications.

When using the term 'Struts', it is not question of the framework, but rather the community. In fact, Struts has two frameworks: Struts Action Framework 2.0 and Shale. For convenience, the framework will henceforth be referred to as 'Struts 2.0', which is shorter than Struts Action Framework 2.0.

Struts 2.0 is a result of the merging of Struts and Webwork 2.2, as both communities found out, that they had many common views on how a new framework should be implemented. The great advantage of the merge is that Struts has a large community and a very popular

brand name known to every JEE developer. Webwork, in turn, has a smaller community, but a better framework than Struts. Therefore, the Struts 2.0 framework is based on the Webwork 2.2 framework and does not have much in common with the old Struts 1.x version. However, Struts 2.0 profits from the large Struts community. This is why Struts 2.0 might have a very bright future, with a powerful framework, a large community and a strong name.

With first stable beta versions appearing in the end of 2006, Struts 2.0 is very new. Some open-source web applications were already released and made publicly available for demonstration purposes (Struts, 2007*b,c*). Moreover, a growing number of documentation started to show up in the beginning of the year, making it easier for developers to understand the framework and start their own web application.

Goal of this research is to find always recurring problems encountered in web applications development, find out how they are solved in Struts 2.0, and evaluate the solutions. By the end of the evaluation it should be clear, if it is worth starting a new project on the Struts 2.0 framework, or if the latter might just be another hype.

Chapter 2.

The Technologies

Struts 2.0 alone is only the framework required to develop a web application on. However, the web application will also need to be tested, built, deployed and hosted on an application server. Therefore, there are several technologies which must be taken into account. This chapter covers the key technologies required for a project and how they work together.

The actual web application development is based on the Struts 2.0 framework. Aim of the framework is to handle as much as possible from the common web application work, so that the developer can fully concentrate on implementing the business logic. In practice, developing an application on top of the framework means to inherit or implement certain classes of the framework, create results pages for the front-end and adapt the framework file descriptors¹ accordingly.

Once an application is programmed, it needs to be compiled and deployed. In JEE projects, this usually happens with the popular build tool *Apache Ant*. In 'modern' projects, it is however advised to use *Maven 2*, which has many advantages over Ant. Maven 2 compiles, packages and deploys the application to the Servlet container of an application server. During the compilation process all Java files are compiled into classes, the binary format. The packaging, in turn, consists in creating a web archive (WAR) or an enterprise archive (EAR), which is a single file containing all the resources of a web application. Finally, the deployment consists in moving the archive file to a Servlet container, so that the application can be accessed through the world wide web. Figure 2.1 visualises the Maven 2 processes described above.

The Servlet container, where the application is deployed to, is used to process the requests for the web application. In most cases the popular *Apache Tomcat* is used as Servlet container. However, more functionality than sole request processing is sometimes needed, in which case an application server is indispensable. Beside providing a Servlet container, an

¹File descriptors hold configuration information for the framework or the container to which the application will be deployed.

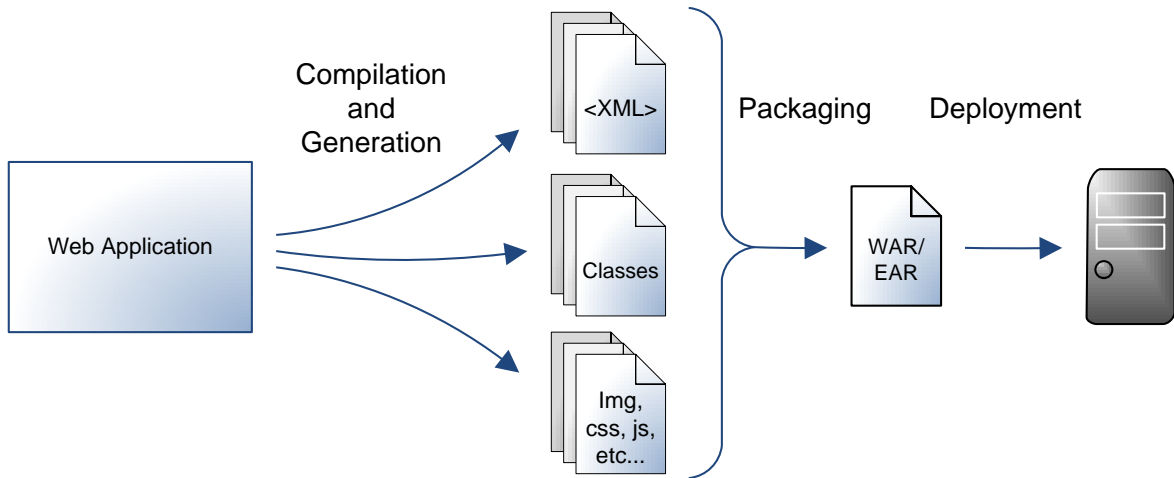


Figure 2.1: Deployment of a web application using Maven 2

application server offers even more functionality, such as handling persistence, caching or clustering. In our project we use *JBoss*, since it is the most used Java application server and comes with Tomcat as Servlet container.

To visualise how all these technologies goes together, Figure 2.2 illustrates the different layers. Each layer depends on the layer below it. To summarise, the web application is built on top of the Struts 2.0 framework and deployed into the Servlet container Tomcat, which in turn is part of the JBoss application server. Finally, JBoss runs in the Java virtual machine of a server's operating system.

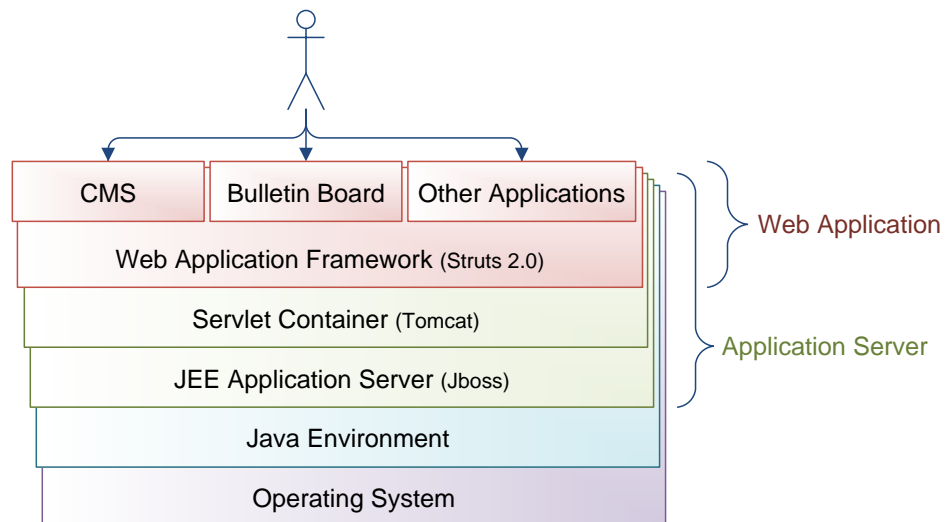


Figure 2.2: The different layers of used technologies

Chapter 3.

Evaluation of the Technologies

The previous Chapter covered the main technologies required to develop JEE web applications. This chapter will present common issues encountered when developing web applications. In a first step, it is researched how the technologies cope with each of the issues. Then, an evaluation of the findings highlights the advantages or probable disadvantages of the technology, comparing the results to older versions or alternative technologies where possible. Two of these technologies will be evaluated in detail: Maven 2 and Struts 2.0 .

3.1. Maven 2

To manage a project's life cycle in large-scale enterprise applications, relying on the IDE's capabilities alone is not enough. Special tools are required, which should at least allow testing, building and deploying of an application.

In most projects, developers tend to use Ant to build and deploy their application. *Ant* was an excellent solution for the past few years, but time and experience showed, that it is lacking of standardisation and reuse of code. Maven took notice of Ant's disadvantages and offers a better, standardised way as how to manage all important steps of a project life cycle. Koke (2007), a developer involved in major open-source projects, has defined six steps to improve and automate development for software projects. One of the steps is the move from Ant to Maven 2.

As mentioned in Chapter 2, Maven 2 is amongst others used to build and deploy web applications. The latter are usually packed into a so-called WAR or EAR file, which is then copied to a Servlet container. An EAR file is typically used for very large applications containing several WAR files. However, in most project only a single WAR file is required.

3.1.1. The Default Folder Hierarchy

To package a WAR file correctly, Maven 2 needs to know where to find the different resources. To avoid spending a lot of time telling Maven 2 where to find the files, a default folder hierarchy exists. It requires the developer to place the different resource files at the right locations. If the folder hierarchy needs to be changed, it is also possible to configure Maven 2 accordingly. However, Massol et al. (2006, p. 37) strongly advise to stick with default settings, and only tweak in last resort.

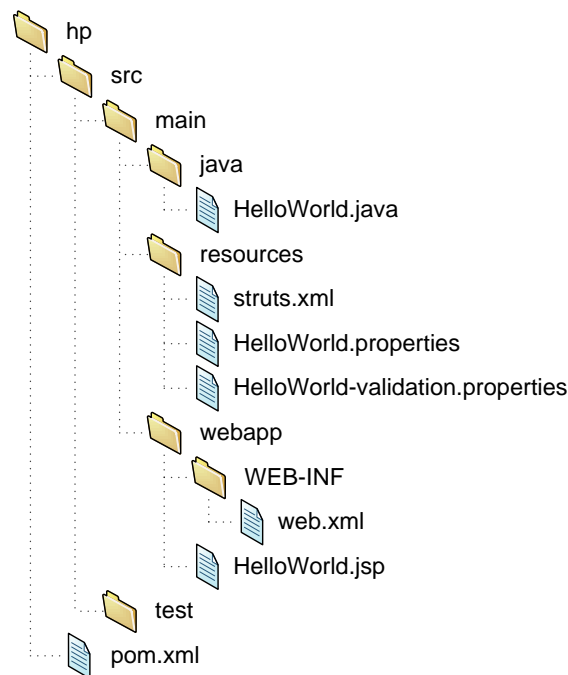


Figure 3.1: Folder Structure of the 'hp' project

Figure 3.1 shows a folder structure for a very basic web application containing only one HelloWorld Action. The interesting folder is `hp/src/main` as it contains the actual resources needed for the web application:

- `java` contains all the Java files;
- `resources` contains any resources files, such as localisation properties and input validation files;
- `webapp` includes the files which will be directly available from the browser, e.g. JSP, HTML or images.

The XML files `pom.xml`, `struts.xml` and `web.xml` are file descriptors used to configure the Maven 2 project, the Struts 2.0 framework and the Servlet container.

3.1.2. The Project Object Model

The heart of Maven 2 is the *Project Object Model*, or POM. The entire configuration for a Maven 2 project is held in the configuration file `pom.xml`, which must be placed in the root directory of the project, as shown in Figure 3.1. An extract of the `pom.xml` used in the honours project looks as follow:

Listing 3.1: `pom.xml` of the hp project

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- $Id: pom.xml 6 2007-04-19 17:27:36Z ps $ -->
3 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
  XMLElementSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
  maven.apache.org/maven-v4_0_0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <!-- parent projects here -->
6   <groupId>com.psbases.jboss</groupId>
7   <artifactId>hp</artifactId>
8   <packaging>war</packaging>
9   <name>Honours Project</name>
10  <url>http://jboss.psbases.com/hp/</url>
11  <dependencies>
12    <dependency>
13      <groupId>javax.servlet</groupId>
14      <artifactId>servlet-api</artifactId>
15      <version>2.4</version>
16      <scope>provided</scope>
17    </dependency>
18    <!-- more dependencies here -->
19  </dependencies>
20 </project>
```

Noteworthy are the `packaging` tag (line 8) and the `dependencies` tag (line 11–19). Possible values for `packaging` are `jar`, `war` and `ear`. By specifying `war` as packaging type, the project is automatically transformed into a web application project. The `dependency` tags in turn, list all the libraries needed for the web application. When a new dependency is added to the project, Maven 2 automatically downloads the dependency from a remote repository into a local repository. Maven 2 also comes as Eclipse plug-in, which makes the adding of dependencies very easy, as the user works through the GUI and do not have to change the `pom.xml` directly (see Section 3.2.9, Tool Support).

3.1.3. Goals in Maven 2

Maven 2 offers many default processes, such as compiling, packaging or deploying. These processes are called *goals*. The most important goal for our project is ‘package’, as it compiles the application, runs JUnit¹ tests (if they are available) and packages the application into the WAR file.

Listing 3.2 shows the output of the ‘package’ goal.

Listing 3.2: Execution of Maven’s ‘package’ goal

```
1 E:\projects\java\hp\applications\hp>mvn package
2 [INFO] Scanning for projects...
3 [INFO] -----
4 [INFO] Building Honours Project
5 [INFO]    task-segment: [package]
6 [INFO] -----
7 [INFO] [antrun:run {execution: copy-sources}]
8 [INFO] Executing tasks
9   [copy] Copying 20 files to E:\projects\java\hp\applications\hp\target\hp\WEB-INF\
   src\java
10  [copy] Copying 12 files to E:\projects\java\hp\applications\hp\target\hp\WEB-INF\
   src\java
11 [INFO] Executed tasks
12 [INFO] [resources:resources]
13 [INFO] Using default encoding to copy filtered resources.
14 [INFO] [compiler:compile]
15 [INFO] Compiling 20 source files to E:\projects\java\hp\applications\hp\target\classes
16 [INFO] [resources:testResources]
17 [INFO] Using default encoding to copy filtered resources.
18 [INFO] [compiler:testCompile]
19 [INFO] Nothing to compile - all classes are up to date
20 [INFO] [surefire:test]
21 [INFO] No tests to run.
22 [INFO] [war:war]
23 [INFO] Exploding webapp...
24 [INFO] Assembling webapp hp in E:\projects\java\hp\applications\hp\target\hp
25 [INFO] Copy webapp webResources to E:\projects\java\hp\applications\hp\target\hp
26 [INFO] Generating war E:\projects\java\hp\applications\hp\target\hp.war
27 [INFO] Building war: E:\projects\java\hp\applications\hp\target\hp.war
28 [INFO] -----
29 [INFO] BUILD SUCCESSFUL
30 [INFO] -----
31 [INFO] Total time: 8 seconds
32 [INFO] Finished at: Fri Apr 20 07:33:47 BST 2007
33 [INFO] Final Memory: 6M/12M
34 [INFO] -----
```

The output is interesting at this point, because it demonstrate the different steps Maven 2 goes through. Each step is introduced with square brackets, e.g. [compiler:compile]

¹JUnit is a testing framework written in Java and used to unit test Java application.

(line 14). All these steps are implemented by default in Maven 2, contrary to Ant, where each had to be written from scratch by the developer.

After the successful execution of the 'package' goal, the created WAR file contains the entire web application and is ready to be used in a Servlet container such as JBoss' Tomcat.

3.1.4. The WAR File

The exploded² WAR file has a similar structure than the one on the local file system (c.f. Section 3.1.1, The Default Folder Hierarchy). Figure 3.2 shows the folder tree and the most important files of the WAR.

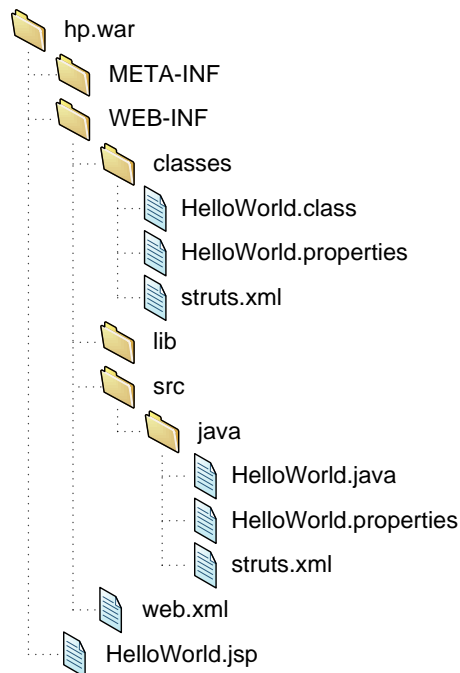


Figure 3.2: Folder Structure of the exploded hp.war file

Two new folders, META-INF and lib, appeared in the WAR file. The META-INF folder contains information related to Maven 2, and is needed for other projects which would like to use the project as a dependency. As for the lib folder, it contains all required libraries defined in the dependencies section of the pom.xml file (c.f. Section 3.1.2, The Project Object Model).

²A WAR file is nothing other than a packed file. Therefore, in this context, the term 'exploded' means unpacked.

Note that the libraries were held in the local repository, and are only duplicated to be included in a packaged file, such as the WAR file. Also, apart from specifying WAR as the value for the packaging tag, the packaging process did not need any additional configuration, as Maven 2 knows where to find the resources to package the WAR since they are placed in a default folder hierarchy. Ant, on the other hand, requires laborious configuration, because it requires a lot of information on where to find the files and where to copy the files.

3.1.5. Summary of Maven's Evaluation

The previous sections gave an insight as to how fast and easy a web application can be built and deployed using Maven 2.

Maven 2 is built upon the principle of 'convention over configuration': Because Maven 2 has pre-built processes based on a standardised folder structure, it is not required to write long configuration files for every single process, which was the case for projects built with Ant. Moreover, because of that standardised folder structure, it is easier and faster for developers to get acquainted with new projects using the same folder structure. Not only does Maven 2 lessen the time required to build a project, but it also offers a coherent repository, where all dependencies are held. The system is based on a dependencies model, which already proved to be very useful in popular packaging systems known from Linux distributions such as Debian (McCuaig, 2003) or Gentoo (Vermeulen et al., 2007).

However, Maven's strength is also its weakness: Its pre-built goals are not written anywhere, which can make it hard for newcomers to understand what Maven 2 actually requires for its goals to run. Because goals had to be explicitly written in Ant, it was possible to read what they are actually doing. Maven 2, on the other hand, assumes that the developer knows what the goals are doing and that the required resources are placed in the right locations in the folder hierarchy. Therefore, Maven 2 could need some more time to get used to.

Nevertheless, there are no doubts, that Maven 2 will have a bright future, as more and more developers will discover the advantages analysed in this project. Struts 2.0 has already noticed Maven's potential, as they use it for the development of the Struts 2.0 framework itself.

3.2. Struts 2.0

Section 3.1 highlighted the advantages of using Maven 2 over other build tools like Ant. We could see that it is used to effectively build and deploy web applications. The web application, in turn, needs to be developed first. This is done using the Struts 2.0 framework which offers a solid ground to build a web application on. The following sections will describe what makes the framework so special.

3.2.1. The Framework Architecture

Struts 2.0 is a *Model 2* implementation, meaning that Servlets and Java Server Pages (hereafter JSP) are used together. The Servlets realise the business logic, whereas the JSPs take care of writing HTML code. Struts 2.0 follows the classic *Model-View-Controller* (henceforth MVC) design pattern which was first introduced with the SmallTalk MVC framework. The pattern is based on dividing the code into three distinct parts consisting of the retaining of the state (Model), the rendering of the Model (View), and the selecting of the View (Controller). Figure 3.3 shows a diagram summarising each of the different parts. The main advantage of using a MVC pattern, is the separation of concerns, which drastically improve team work, as back-end and front-end can be developed individually. It also betters maintenance and configuration of the application.

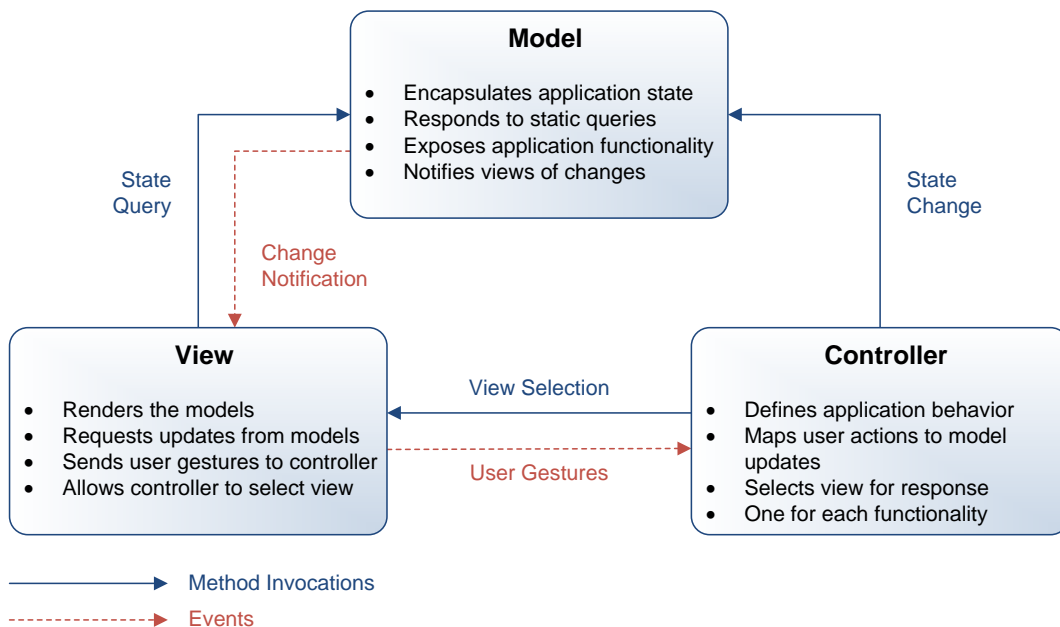


Figure 3.3: Diagram of the Model-View-Controller pattern. Source: Sun (2007)

At the heart of the Struts 2.0 framework, there are three key pieces: The Actions, the Results and the Interceptors. The following Sections will cover each of these elements.

Actions

Every web page or resource of the framework is implemented through a so-called *Action*. When a request is received by the application, the framework selects an Action to handle the request, and more specifically, a *method* of the Action class. After the execution of an Action method, the Action can return a string token, like 'input' or 'success', to indicate which view should be presented to the user. Listing 3.3 shows an example of a Login Action, which should clarify how the latter works.

Listing 3.3: An Action example: Login.java

```
1 public class Login extends com.opensymphony.xwork2.ActionSupport.ActionSupport {
2
3     public String execute() throws Exception {
4         if (isInvalid(username)) return INPUT;
5         if (isInvalid(password)) return INPUT;
6         return SUCCESS;
7     }
8
9     private boolean isInvalid(String value) {
10        return (value == null || value.length() == 0);
11    }
12
13    private String username;
14    private String password;
15
16    public String getUsername() {
17        return username;
18    }
19    public void setUsername(String username) {
20        this.username = username;
21    }
22    public String getPassword() {
23        return password;
24    }
25    public void setPassword(String password) {
26        this.password = password;
27    }
28 }
```

By default the method `execute()` will be invoked, if not specified otherwise. The member fields `username` and `password` will have been automatically set, if they are transmitted in the HTTP request (see Section 3.2.2, Data Binding). This is for instance the case, when a

form has been submitted. Depending on the returned string token from the `execute()` method (`INPUT` or `SUCCESS`), a *Result* will be displayed to the client.

Results

The *Result*, which is the View of the MVC pattern, is generated once the Action has been executed. It can be a simple XHTML file, generated through JSP, or any other imaginable type, such as a PDFs or images. The method used for the Result generation can be realised through any possible technology available, including JSP, Velocity or XSL. Lightbody, Husted and Luppens (2006) list a total of twelve predefined Result types available in Struts 2.0.

Interceptors

Beside the effective MVC architecture, Struts 2.0 also provides an ingenious Interceptors concept, making the framework very flexible and extensible. Figure 3.4 visualises how this concept works.

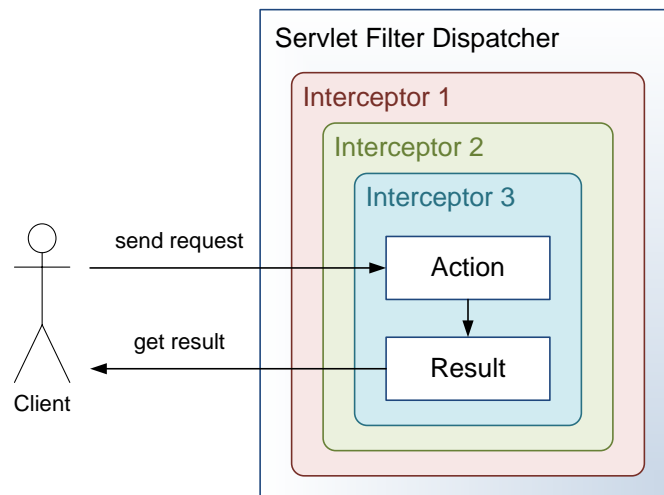


Figure 3.4: Interceptor Architecture

When a client requests an Action, the request is first processed by so-called *Interceptors*, as shown in Figure 3.4. Very basic Interceptors handle validation, check for duplicated submits or apply internationalisation (i18n). Other Interceptors, such as the `ExecuteAndWaitInterceptor`, can provide automatic ‘Please wait...’ pages for long-running requests,

for instance when booking a journey. It is also possible to configure Interceptors which run only after the Action has been executed.

For every available Action, it can be configured, which Interceptors and Results should be used. Listing 3.4 shows an extract of a very basic `struts.xml`. The latter is a file descriptor, which holds all configuration related to the Struts 2.0 framework.

Listing 3.4: Extract of the `struts.xml`

```
1 <package name="default" extends="struts-default">
2   <interceptors>
3     <interceptor name="timer" class=".."/>
4     <interceptor name="logger" class=".."/>
5   </interceptors>
6
7   <action name="login" class="Login">
8     <interceptor-ref name="timer"/>
9     <interceptor-ref name="logger"/>
10    <result name="input">login.jsp</result>
11    <result name="success" type="redirect-action"/>/secure/home</result>
12  </action>
13 </package>
```

In the example above, an Action named `login` has been configured, which will be handled by the `Login` class. Furthermore, the logging Action requires two Interceptors: `timer` and `logger`. Note that the order is also important. Finally, the Result is generated through one of the defined results, depending on the returned string token from the Action (either `input` or `success`).

At this date, many Interceptors have already been implemented: Brown et al. (2007) published a list of all available Interceptors in version 2.0.6 of Struts, totalling 27 Interceptors. Also, most Actions will not need to explicitly specify which Interceptors they require, as there is a default Interceptor stack. In order to improve the performance, it is however advised to explicitly reference only the used Interceptors. Additionally, if an Interceptor is missing, it is possible to create an own Interceptor, by implementing the provided Interceptor interface or by extending `AbstractInterceptor` (Husted, 2006a). For instance, an own Interceptor could be written to ensure that a user is authenticated, before the Action is processed.

The great advantage of Interceptors, is that they provide a unique class to process common functionality. This greatly supports the principle of code-reuse, and make maintenance of the application easier. Furthermore the Interceptors handle tasks, which would otherwise have been implemented in the Action classes. It is, however, a good practice to have an as short as possible Action, containing only the business logic required for a specific task.

Everything else, related to the web application or the framework, should be handled by the Interceptors.

The Value Stack

Beside the clever Interceptor concept, Struts 2.0 comes with another ingenious feature: the value stack. To grasp the meaning of the latter, it must first be understood how data is passed from the Action to the Result. In most frameworks, this happens by placing Objects in a Servlet scope, which in turn can be accessed from the JSPs. Four scope exists in Struts: page, session, request and application scope. When a JSP tries to access an Object, it will look in each scope in the cited order, until it finds the Object.

Struts 2.0, however, uses an additional scope, the value stack, to store Objects. When an Action is executed all its accessible JavaBean fields³ are automatically pushed onto the value stack and made available to the JSP.

The value stack comes in handy, when it is necessary to access Bean data from Objects such as Collections. This usually happens in an iteration in the JSP. An example will clarify how this works: Assuming that a Collection of Person Objects was pushed from an Action onto the value stack, it is then possible to iterate over the Collection using the Struts 2.0 tag library as demonstrated in Listing 3.5 (c.f. Section 3.2.6, Struts 2.0 Tag Library).

Listing 3.5: Accessing the `personList` Collection from JSP

```
1 <s:iterator value="personList.person">
2   <p>
3     Name: <s:property value="name"/> <br/>
4     Gender: <s:property value="gender"/> <br/>
5     Age: <s:property value="age"/>
6   </p>
7 </s:iterator>
```

The great advantage of the value stack is that it is not necessary to use qualified references, such as `personList[0].name`, to access the Person Objects inside the `personList` Collection. This is because the `<s:iterator>` tag pushes the currently iterated Person Object onto the value stack for the time of an iteration, allowing direct access of the Person's getters. If another Person Object was already on the value stack before the iteration took place, this Object will neither be accessible, nor overwritten. This is because of the 'last-in, first-out' principle of stacks: All Person Object from the iteration will be popped from the stack as soon as the iteration is over, making the previous Person Object accessible again.

³These are the fields which are accessible through the public getter methods (e.g. `getName()`).

The value stack is definitely one of the greatest features in Struts 2.0, as it prevents direct access to Bean data from an Action and makes accessing the data very easy.

3.2.2. Data Binding

Data binding is needed, when a web form is being submitted to the server. Contrary to Java, HTTP is not aware of data types. Therefore, when a form is being submitted, all strings need to be converted to the correct type used in the application. Struts 2.0 handles the type mismatch on its own, since it has a built-in type conversion (Lightbody, Hermanns, Luppens, Husted and Barroso, 2007).

Best practice is to use domain Objects directly, instead of using simple types. The first are usually *JavaBeans*, Objects consisting of member fields and getters and setters accordingly. An example of a JavaBean and a JSP extract will clarify the easy way of getting data from and to a JSP. Figure 3.5 shows an UML class diagram of the Person JavaBean.

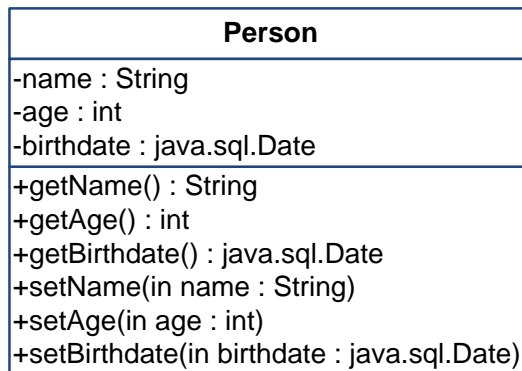


Figure 3.5: UML class diagram of the Person JavaBean

In the JSP file, the JavaBean can then be accessed using the Struts form tags, as shown in Listing 3.6.

Listing 3.6: Accessing JavaBeans from a JSP

```
1 <s:form name="editForm" action="save">
2     <s:textfield name="person.name"/>
3     <s:textfield name="person.age"/>
4     <s:textfield name="person.birthdate"/>
5 </s:form>
```

When the form is submitted, the JavaBean Person is automatically populated with the content of the form values. If the types used in the JavaBean differ from the submitted strings, as it is the case for the fields age or birthday, Struts 2.0 will convert the submitted

strings into the expected types `int` and `java.sql.Date` accordingly. This task is done by a pre-built Interceptor, which is included in the default Interceptor stack used by all Actions, if not specified otherwise (c.f. Section 3.2.1, The Framework Architecture). Of course Struts 2.0 allows the configuration of default error messages for exceptions arising during the conversion. This can be configured in validation XML files (c.f. Section 3.2.3, Validation).

Furthermore Struts 2.0 also handles the conversion of values for more advanced types, including Collections and Maps. The latter are for instance required in drop-down lists or multiple choice lists. In special cases, the developer will need a conversion, which is not included in Struts 2.0 by default. For this case Struts 2.0 offers the possibility to create own converters, simply by extending the `StrutsTypeConverter` class. Two methods can be overridden: `convertFromString(..)` and `convertToString(..)`. These methods do the conversion from the type used in the application, and the string used in the HTTP request or response.

Another powerful feature provided by Struts 2.0, is the creation of new Object if these where not provided by the Action. If for instance a JSP contains a form with a list of strings which where nowhere defined in the Action, the framework will create the list and make it available to the Action. Of course this procedure implies some special care with regard to security.

In short, it is no more necessary to cast strings to types using own implemented utility classes. Struts 2.0 handles the entire data binding. It holds all the possibles type conversions at a central place, thereby ensuring code-reuse and easier maintenance. Again, the Action class is freed from unnecessary lines of codes, which consequently makes it easier to fully concentrate on the business logic.

3.2.3. Validation

Validation is another issue, which needs to be taken care of in every web application. Struts 2.0 solves this problem with an Interceptor as well (Lightbody, Gielen, Luppens, Brown, Husted and Barroso, 2007).

A set of common validation rules is already provided by the framework. In total, there exists twelve validation rules, which can be used out of the box. Amongst others, these provide validation for required fields, email addresses, regular expressions, date ranges, and many more.

Struts 2.0 offers the possibility to do validation on Action or field level. This means, that it is possible to generate error messages, which can be shown for an entire Action or for each single field of the form.

There are three ways to handle validation in Struts 2.0:

1. All validation rules can be configured in a `validation.xml` file belonging to an Action.
2. Alternatively, the validation can be configured as annotations directly inside the Action class.
3. Another possibility is to make the Action implement the `Validateable` interface and write the validation in the `validate()` method provided by the interface. This however, should only be done for more complex validation rules, which require database access for instance.

An example will demonstrate how easy validation rules can be set up. The example consists in calling the Registration Action, where the user has to submit a form containing an email address and a password. The password must be supplied twice to make sure that the user did not misspell it.

The first solution is to realise the validation through a `validation.xml` file. Listing 3.7 shows the XML file which the Interceptor needs in order to validate the submitted form.

Listing 3.7: Registration-validation.xml belonging to the Registration Action

```
1 <!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
2   "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
3
4 <validators>
5
6   <field name="emailAddress">
7     <field-validator type="requiredstring">
8       <message key="error.emailddress.required"/>
9     </field-validator>
10    <field-validator type="email">
11      <message key="errors.email"/>
12    </field-validator>
13  </field>
14
15  <field name="password">
16    <field-validator type="requiredstring">
17      <message key="error.password.required"/>
18    </field-validator>
19    <field-validator type="stringlength">
20      <param name="trim">true</param>
21      <param name="minLength">4</param>
22      <param name="maxLength">10</param>
```

```
23     <message key="errors.range"/>
24   </field-validator>
25 </field>
26
27 <field name="password2">
28   <field-validator type="requiredstring">
29     <message key="error.password2.required"/>
30   </field-validator>
31 </field>
32
33 <validator type="expression">
34   <param name="expression">password eq password2</param>
35   <message key="error.password.match"/>
36 </validator>
37
38 </validators>
```

Every field validation is realised through the `field` tag, where as many as needed validation rules can be defined. Since every field of the form is mandatory, a validation of type `'requiredstring'` is added to each field tag. Furthermore, it can be ensured that the entered email is a valid one by adding the `'email'` type (line 10–12), or that the password should be between 4 and 10 characters long with the `'stringlength'` type (line 19–23). Finally, an error message on Action level can be added if the two passwords do not match (line 33–36). The error messages in the form are made available through properties files (see Section 3.2.5, Internationalisation (i18n)).

The second solution to implement validation is to use annotations directly in the Action class (Hermanns, Husted and Brown, 2006). Annotations is a Java specific feature available since Java 5. It consists in adding meta information directly in the programme code. Listing 3.7 shows how this is done.

Listing 3.8: The Registration Action with annotations

```
1 @Validation()
2 public class Registration extends ActionSupport {
3
4   private String emailAddress;
5
6   @RequiredFieldValidator(type=ValidatorType.FIELD, message="The email is required")
7   @EmailValidator(type=ValidatorType.FIELD, message="The email is not valid")
8   public void setEmailAddress(String email) {
9     this.email = emailAddress;
10  }
11
12  public String getEmailAddress() {
13    return emailAddress;
14  }
15
16  // the other methods where left out
```

As shown in the Action class, it is possible to configure validation by adding annotations preceding the setters of the fields, which needs to be validated. This was already done in earlier frameworks using *XDoclet*. The latter, however, consists in writing meta information into the Javadoc of the methods and classes. *XDoclet* might have become a pseudo-standard, but fact remains, that Java annotations are part of Java and do not need any additional tools. Using these annotations has huge advantages: It prevents the maintenance of two separate files and ensures that all the Action fields are covered during a validation.

Finally, to avoid unnecessary duplication of validation, it is possible to define global validation rules, which can be used by any Actions. This ensures code-reuse and easier maintenance.

One of the most powerful features in Struts 2.0 new validation system, is without doubt the client-side validation (Lightbody and Husted, 2006). Amongst others, it uses AJAX, and more specifically Getahead's Direct Web Remoting (DWR) which allows Javascript to interact with Java on a server (Getahead, 2007). Without any further programming effort, the entire validation can take place on the client-side. The effect is the same, with the difference that no request has to be sent back to the server, thereby lessening server round-trips and giving the user a even faster feedback.

Summing up, the validation is very well implemented in Struts 2.0. Developers have the choice between different approaches, either by decoupling validation from Actions, or managing both inside Actions using annotations. Furthermore, Struts' validation allows for customisation, which makes the framework very flexible. Finally, the clever client-side validation makes the application web 2.0 ready.

3.2.4. Double Submits

Double submits are a common problem when designing web applications. A double submit occurs, when the user submits a form more than once. This can happen, when the user double-clicks a button, or clicks a button several times because the page is not loading quick enough.

Struts 2.0 can manage double submits, by enabling the 'token' or 'token-session' Interceptor for an Action where a double submit can occur (Husted, 2006*b,c*). These Interceptors save a struts token as hidden field using the `<s:token/>` tag, so that a double submit can be recognised by the framework. If the submitted token does not match the struts token,

the request and all following requests are disregarded. In the past, with Struts 1.x, double submits must have been coded into the application logic, which made the programme code longer and more difficult to maintain.

3.2.5. Internationalisation (i18n)

For international corporation it is important to support several languages inside the same web application. In Struts 2.0, internationalisation is realised through resource bundles, property files which must be placed in the class path of the application. They are used by the Struts 2.0 tags and during the validation.

The resource bundles can be stored in seven different locations. One possibility is to include the resources in a file `<ActionName>.properties`, belonging to an Action with the same name. An alternative is to put the messages into a more global properties file, which can be accessed from several Actions. When a message resource is required, it will be searched through these locations in a predefined order. It is therefore possible for individual classes to override global messages in their own properties files.

Using Struts 2.0 tags, it is possible to access the messages resources in three different ways described in Listing 3.9.

Listing 3.9: Accessing message resources in JSP

```
1 <!-- using getText() -->
2 <s:property value="getText('some.key')" />
3
4 <!-- using the text tag -->
5 <s:text name="some.key" />
6
7 <!-- using the i18n tag -->
8 <s:i18n name="some.package.bundle" >
9     <s:text name="some.key" />
10 </s:i18n>
```

Internationalisation is realised by the naming pattern of the resource bundles. For Swedish a file `FooAction_se.properties` has to be created, containing the country code 'se' in its name. When the language is switched to Swedish, all files containing the 'se' country code will be used instead of the default property files.

Summing up, Struts 2.0 proposes to separate JSP code from actual text, by saving the text messages in properties files. This ensures that language dependent text can be maintained separately. Additionally, the text messages can be saved in different location. The advantage is that messages can be defined for several Actions at once, with the possibility to

override these messages for specific Actions. Furthermore Struts 2.0 offers three ways to access messages, leaving the developer a lot freedom about which method to choose. However, it is not clear if the messages can be stored in other places, such as a database or XML files. This could become a problem for companies with already available messages.

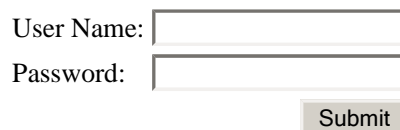
3.2.6. Struts 2.0 Tag Library

When designing an HTML interface, a lot of time is spent in writing HTML code. In Struts 2.0, most of that work is done by *Struts 2.0 tags* (or *UI tags*), which provide form controls to dynamically create XHTML and Javascript code. To demonstrate the ease of use, Listing 3.10 shows a very basic Login form using form controls.

Listing 3.10: Login form demonstrating Struts 2.0 tags

```
1 <s:form action="Login" validate="true">
2   <s:textfield key="username"/>
3   <s:password key="password" />
4   <s:submit/>
5 </s:form>
```

This short snippet produces a XHTML form, realised as a two-column table with the field labels in the left column, and the text fields in the right column. Figure 3.6 shows a screenshot of the rendered HTML.



The screenshot shows a simple login form. It consists of two rows of labels and input fields. The first row has the label "User Name:" followed by a text input field. The second row has the label "Password:" followed by a password input field. Below these fields is a "Submit" button.

Figure 3.6: Screenshot of the rendered Login form

The look-and-feel of the form can be influenced by specifying a *theme*. Themes are a very useful way to adapt the look of the outcome at a central place. Moreover, themes can extend each others, so that the entire theme does not need to be rewritten. In the example above, the XHTML theme was chosen, which produces a simple two-column layout. Other themes exist, such as the AJAX theme, which allows the user to get dynamic content from the server.

Furthermore, Struts 2.0 fixes a misconception of the HTML standard, which is to specify an Action in the form tag. The framework allows to bind Actions to buttons inside the form. To make this possible, Struts 2.0 generates Javascript code which calls the right Action, depending on the pressed button.

Another problem with HTML are checkboxes. The latter are usually used to represent boolean values. The problem with checkboxes are that unchecked checkboxes are not submitted with the request. In Struts 1.x, developers had to use a long-winded reset method as a work-around for this issue. Struts 2.0, in turn, handles checkbox state automatically, meaning that the framework can detect if a checkbox was checked or not.

It is also possible to use parameters when calling Struts 2.0 tags. This is done with the param tag. The following JSP snippet for instance calls the url tag with the host name as parameter:

```
1 <s:url action="Subscription_edit"><s:param name="host" value="google.com"/></s:url>
```

The following string is then rendered:

```
1 Subscription_edit.do?host=google.com
```

In summary, the Struts 2.0 tags are a very powerful feature of the Struts 2.0 framework. They allow to easily construct a thorough and always same-looking layout. Furthermore developers can focus exclusively on the form elements, without having to worry about HTML issues. The only problem which could come up, is that Struts 2.0 tags are not flexible enough. However, Struts 2.0 countered this inflexibility by providing the concept of themes, which allows for complete customisation of the design.

3.2.7. Testing Framework

Regular testing is the key to quality releases. Extreme programming even suggests a test-first approach, where tests are written before the programme implementation. In JEE projects, unit tests proved to be the most successful approach to allow a maximum test coverage. The latter consists in testing each single unit of the application independently.

For the unit testing, the traditional tool JUnit is the most used. It is available as stand-alone application or integrated in most IDE's such as Eclipse. Furthermore, it can be run as GUI or in the command line.

Usually, when testing the front-end of web applications, it is necessary to go through a HTTP request to check for expected responses from the application. Struts 2.0 proposes an easier way, which allows to test the application directly without the requirement of a request. An example will demonstrate how easy a test can be implemented. Listing 3.3 on page 12 showed the Login Action which will be tested.

Listing 3.11: LoginTest.java

```
1 public class LoginTest extends org.apache.struts2.StrutsTestCase {
2
3     public void testLoginSubmit() throws Exception {
4         Login login = new Login();
5         login.setUsername("username");
6         login.setPassword("password");
7         String result = login.execute();
8         assertTrue("Expected a success result!", ActionSupport.SUCCESS.equals(result));
9     }
10
11 }
```

As shown in the above class (Listing 3.11), the first step is to extend `StrutsTestCase`. Each public method of the `LoginTest` class will automatically be run when the test case is launched. To test the Login Action, it is necessary to create a new Login Object (line 4) and populate it with the data that would have normally be sent over a HTTP request (line 5–6). Then the method that should be tested is called (line 7), and it is checked if the returned string correspond to what was expected (line 8).

Testing is not just limited to JUnit inside an IDE. It is also a mandatory step when packaging the web application using Maven 2. Listing 3.2 on page 8 showed the package goal of Maven 2. As mentioned before, many goals are run when packaging the web application. One of them is the `[surefire:test]` goal (line 20–21 of Listing 3.2), which runs all the unit tests available in the application⁴. Since all the goals depend on each others, the web application will not be packaged if one of the goals fails. In other words, the developer must ensure a failure free product if they want to create a WAR file.

3.2.8. Support & Resources

Support and documentation is a key argument when choosing a technology. It must be guaranteed that the technology has a large community, which provides documentation, is aware of developers needs, and constantly improves the software. Community activity can be measured through the number of active users in mailing lists, message boards, chats and websites dedicated to the technology.

Websites

For Struts 2.0, most information can be found in the Struts 2.0.6 documentation (Struts, 2007a). The latter is realised as Wiki, maintained by very important developers, including

⁴For the unit test to run, they must be placed in the test folder (as shown in Figure 3.1 on page 6)

P. Lightbody, a leading committer of the WebWork application framework, and T. Husted, a release manager and senior member of the Struts development team.

As for Maven 2, it has a detailed Documentation (Maven, 2007b) with useful 'getting started' tutorials. It is however recommended to read the very complete book 'Better Builds with Maven' by Massol et al. (2006).

Finally, m2eclipse does not require much documentation, as it is quite easy to use. Codehaus (2007) provides two very useful flash animations, which explains how to install the plug-in in Eclipse, and how to use the plug-in.

Overall, the online resources for the technologies are sufficient to be able to realise a project. It must however be added that the documentation was very poor in the year 2006, and that there has been a lot of improvement, especially for Struts 2.0, whose Wiki is growing every month.

Mailing Lists

Mailing Lists are one of the most used ways to communicate within a community. For our project, the following mailing lists are of interest:

- Struts' user mailing list: user@struts.apache.org
- Webwork's user mailing list: users@webwork.dev.java.net
- Maven's user mailing list: users@maven.apache.org
- m2eclipse's user mailing list: user@m2eclipse.codehaus.org

Of course the key technologies, Struts 2.0 and Maven 2, are the most important ones. However, the Webwork's mailing list is worthy of note as well, since Struts 2.0 resulted of the merging of Struts and Webwork. The latter was used as basis to built Struts 2.0 on. Therefore, the Webwork mailing list covers most topics, which are of interest in Struts 2.0 as well.

With a total of over 5000 messages for Struts and 7000 messages for Maven in the year 2007, these technologies are well covered by the community (see Figure 3.7).

Since the Struts mailing list is still used for the older Struts versions (1.x), it is crucial to analyse the messages with regard to their content. Analyses of messages posted in April 2007 show that almost two-thirds of the threads concern issues encountered in Struts 2.0.x (see Figure 3.8). These results implies that Struts 2.0 is not only an active topic, but also that the Struts 1.x users moved on and tried the new version.

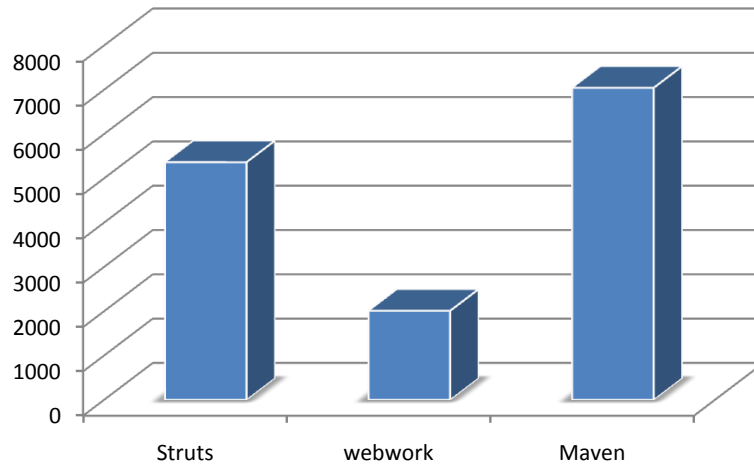


Figure 3.7: Messages posted in the mailing lists from January to April 2007. Source: Apache (2007b,c); Webwork (2007)

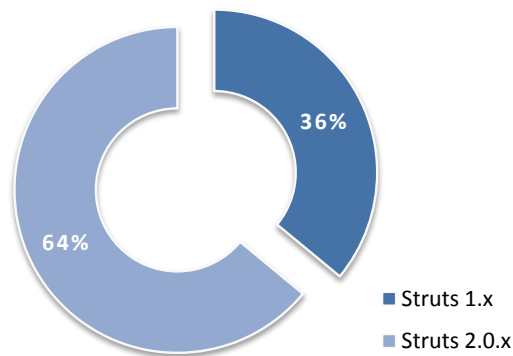


Figure 3.8: Proportion of Struts 1.x and Struts 2.0.x threads in the Struts mailing list in April 2007. Source: Apache (2007c)

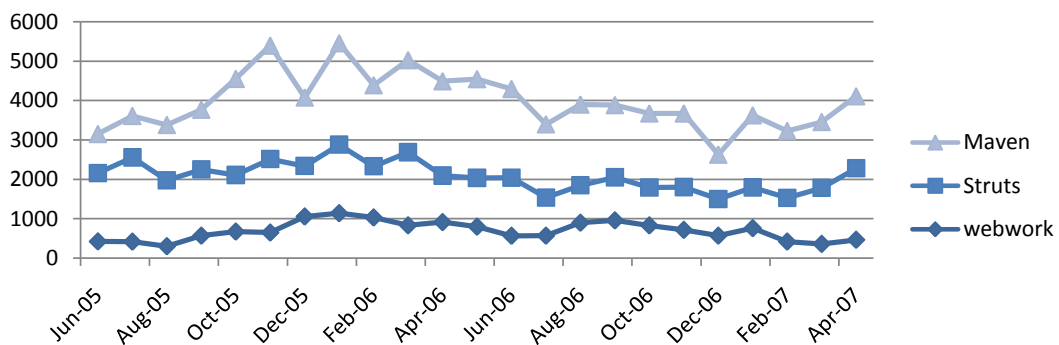


Figure 3.9: Number of messages posted monthly to the mailing lists. Source: Apache (2007b,c); Webwork (2007)

As shown in Figure 3.9 the mailing lists are all actively used through the last few years. Noteworthy is the slight decrease of Webwork's mailing list activity, which could however be interpreted as a consequence of the merge, since the decrease started with the appearing of Struts' first beta versions in the end of 2006. However, no considerable activity decrease is noticeable for the Struts and Maven mailing list; their activity is constantly increasing since February 2007, reflecting a growing interest for these technologies.

Internal Relay Chat (IRC)

The Internal Relay Chat (henceforth IRC) permits users to chat together in so-called channels, or have individual discussions in private chats. Most technologies have an IRC channel dedicated to that technology, enabling users to seek help, or exchange ideas and opinions.

Struts discussions take place in the channel #struts on irc.freenode.net. With over 30,000 users, freenode has the largest IRC community for open-source related technologies (Freenode, 2007). The #struts channel, however, is quite small with an average of 10 to 20 users, and does not provide much help, as users only idle in the channel.

Maven 2 has a much more active IRC community, with an average of 50 users on their channel #maven on irc.codehaus.org. In this channel, chances are good to get help in real-time.

To summarise, each technology, especially Struts 2.0 and Maven 2, provides a solid base of documentation through their website. As for problems or questions, they will most likely be handled through the very active mailing lists.

3.2.9. Tool Support

Tools can save time on more complicate or redundant tasks. Therefore it was researched, which tools could support the project or the framework.

m2eclipse Plug-in for Eclipse

As mentioned in Section 3.1, Maven 2 is a valuable tool, which greatly improves the project development. Usually, it is used in the command prompt, but as an Eclipse plug-in, Maven 2 is enhanced by an easy to use GUI. However m2eclipse was difficult to set up, because of dependency problems encountered when used as plug-in within Eclipse. The

problem comes from a transitive dependency not finding its artifact (Bolcina, 2007a). In other words, because a library (tools.jar) could not be found in Struts' dependency tree, the project could not compile. The proposed solutions (Bolcina, 2007b; Maven, 2007a) were without success. Because dependencies are stored on the local repository and made available to Eclipse through the Maven 2 plug-in, Eclipse was missing important libraries to run and compile the code correctly. A temporary work-around was to copy the dependencies to a library directory, which Eclipse had access to, so that it was still possible to work in Eclipse. Consequently, Maven 2 had to be used through the command line prompt.

To summarise, m2eclipse is certainly a useful tool, making the development faster and easier, since it acts as user-friendly graphical interface to the command line version of Maven 2. However, it can cost a lot of time to resolve encountered problems.

XDoclet

XDoclet is one of the most famous tools in JEE development. The principle is to add so-called XDoclet tags in JavaDocs, which are, strictly speaking, only comments in the programming code. However, the comments are contained inside special brackets (`/**` and `*/`), which allows the XDoclet preprocessor to read the XDoclet tags before the actual compiling. XDoclet can therefore easily extract information about Actions or beans, and generate the appropriate descriptors, or even classes. However, no XDoclet tags were defined for Struts 2.0 yet. Nevertheless, since Struts 2.0 is built on the Webwork framework, it is possible to use the already defined webwork tags.

Apart from XDoclet, there exist no tools yet which would support Struts 2.0 projects. However, the framework is very young, so it might be possible, that new tools will appear in a near future.

3.2.10. Performance

Performance is a key factor when choosing the technologies to implement and host a web application. However, performance is expensive, as it requires high-end CPU's or web-farms. An application consuming less CPU and memory, can be hosted on cheaper hardware or support more simultaneous users. Allowing a high number of users is very important for large enterprise applications, which JEE claims to be made for.

For this reason some load-tests have been realised on a Struts 2.0 web application in the practical part of the project. The load-testing was split into two steps: The first, is a single user load-test, which analyses which Struts 2.0 features takes the longest to process,

whereas the second step truly load-test the application with multiple simultaneous users. For both tests the same test scenario was used.

Test Scenario

For the test scenario, special care was spent in incorporating as much Struts 2.0 features as possible. The following list explains in detail which steps a test user had to go through during one iteration:

1. A user visits the homepage of the application (Figure A.1, p. 38),
2. logs in with username and password (Figure A.2, p. 38),
3. comes to a main menu (Figure A.3, p. 38),
4. clicks her way through to a registration page (Figure A.4, p. 39),
5. adds an email account to her subscriptions (Figure A.5, p. 39),
6. deletes the just added subscription (Figure A.7, p. 40),
7. changes her profile information, but receives a validation error (Figure A.8, p. 41),
8. goes to the main menu (Figure A.3, p. 38),
9. and finally logs out (Figure A.1, p. 38).

Single User Load-Test Results

A single user load-test was done, in order to see the strength and limits of each Struts 2.0 feature. Table 3.1 shows a summary report of the load-test results for a single user doing the above scenario 50 times in a loop.

Label	# Samples	Average	Min	Max	Error %	Throughput	KB/sec	Avg. Bytes
/struts2-mailreader-2.0.6/Welcome.do	100	30	15	94	0.00%	4.0/sec	6.48	1.63
/struts2-mailreader-2.0.6/Login_input.do	50	26	15	63	0.00%	2.0/sec	3.95	1.96
/struts2-mailreader-2.0.6/Login.do	50	7	0	16	0.00%	2.0/sec	0.00	0.00
/struts2-mailreader-2.0.6/MainMenu.do	100	5	0	62	0.00%	4.0/sec	2.81	0.70
/struts2-mailreader-2.0.6/Registration_input.do	150	45	31	234	0.00%	6.0/sec	24.05	4.01
/struts2-mailreader-2.0.6/Subscription_input.do	50	70	46	297	0.00%	2.0/sec	10.99	5.44
/struts2-mailreader-2.0.6/Subscription_save.do	100	17	15	47	0.00%	4.0/sec	0.00	0.00
/struts2-mailreader-2.0.6/Subscription_delete.do	50	71	62	172	0.00%	2.0/sec	10.98	5.43
/struts2-mailreader-2.0.6/Registration_save.do	50	52	46	78	0.00%	2.0/sec	8.28	4.09
/struts2-mailreader-2.0.6/Logout.do	50	6	0	16	0.00%	2.0/sec	0.00	0.00
TOTAL	750	31	0	297	0.00%	29.7/sec	66.63	2.24

Table 3.1: Single user load-test: Summary report

The results show that a request takes 31ms in average and that approximately 30 requests can be processed in a second. As expected, it takes longer to generate a large amount of dynamic content, as it is the case when using forms: All pages including forms takes 45–71ms to process, whereas other pages only takes 5–7ms. Also, the Login Action is generated twice as fast as the Subscription or Registration Action. This clearly shows that, the more form elements are contained in an Action, the slower the page is processed. In any case, these values are extremely low considered the fact, that a user won't notice any delay under 200ms.

The aggregate graph data shown in Figure 3.10, is interesting in consideration of business requirements, as it provides supplement results, such as the 90% Line. The latter expresses the processing time that can be expected for 90% of the requests. In the load-test, the 90% Line has a value of 62ms. This value is of interest for service level agreements (SLA) contracts. For instance, if 11% exceeds the 63ms for a given month, the customer gets a discount.

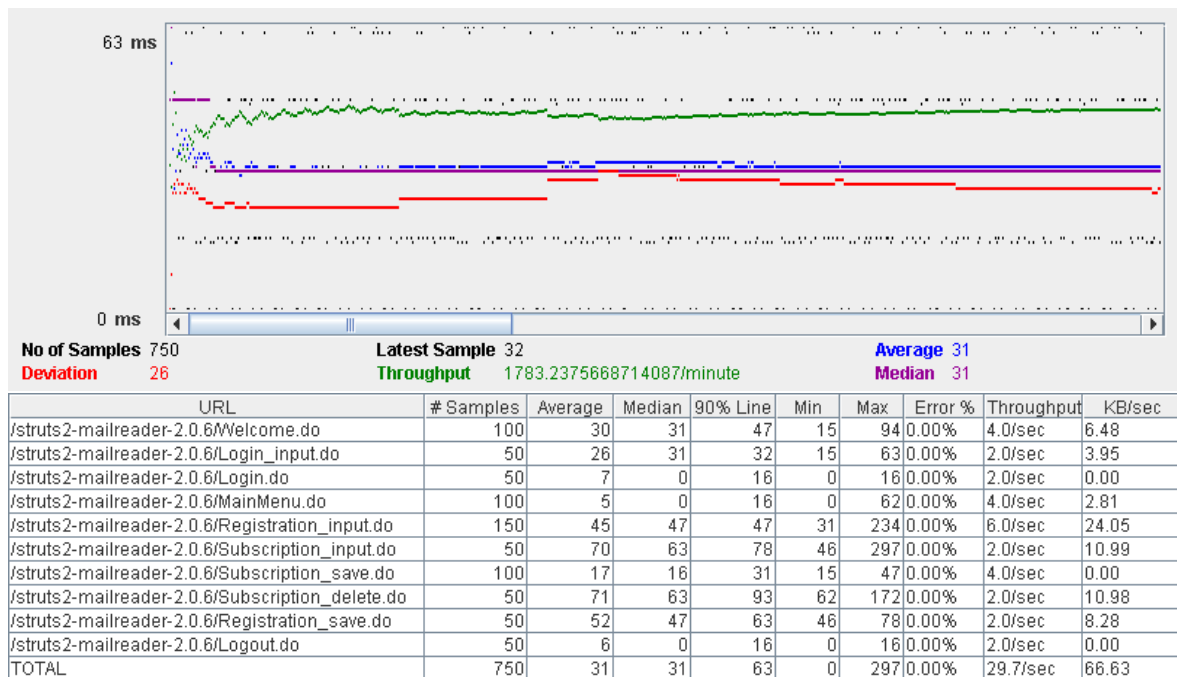


Figure 3.10: Single user load-test: Aggregate Graph and Graph Data

Another data provided by the aggregate graph is the standard deviation, defined as the square root of the total of the deviation of each sample⁵ from the average. Figure 3.10 shows a deviation of 26 for a total of 750 samples. This is a relatively small number, meaning that

⁵In JMeter, the term 'sample' is used for a request

the web application is stable. A high deviation, would imply that the response time will vary for each user, meaning that some users get a response very quickly, whereas other users need to wait for a longer time.

The aggregate graph further shows the samples, expressed in the black colour. Each black dot represent a sample. It can easily be observed, that the time to process each request is constant, as the dots form a horizontal line. This result is not surprising, because only a single user is connected to the application for now.

Multi-User Load-Test Results

In a second step, the web application was load-tested with several simultaneous users. Two test cases were considered during the evaluation:

1. Load-test: To find the maximum number of users, all by maintaining acceptable response times.
2. Test destructively: To find out the hard limit of the application.

For the first case, the results showed that the maximum of simultaneous users is 45 to 50, considered the constraint, that the application should have an acceptable response time. This result was obtained by letting a certain amount of users n do the scenario twice, with 2 seconds between each request and a ramp-up period of 10 seconds. The ramp-up period is the time in which each of the n users start the scenario. In other words, each user starts the scenario every $\frac{10}{n}$ seconds. The tests were run several times, varying the total number of users, with regard to not have an average response time over 100ms. The test results below were obtained with 47 users.

Table 3.2 shows the results obtained for each request. The pages including a large form, still have an acceptable response time of 178–280ms. Smaller pages have a response time below 41ms.

Label	# Samples	Average	Min	Max	Error %	Throughput	KB/sec	Avg. Bytes
/struts2-mailreader-2.0.6/Welcme.do	180	41	15	172	0.00%	2.5/sec	4.14	1.63
/struts2-mailreader-2.0.6/Login_input.do	90	33	15	218	0.00%	2.2/sec	4.28	1.96
/struts2-mailreader-2.0.6/Login.do	90	6	0	78	0.00%	2.2/sec	0.00	0.00
/struts2-mailreader-2.0.6/MainMenu.do	180	4	0	79	0.00%	3.0/sec	2.08	0.70
/struts2-mailreader-2.0.6/Registration_input.do	270	186	31	906	0.00%	4.9/sec	73.90	14.96
/struts2-mailreader-2.0.6/Subscription_input.do	90	178	46	578	0.00%	2.1/sec	11.66	5.44
/struts2-mailreader-2.0.6/Subscription_save.do	180	51	0	266	7.22%	3.7/sec	0.26	0.07
/struts2-mailreader-2.0.6/Subscription_delete.do	90	280	47	844	0.00%	2.1/sec	11.54	5.43
/struts2-mailreader-2.0.6/Registration_save.do	90	197	31	1000	0.00%	2.1/sec	21.25	10.01
/struts2-mailreader-2.0.6/Logout.do	90	2	0	32	0.00%	2.1/sec	0.00	0.00
TOTAL	1350	96	0	1000	0.96%	19.0/sec	91.96	4.84

Table 3.2: Multi-user load-test: Summary report

The aggregate graph shown in Figure 3.11, clearly illustrate what happened during the ramp-up period. The graphs for deviation and average response time increase very slowly until a certain point, when they increase sharply. At that point, the application had to save and delete user data from a database implemented as XML file. It is very possible that the XML file is the bottle neck of the application, since every request needs exclusive access on the file, thereby slowing down requests which must wait for the file lock to be released before they can access it. Furthermore the only failing requests are those handled by the Subscription_save.do Action, an Action which appends a large amount of data to the XML file. However, the fails were only noticeable to the user, but had no influence on the consistency of the XML file.

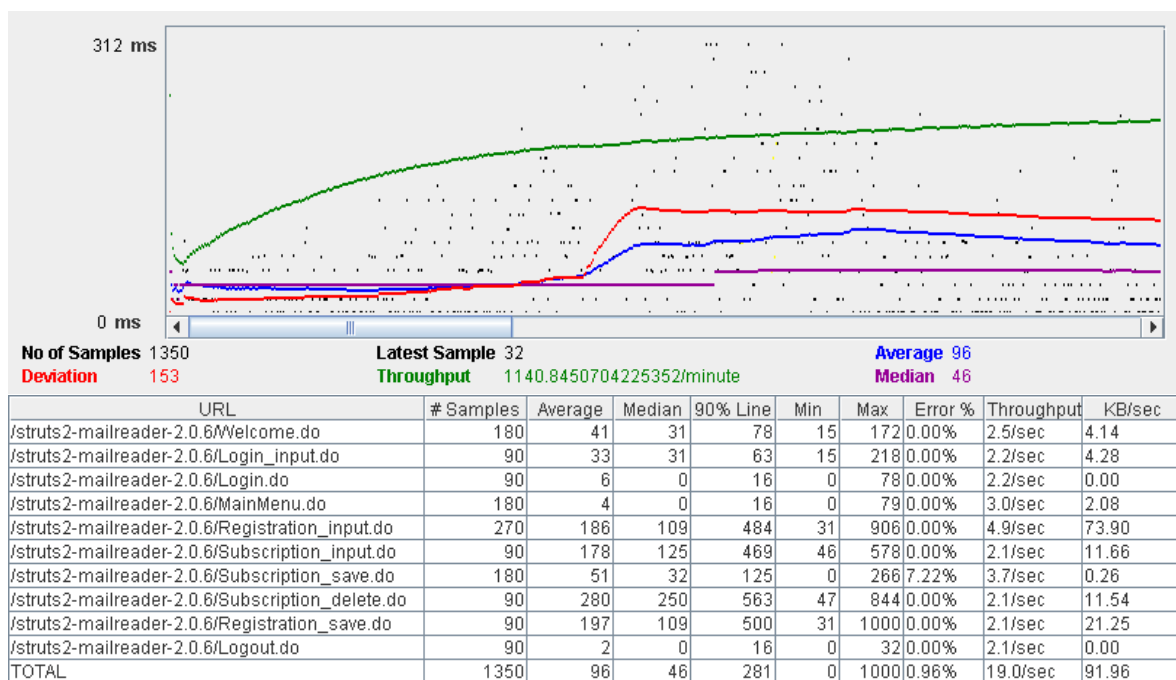


Figure 3.11: Multi-user load-test: Aggregate Graph and Graph Results

A second step was to test how much users the application could handle (test destructively). Figure 3.12 shows the graph results of the test with a total of 200 simultaneous users, which was the limit of users the test client system could handle. Surprisingly, the application handles all requests, even if the response time is very long, with values of 17 seconds.

Summary of load-tests

The test results are only of interest if it is clear, what can be spent on hardware or which infrastructure is already available. Furthermore it must be evaluated what the anticipated

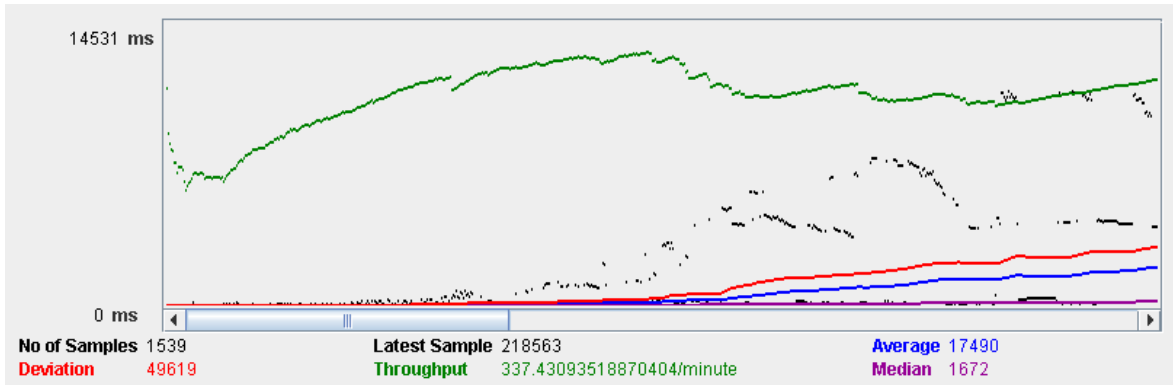


Figure 3.12: Multi-user test destructively: Graph Results

average and peak number of users will be. These questions are normally answered in the application requirements and in a SLA.

In the project's load-tests a mid-range PC was used with a 2.01GHz processor and 2GB of memory with 512 MB allocated to JBoss. Usually, dedicated servers can offer much more performance, thereby allowing more simultaneous users. Overall, the results showed that Struts 2.0 can handle an acceptable amount of simultaneous requests, which will be sufficient for most companies anticipating an average of 50 simultaneous users on relatively cheap server. A destructive test showed that more users are possible with more performance and a better persistence layer.

3.2.11. Flexibility & Extensibility

Being able to extend the framework and adapting it to the needs of an application is very important. The framework should handle most of the general work which an average application require, but then again allow for customisation. Many features of Struts 2.0 make it a very flexible and extensible framework.

Struts 2.0 took care of making as much interfaces as possible available to the developers, in order to extend the framework with additional functionality. Some of the interfaces or abstract classes include amongst others the `Interceptor` or `AbstractInterceptor`, to implement an own `Interceptor`; `Preparable`, whose `prepare()` method is called before the Action is executed; `StrutsTypeConverter`, to add further type conversion; or `Validateable`, to add custom validation. Interfaces are a good practice in software development, because they hide the implementation from the calling class. Contrary to interfaces, abstract classes can have an implementation, but require that the developer implements some of the abstract methods. The amount of available interfaces and abstract classes shows that it is

possible to implement many aspects of the framework, which in turn makes the latter very extensible and flexible.

Furthermore the interceptor concept allows for great extensibility, as described in Section 3.2.1, The Framework Architecture. Struts 2.0 also proposes different types of results (PDF, XSL, XHTML, etc.), and the possibility to add own results.

3.2.12. Revolutionising Features

Struts 2.0 is built on many years of JEE best practices and experience. Beside being the result of the best and improved features available, Struts 2.0 innovates with new concepts in the JEE world.

QuickStart

QuickStart is one of these new concepts. One of the reason for the Ruby on Rail (RoR) hype, is the ability to quickly develop an application. Inspired by the advantages of RoR and Appfure, QuickStart permits to test and develop a JEE application in real-time, without the need of long built and deployment processes. The application can be developed directly in a Jetty server, which automatically compiles sources files on the fly. According to Lightbody (2007), the installation of QuickStart is a very easy 3 step procedure.

Continuations

Continuations is a feature which is only available in very few frameworks, such as Cocoon. It is a native feature in some languages, but not in Java. Therefore, Struts 2.0 emulates the continuation behaviour. To explain the principle of continuations, Listing 3.12 shows a simple GuessAction, which implements the classical number guessing game.

Listing 3.12: Continuations demonstrated by the GuessAction

```
1 int answer = 45;
2
3 while (answer != guess && tries > 0) {
4     pause(SUCCESS);
5     if (guess > answer) {
6         addFieldError("guess was too high!");
7     } else if (guess < answer) {
8         addFieldError("guess was too low!");
9     }
10    tries--;
11 }
```

In the guessing game, the user has to guess a number. The application helps the user, by telling her, if she guessed to high or too low. The special part in this programme is the `pause()` function (see line 4). It renders the result page configured for the string `SUCCESS` and waits for the user to input a new number. When the new number is submitted, the programme is resumed (at line 5). According to Lightbody (2007), this feature is not production ready yet, but should demonstrate what will be possible in future Struts releases.

3.2.13. Summary of Struts' Evaluation

This Chapter covered the most important criteria, which makes a good framework. First of all, Struts 2.0 uses an MVC architecture, which proved to work very well since it was introduced. The framework is further split up in three key components: Action Results and Interceptors. Each of these components are very powerful assets to the framework, as each cope with different concerns, thereby allowing better distribution of tasks within a team, and easier maintenance. Furthermore, Struts 2.0 comes with automatic type conversion; a powerful validation system; great support for internationalisation; a Struts 2.0 tag library, which uses the ingenious value stack; great support for testing environment; a large community of developers and users; a performance, which did well during load-tests; a default framework configuration, which should satisfy most application's needs, but which is also very flexible and extensible; and finally, some new revolutionising JEE features, such as QuickStart or Continuations. The frameworks limits, however, are the lack of tools, which may soon be available, once the framework will be more popular.

Chapter 4.

Conclusion

The aim of this research project was to evaluate Maven 2 and Struts 2.0, since these are two very recent technologies used to create web applications. Special care has been taken to analyse specific aspects of each technology in order to cover every important criteria for a effective project management tool or framework accordingly.

Maven 2 is without any doubt the best tool available so far to handle the lifecycle of JEE projects. It removes the tedious work that a developer must perform with other build tools such as Ant. This is only made possible through a consequent standard project layout and a standardised dependency-system. All tasks required by a JEE project are implemented by default and can, therefore, be run out of the box without any further configuration. Consequently, it is no longer necessary to create and maintain large configuration files and projects can be developed faster than ever.

As for Struts 2.0, it proved to successfully meet all requirements of a modern web application framework. It is built on best practices which are founded on years of experience. Previous users will enjoy the fact that Struts 2.0 is still Action driven and built around the MVC pattern, but they will further benefit from new features such as the Interceptor concept, better automation of data binding, a much simpler validation system, and an improved struts tag library. Beside offering improved features, Struts 2.0 also innovates by introducing annotations, which increase the speed of development. Finally, it can be noticed that Struts is actively working on improvements for the new framework and has plans for the future, for instance with the new concept of continuations.

Summing up, Struts 2.0 alleviates the web development work allowing a faster production delivery and better maintenance. The framework provides a flexible and extensible application architecture with labour-saving Struts 2.0 tags and many ways to configure common workflows within an application. For all these reasons, Struts 2.0 should most certainly be considered as a framework for new development projects.

Nevertheless, the results of this research project also showed that Struts 2.0 is not a product for everybody. First of all, the load-tests indicated that great performance is required to handle over 50 simultaneous users. If the customer expects more users, expensive hardware is required. Furthermore the application needs to have Javascript enabled, as Struts 2.0 uses it frequently, especially for forms. If one of the customers requirement is to allow users with Javascript being disabled, Struts 2.0 is probably not the right framework to choose. For any other customer requiring a large-scale enterprise application, Struts 2.0 is without doubts the right choice.

Outlook

One of Struts' weak points, is the lack of supported tools to develop an application based on the framework. Of course, annotations and XDoclet can be used to generate the file descriptors, but it could be a great advantage to have an IDE integration for the most important tasks. Struts 2.0 took advantage of several concepts, allowing to maintain many concerns in many places. A GUI could increase usability and better maintenance by helping to manage all these different resources.

However, Struts 2.0 is only at its beginning and seems to already have found a large community of supporting developers. Only the future will tell if Struts can maintain its strong position as a community with one of the most used frameworks in JEE development.

Appendix A.

The JMeter Test Plan

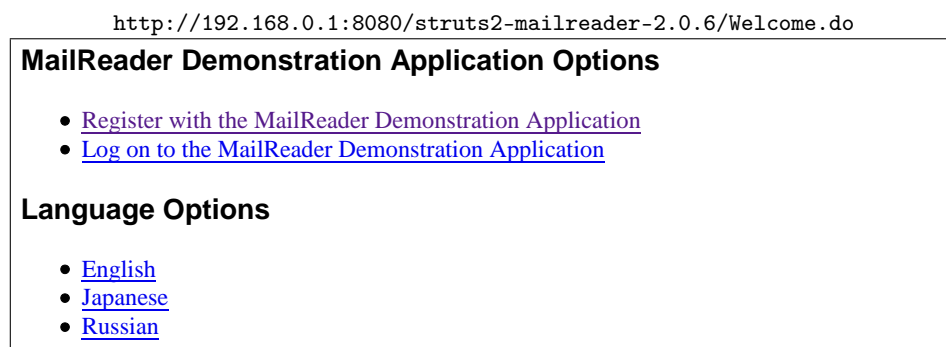


Figure A.1: Screenshot of the Welcome screen

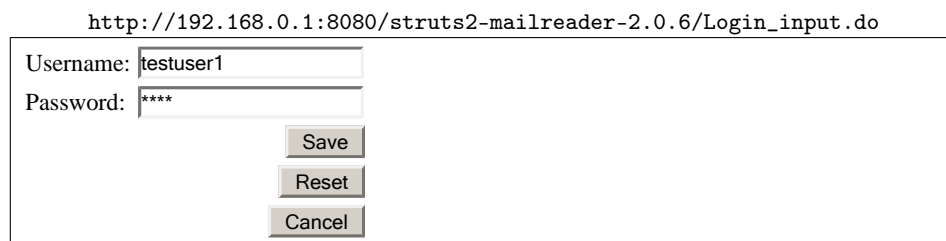


Figure A.2: Screenshot of Login screen

http://192.168.0.1:8080/struts2-mailreader-2.0.6/MainMenu.do

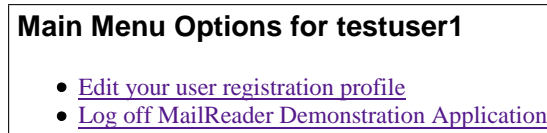


Figure A.3: Screenshot of the main menu

http://192.168.0.1:8080/struts2-mailreader-2.0.6/Registration_input.do

Username: testuser1
Password:
(Repeat) Password:
Full Name: testuser1
From Address: testusers@testers.com
Reply To Address: testusers@testers.com

Current Subscriptions

Host Name	User Name	Server Type	Auto	Action
Add				

Figure A.4: Screenshot of the registration screen, showing the subscriptions

http://192.168.0.1:8080/struts2-mailreader-2.0.6/Subscription_input.do

Username: testuser1
Mail Server: mail.yahoo.com
Mail Username: testuser1
Mail Password: hons
Server Type: POP3 Protocol ▾
 Auto Connect

Figure A.5: Screenshot of the screen to add a new subscription

http://192.168.0.1:8080/struts2-mailreader-2.0.6/Registration_input.do

Username: testuser1
Password:
(Repeat) Password:
Full Name: testuser1
From Address: testusers@testers.com
Reply To Address: testusers@testers.com

Current Subscriptions

Host Name	User Name	Server Type	Auto	Action
mail.yahoo.com	testuser1	pop3	false	Delete Edit

[Add](#)

Figure A.6: Screenshot of registration screen with a new subscription

http://192.168.0.1:8080/struts2-mailreader-2.0.6/Subscription_delete.do?host=mail.yahoo.com

Username: testuser1
Mail Server: mail.yahoo.com
Mail Username: testuser1
Mail Password: hons
Server Type: pop3
Auto Connect: false

Figure A.7: Screenshot of the screen to delete a subscription

http://192.168.0.1:8080/struts2-mailreader-2.0.6/Registration_save.do

Username: testuser1
Password:
(Repeat) Password:

Full Name is required
Full Name:

From Address is required
From Address:
Reply To Address:

Current Subscriptions

Host Name	User Name	Server Type	Auto	Action
-----------	-----------	-------------	------	--------

[Add](#)

Figure A.8: Screenshot of a submitted form where validation failed.

Glossary

Notation	Description
AJAX	Asynchronous JavaScript and XML: <i>AJAX is used to only refresh specific parts of a website instead of reloading the entire page.</i>
ASF	Apache Software Foundation: <i>The Apache Software Foundation is a non-profit corporation, operating many different web related projects and a wide community of members.</i>
EAR	Enterprise ARchive: <i>An EAR file packages one or more WARs.</i>
HTTP	Hyper Text Transfer Protocol: <i>HTTP is amongst others the protocol used to browse the World Wide Web.</i>
IRC	Internal Relay Chat: <i>HTTP is amongst others the protocol used to browse the World Wide Web.</i>
J2EE	Java 2 Platform, Enterprise Edition: <i>J2EE is a version of Java for developing and deploying enterprise applications.</i>
JAR	Java ARchive: <i>JAR is a file format used to package Java applications</i>
JDK	Java Development Kit: <i>The Java Development Kit is a collection of developer tools for Java developers provided by Sun Microsystems.</i>
JEE	Java Platform, Enterprise Edition: <i>Formerly known as J2EE up to version 1.4, the term JEE is now used for web application running on version 5 of the Java Platform.</i>

Notation	Description
JRE	Java Runtime Environment: <i>The Java Runtime Environment is a collection of libraries and other components provided by Sun Microsystems that allows a computer system to run applets and applications written in the Java programming language.</i>
JSF	Java Server Faces: <i>JSF is a Java-based web application framework.</i>
JSP	Java Server Pages: <i>JavaServer Pages is a server side scripting language developed by Sun Microsystems that is used by Java developers to dynamically generate web pages.</i>
MVC	Model View Controller: <i>A paradigm, stating that the data (model) should be separated from the user interface (view) and the processing (controller).</i>
POM	Project Object Model: <i>The POM is an XML file, which holds the entire configuration for Maven projects.</i>
URL	Uniform Resource Locator: <i>A URL is the unique address for a file that is accessible on the Internet.</i>
W3C	The World Wide Web Consortium: <i>The W3C defines the specifications and guidelines for internet standards.</i>
WAF	Web Application Framework: <i>A WAF provides various functionalities required to build a web application.</i>
WAR	Web ARchive: <i>A WAR file is a JAR used to deploy a collection of resources needed for a Java web application.</i>
XHTML	eXtensible Hypter Text Markup Language: <i>XHTML is a text-based markup language written in XML for the creation of web pages. It has been developed by the W3C as the successor of HTML.</i>
XML	eXtensible Markup Language: <i>XML is a general-purpose markup language for creating special-purpose markup languages, and is used to describe different kinds of data.</i>

Bibliography

- Apache. 2007a. "Apache JMeter.". Available at: <http://jakarta.apache.org/jmeter/>.
- Apache. 2007b. "Mailing list archives: users@maven.apache.org.". Available at: http://mail-archives.apache.org/mod_mbox/maven-users/.
- Apache. 2007c. "Mailing list archives: user@struts.apache.org.". Available at: http://mail-archives.apache.org/mod_mbox/struts-user/.
- Bolcina, Borut. 2007a. "[m2eclipse-user] Won't compile because of transitive dependency not finding its artifact.". Available at: <http://www.mail-archive.com/user@m2eclipse.codehaus.org/msg00376.html>.
- Bolcina, Borut. 2007b. "Re: [m2eclipse-user] Won't compile because of transitive dependency not finding its artifact.". Available at: <http://www.mail-archive.com/user@m2eclipse.codehaus.org/msg00378.html>.
- Brown, Don, Ted Husted, Dave Newton, Musachy Barroso and Philip Luppens. 2007. "Interceptors.". Available at: <http://struts.apache.org/2.0.6/docs/interceptors.html>.
- Burns, Ed, Jason Carreira, Howard M. Lewis Ship David Geary, Jonathan Lock and Kevin Osborn. 2005. Web Framework Smackdown. In *JavaOne '05: Session 7642*.
- Codehaus. 2007. "The Maven Integration for Eclipse.". Available at: <http://m2eclipse.codehaus.org>.
- Eclipse. 2007. "Eclipse.org home.". Available at: <http://www.eclipse.org>.
- Fanelli, Tim. 2005. "Getting Started with Struts Shale.". Available at: <http://www.timfanelli.com/item/134>.
- Freenode. 2007. "About the Network.". Available at: <http://freenode.net/>.
- Getahead. 2007. "DWR - Easy AJAX for JAVA.". Available at: <http://getahead.org/dwr/>.
- Hermanns, Rainer, Ted Husted and Don Brown. 2006. "Validation Annotation.". Available at: <http://struts.apache.org/2.0.6/docs/validation-annotation.html>.

- Hibernate. 2004. "Hibernate Aware Action.". Available at:
<http://www.hibernate.org/51.html>.
- Husted, Ted. 2006a. "Building Your Own Interceptor.". Available at:
<http://struts.apache.org/2.x/docs/building-your-own-interceptor.html>.
- Husted, Ted. 2006b. "Token Interceptor.". Available at:
<http://struts.apache.org/2.0.6/docs/token-interceptor.html>.
- Husted, Ted. 2006c. "Token Session Interceptor.". Available at:
<http://struts.apache.org/2.0.6/docs/token-session-interceptor.html>.
- Husted, Ted. 2007. "Apache Struts 2 from Square One (training course)". Available at:
<http://code.google.com/p/sql-struts2/>.
- JBoss. 2007a. "Apache Reference: mod_proxy.". Available at:
<http://labs.jboss.com/portal/jbossas/download/>.
- JBoss. 2007b. "Securing JBoss." *Jboss Wiki* . Available at:
<http://wiki.jboss.org/wiki/Wiki.jsp?page=SecureJBoss>.
- Koke, Justin. 2007. "From manual to automatic.". Available at:
http://blogs.atlassian.com/developer/2007/03/from_manual_to_automatic.html.
- Lanquetin, Nicolas. 2007. "Research Proposal for Future JEE Technologies.". Available at:
http://psbase.com/studies/uad/wdd/ca1068a_research_proposal/.
- Laurie, Ben and Chuck Murcko. 1996. "Apache Reference: mod_proxy.". Available at:
http://www.apacheref.com/ref/mod_proxy.html.
- Lightbody, Patrick. 2007. "WebWork (Struts 2) In Action.". Available at:
<http://www.infoq.com/presentations/struts-2-webwork-pat-lightbody>.
- Lightbody, Patrick, Rainer Hermanns, Philip Luppens, Ted Husted and Musachy Barroso. 2007. "Type Conversion.". Available at:
<http://struts.apache.org/2.0.6/docs/type-conversion.html>.
- Lightbody, Patrick, Rene Gielen, Philip Luppens, Don Brown, Ted Husted and Musachy Barroso. 2007. "Validation.". Available at:
<http://struts.apache.org/2.0.6/docs/validation.html>.
- Lightbody, Patrick and Ted Husted. 2006. "Client Side Validation.". Available at:
<http://struts.apache.org/2.0.6/docs/client-side-validation.html>.
- Lightbody, Patrick, Ted Husted and Philip Luppens. 2006. "Result Types.". Available at:
<http://struts.apache.org/2.0.6/docs/result-types.html>.

- Mann, Kito D. 2005. From Struts to JavaServer Faces: Evolving Your Web Applications to Support the New Standard. In *JavaOne '05: Session 7642*.
- Massol, Vincent, Jason va Zyl, Brett Porter, John Casey and Carlos Sanchez. 2006. *Better Builds with Maven*. Mergere Inc.
- Maven. 2007a. "How do I include tools.jar in my dependencies?". Available at: <http://maven.apache.org/general.html#tools-jar-dependency>.
- Maven. 2007b. "Maven Documentation.". Available at: <http://maven.apache.org/guides/index.html>.
- McClanahan, Craig. 2005a. "The Best of Both Worlds: Integrating JSF with Struts in Your J2EE Applications." *Oracle Technology Network* . Available at: http://www.oracle.com/technology/pub/articles/masterj2ee/j2ee_wk8.html.
- McClanahan, Craig. 2005b. Shale: The Next Struts?? In *ApacheCon US '05: Session TU24*.
- McCuaig, Pann. 2003. "Debian Packaging System.". Available at: <http://www.chuug.org/talks/20030325/>.
- McLaughlin, Brett. 2006. "All Hail Shale: Shale isn't Struts." *IBM developerWorks* . Available at: <http://www.theserverside.com/talks/library.tss#mcclanahan2>.
- Shah, Gautam. 2006. "Apache Shale Takes JavaServer Faces to the Next Level." *devX* . Available at: <http://www.devx.com/Java/Article/31419/>.
- Sipe, Ken. 2005a. Struts Shale... I Mean Struts. In *DevCon '05*. Available at: <http://bdn1.borland.com/devcon05/article/1,2006,33222,00.html>.
- Sipe, Ken. 2005b. Web Application Development using Struts, Shale, and JSF. In *DevCon '05*. Available at: <http://bdn1.borland.com/devcon05/article/1,2006,33218,00.html>.
- Struts. 2007a. "Apache Struts 2 Documentation.". Available at: <http://struts.apache.org/2.0.6/docs/home.html>.
- Struts. 2007b. "MailReader Demonstration Application.". Available at: <http://www.planetstruts.org/struts2-mailreader/>.
- Struts. 2007c. "Struts 2.0.6 Distributions.". Available at: <http://struts.apache.org/download.cgi#struts206>.
- Sun. 2007. "Java BluePrints: Model-View-Controller.". Available at: <http://java.sun.com/blueprints/patterns/MVC-detailed.html>.
- Tigris. 2007. "Subclipse Eclipse Plugin.". Available at: <http://subclipse.tigris.org/>.

Bibliography

UCBerkeley. 2006. "Connecting Apache 2.0.## to JBoss-Tomcat via mod_jk2.". Available at: http://sis36.berkeley.edu/projects/streek/howto/apache-mod_jk2-win.html.

Vermeulen, Sven, Grant Goodyear, Roy Marples, Daniel Robbins, Chris Houser and Jerry Alexandratos. 2007. "Portage Introduction.". Available at: <http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?part=2&chap=1>.

Webwork. 2007. "Mailing list archives: users@webwork.dev.java.net.". Available at: <https://webwork.dev.java.net/servlets/SummarizeList?listName=users>.

Index

- #maven, 27
- #struts, 27
- Methods
 - prepare(), 33
- Abstract classes
 - AbstractInterceptor, 14
 - StrutsTypeConverter, 17
- Action, 11–13
- Annotations, 19–20
- Ant, *see* Apache Ant
- Apache Ant, 5
- Apache Tomcat, 3
- Application server, 3–4
- Classes
 - ExecuteAndWaitInterceptor, 13
- Conversion, *see* Data Binding
- Data Binding, 16–17
- Direct Web Remoting, 20
- Double Submits, 20–21
- DWR, *see* Direct Web Remoting
- EAR, *see* Enterprise Archive
- Enterprise Archive, 5
- Files
 - hp.war, 9
 - pom.xml, 7
 - struts.xml, 14
 - validation.xml, 18–19
- freenode, 27
- i18n, *see* Internationalisation
- Interceptor, 13–15
- Interfaces
 - Interceptor, 14
 - Preparable, 33
 - Validateable, 18
- Internal Relay Chat, 27
- Internationalisation, 21–22
- IRC, *see* Internal Relay Chat
- Java Server Pages, 11
- JavaBean, 16–17
- JBoss, 4
- JSP, *see* Java Server Pages
- JUnit, 23–24
- m2eclipse, 27–28
- Maven 2
 - goals, 8–9
- Methods
 - execute(), 12–13
 - validate(), 18
- Model 2, 11
- Model-View-Controller, 11
- MVC, *see* Model-View-Controller
- POM, *see* Project Object Model

Project Object Model, 7

Results, 13

Servlet, 11

Servlet container, 3–4

Struts token, 20–21

theme, 22

Tomcat, *see* Apache Tomcat

Type Conversion, *see* Data Binding

Validation, 17–20

Value Stack, 15–16

WAR, *see* Web Archive

Web Archive, 5

XDoclet, 20, 28