Tallinna Pedagoogikaülikool
Matemaatika-loodusteaduskond
Matemaatika-informaatika osakond

Mikk Normak

OD Struts in Three - Tier Web Applications

Juhendaja: Pasi Salminen

Tallinn 2002

Table of Contents

1. Introduction

The Internet has changed the world. The ability to be anywhere and do anything without leaving the confines of your home has revolutionized the business world. The success or failure of a company is often decided by its ability to attract clientele from the web. Companies want to transfer more and more of their functionality into the web, because of the maintainability, and savings from excluding the human factor from client interaction process. This requires a transition from the traditional static HTML-based environments to a dynamic solution that could actively interact with the client in a fast, supportive and easily maintainable fashion.

In the competitive business world time-to-market is the keyword for the success of any business. A business without the ability to rapidly deploy new ideas and stay ahead of the competition is ultimately doomed.

With websites becoming larger and more complex another problem looms on the horizon. How to maintain and when necessary, update this vast network of pages, code and configuration files in a cheap and effective manner and where to find the human resources with all the skills necessary.

Struts is a framework that targets these key points. Using Struts as the basis for web development increases the speed, at which the product will be completed, allows the usage of specialists that must not necessarily be proficient with all the aspects of web development and keeps the maintenance effort and therefore cost less than traditional web development methods. The skills needed by people working with Struts are easily found in today's market. Knowledge of Java, HTML and JSP is widespread, but using Struts lowers the qualification bar even more. No single person in a development team must have all the skills listed above.

Struts is created by Craig R. McClanahan, who on Mai 200 donated it to the Apache Software Foundation. Since then McClanhan has remained as the leader of the Struts project even though he is actively participating in other Apache projects.

2. Struts-related technologies

2.1. Java

Struts is a development platform for web systems developed in Java. Struts itself is also written in Java.

Java is a general – purpose object – oriented programming language developed to be completely platform independent. This goal has almost been reached and has granted considerable success to the Java language. A Java program once compiled has the remarkable ability to work under several operating systems and computer types without any change.

Java's greatest advantage is also its greatest drawback – it is an interpreted language that is not compiled to machine code as most other programming languages do. When a Java program is run, the Java Virtual Machine reads the bytecode and translates it to the platform – dependent machine code. This solution provides the portability (also known as "write one, run anywhere"), but also suffers a performance loss compared to traditional programming languages like C. However every new Java release and the continuous enhancements in computer hardware, this performance gap is becoming ever narrower. The incredible success of the Java platform is a solid proof that the platform – independency, the wide array of included libraries and the simplicity of learning far outweigh the drawbacks.

2.2. Servlet

A technology, that allows a Java program to dynamically create a web page. The trend in web systems has been towards thin clients, where all the work gets done on a central server and clients simply display the information originating from the server.

When a request is made to a web server, it first decides based on the configuration files, which page to display to the client. If the page that is currently being requested is mapped to a servlet, then the web server runs that servlet and calls its "service( )", which in turn calls usually either “doGet()” or “doPost()” method, depending on whether the page was requested with GET (typical in links) or POST (typical in form submission). When the servlet finishes it will have created a response, for example an HTML page, into the web server’s memory, which is then served back to the client as if it were a static resource. The client (usually a web browser) receives the response (e.g. HTML page or PDF document) never knowing, that the resource that was requested did not exist before the request and handles it as any other web page.

The same sequence of actions also happens in case of a CGI (Common Gateway Interface), Perl, PHP, ASP and etc. pages. The reason, why Java, generally known as a slow language, is so widely used as a web development platform is, that when the bytecode for a servlet first gets loaded by the web server, it remains in the web server’s memory and the next time the servlet is requested, the appropriate method (“doGet ()” or “doPost ()”) gets called on the instance of the servlet already in memory.

Coupled with the ability to use already existing Java classes, servlets were the answer to creating dynamic web content in Java. However, there is a downside. A servlet must contain the HTML output within itself.

2.3. JSP – Java Server Pages

JSP is a natural evolution of servlet technology. The problem with servlets is that all HTML output must be contained in the servlet itself.

```
Out.println("<HTML>");
Out.println("<BODY>");
Out.println("Hello World");
Out.println("</BODY>");
Out.println("</HTML>");
```

As seen from the example above this method presents several difficulties to the web developer:

1. The code for a servlet becomes difficult to understand for the programmer.
2. The HTML content of such a page is difficult if not impossible for a web designer to understand or design.
3. Any changes in the HTML content require the rebuilding of the whole servlet.
4. The learning curve for a programmer to maintain or develop the web application is very steep and becomes increasingly difficult with the expansion of the application.

JSP solves these problems by giving a way to include java code into an HTML page using scriptlets. This way the HTML code remains intact and easily accessible to web designers, but the page can sill perform its task.

```
<HTML>
<BODY>
<%SimleDateFormat sdf = new SimpleDateFormat("dd.nm.yyyy");%>
The date is: <%=sdf.format( new Date() )%>
</BODY>
</HTML>
```

A JSP page is handled differently compared to a servlet by the web server. When a servlet is deployed into a web server in compiled (bytecode) form, then a JSP page is deployed in its original, human-readable form. When a user requests the specific page, the web server compiles the page into a servlet and from there on handles it as a standard servlet. This accounts for a small delay, when a JSP page is first requested, but any subsequent requests benefit from the same speed effects that are associated with servlets.

2.4. EJB – Enterprise Java Beans.

Enterprise JavaBeans (EJB) is a transaction server development technology. EJB both enables transaction processing and control transactions across multi – platform enterprise systems and database servers.

EJB is actually a specification – not a product. As a specification, it defines a way to do multi-platform transaction processing in Java. However, EJB is implemented as a product as the center piece of Sun's J2EE (Java 2 Enterprise Edition).

The basic idea behind EJB is distributed networking that benefits from Java's cross – platform portability. An Enterprise JavaBean is a piece of Java code usually performing a single business logic function. It can be deployed in an EJB server (also known as "container") that manages the deployment, execution and maintenance of all EJB-s in it. Distributed networking means the ability to harness the processing power of several computers especially for high user – load environments. An EJB server has the ability to forward requests coming to it to another server containing the same EJB instead. This forwarding decisions are made based on the load of both computers, a busy server forwards the requests to an idle server or servers thereby distributing the computer load.

EJB has become the proffered way of deploying Java business logic for client – server applications.

3. Struts and the MVC solution

When JSP first appeared it was rightfully hailed as a revolution in Java web programming and the development pattern, where JSP replaces servlets became known as “Model 1”. However it was soon realized that placing the whole business logic of an application into JSP files is cumbersome and defeats the advantage of making JSP-s easily accessible and editable.

To solve that problem the “Model 2” solution or the MVC solution was crated. The basic idea behind the MVC solution is to separate the presentation part that the customer interacts with from the business logic part of the application. To do this the application must be divided into three distinct parts:

- View - presentation part, that the customer interacts with
- Model - business logic behind the application
- Controller - the central framework holding the Model and the View together

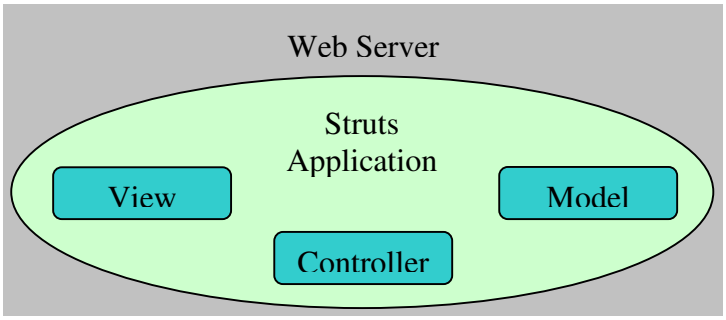

Figure 1. MVC

These parts perform different tasks and also physically apart (in different files). This allows the Model and the View parts of the application to be developed separately by people with different skills. For example the person creating JSP and HTML pages for an application does not need to have any Java knowledge, since his task is separate from a Java programmer’s task of creating business logic.

The Model and the View are linked by the Controller, a central servlet that manages the flow of control throughout a Struts application. This means that neither the Model nor the View are hard linked to each other, but use abstract names to reference to each other and the Controller deciphers the abstract names based on its configuration.

This makes the components of a Struts application highly reusable, since neither the Model nor the View have to be rewritten and reconfiguring the Controller will suffice.

Since the control flow of the whole application is visible from the configuration of the Controller, there is always a clear overview of the entire system. Using Struts considerably reduces the cost in time and effort to maintain or modify an existing system.

4. “Hello World” in Struts

Before discussing the operation of Struts components in detail, lets look at a sample application – “Hello World”. Locate the “HelloWorld.war” file alongside this document. This sample application consists of 9 files:

HelloWorld.html .....the View

WEB-INF folder :

app.xml.....Controller configuration

web.xml.....web server configuration

lib\od-struts.jar .....the Controller and Struts-related files

lib\od-core.jar .....other necessary files for Struts

tlds\od-struts.tld.....tag library for Struts-specific tags

classes\strutsDoc\HelloWorld folder:

HelloWorld.java .....the Model

HelloWorld.class .....HelloWorld.java compiled

This looks confusing, but most of these files are standard and are never changed. Only the following files are application-specific and were created for the “Hello World” application.

- HelloWorld.html
- app.xml
- HelloWorld.java

4.1. How to run “Hello World”

A Struts application must be deployed inside a web server in order to run it. The web server must support Servlets and JSP. For demonstration purposes the web server, that the sample applications are developed for is Apache Tomcat, because Tomcat is freeware, is only 4.2 MB in size, takes less than 5 minutes to install and it takes only seconds to deploy the sample applications under Tomcat. The latest version of Tomcat

can be downloaded from http://tomcat.apache.org/. Sample applications were developed and tested in Apache Tomcat 4.0, but they should also run under newer versions of Tomcat.

Follow these steps to run the "Hello World" application:

1. Download Tomcat
2. Install Tomcat
3. Copy "HelloWorld.war" to the "webapps" folder, in the Tomcat installation folder
4. Run tomcat (Start Menu/Programs/Apache Tomact 4.0/Start Tomcat)
5. Open the URL http://localhost:8080/HelloWorld/HelloWorld.do in a web browser

5. Control flow in Struts

The best way to understand Struts is to see, how a simple application functions. “HelloWorld” is the simplest Struts application possible. It has one Model and one View component and the necessary files to configure and run Struts Controller binding them together. The Model does not do anything and the View is a simple HTML page displaying “Hello World” in the browser.

The web server, containing a Struts application, is set up to listen to HTTP requests arriving at a specific port of the server. In case of the “Hello World”, Tomcat’s default setup is to listen to port 8080.

The client contacts the web server (localhost at port 8080) and requests a web page (HelloWorld/HelloWorld.do). This translates to “HelloWorld.do” page from the “HelloWorld” application.

Tomcat examines the web.xml file at startup to determine the configuration of each deployed application. The lines:

```
(WEB-INF\web.xml)
<servlet>
        <servlet-name>
                action
        </servlet-name>
        <servlet-class>
                com.oncedone.struts.action.ActionServlet
        </servlet-class>
        <init-param>
                <param-name>config</param-name>
                <param-value>/WEB-INF/app.xml</param-value>
        </init-param>
</servlet>

<servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

tell Tomcat to send all requests to all “*.do” pages to servlet “action” which is implemented by “com.oncedone.struts.action.ActionServlet”, which with its supporting

classes makes up the Controller. The location and name of the Struts configuration file is also defined within the <servlet> tag.

The request is passed to the Controller that extracts the string between the last slash and the last full stop – "…lloWorld/**HelloWorld**.do" and finds out, that the "HelloWorld" action was requested. The Struts configuration file maps the "HelloWorld" URL to the actual file performing that action.

(WEB-INF\app.xml)

```
<action-mappings>
        <action path="/HelloWorld"
                type="strutsDoc.HelloWorld.HelloWorld">
                <forward name="welcome" path="/HelloWorld.html"/>
        </action>
</action-mappings>
```

This means that any request made to the "HelloWorld" page is first routed to "strutsDoc.HelloWorld.HelloWorld" class, which represents the Model (the business logic) in this example. On this occasion the Model does nothing, but returns the control flow back to the Controller and requests a forward page "welcome"

(WEB-INF\classes\strutsDoc\HelloWorld\HelloWorld.java)

```
ActionForward forward = mapping.findForward( "welcome" );
return forward;
```

Looking back at the Controller configuration, the "welcome" page was mapped to the actual "HelloWorld.html" page. Controller forwards the request to the View component implemented by "HelloWorld.html". The view page is written to the HTTP response and the activation sequence falls back to the Controller. The Controller then forwards the page to the client.

In a client's perspective a page was requested (HelloWorld.do) and a page was returned (HelloWorld.html). In most cases the client is a web browser thus the end result of a request to a Struts application is displayed in that web browser and the client remains unaware of the Struts application that delivered the page behind the scenes. Even the returned result is not displayed as "HelloWorld.html", but instead as "HelloWorld.do", because as far as the browser is concerned, the information returned came from the "HelloWorld.do" page.

From the Struts application's perspective the following sequence of actions took place:

- The client requests the "HelloWorld.do" page
    - ActionServlet.doGet()

      The request to HelloWorld.do page was forwarded to the "ActionServlet" class and Tomcat calls the doGet() method on ActionServlet
        - HelloWorld.perform()

          This method was called by the Controller to invoke the business logic. When finished control is returned to the Controller.
        - Controller called the view

          The HTML page was loaded into the HTTP response

          When finished, View returned to the Controller
    - The response was sent back to the client
- The client displays the page

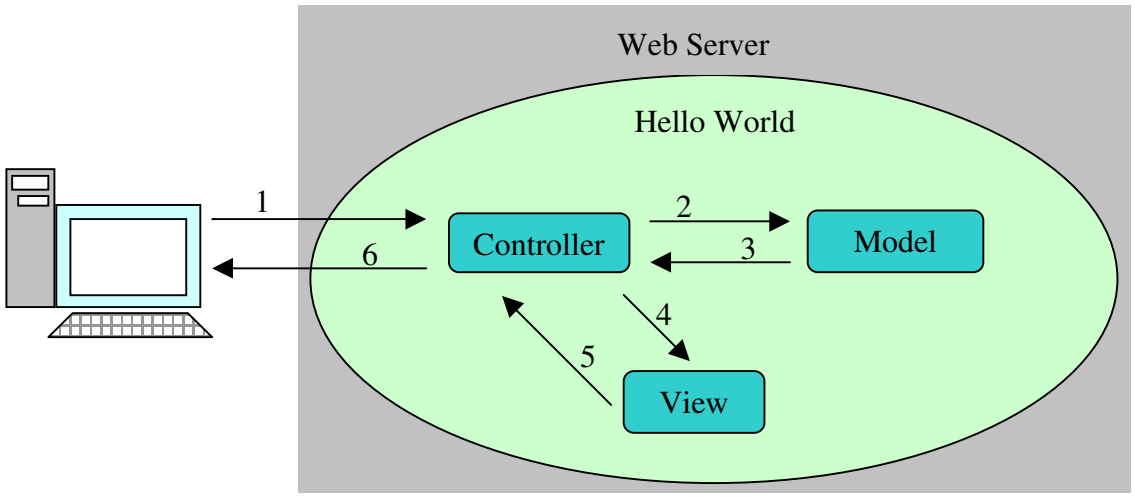

Figure 2. Struts activation order

This is a simplified view of a Struts application. In a reality much more is happening behind the scenes like Action Filters, Form Bean population, form validation, etc. This is a basic overview of Struts and necessary to understand more thorough descriptions of Struts components and on the other hand a more comprehensive description of Struts cannot be given without descending into the nuances of Struts components.

A more detailed view of activation sequence in Struts is given in the Controller description later in this document.

6. Information Flow in Struts – Form Beans

Since Struts separates the View from the Model, it must also provide means of information transfer between them that retains the ability of different Struts components to be easily reusable.

Any web page served by a Struts application is usually acquired by the following sequence: Client -> Controller -> Model -> Controller -> View -> Controller -> Client. The client is usually a web browser contacted over the Internet. This means that all the information going to a client must be in HTML format and the information coming from the client must be in an HTML form.

Between Struts components more sophisticated means can be used. In a standard application two distinct information routes can be identified. When a client submits information from an HTML form, then that information must reach the Model, that can act accordingly, and information must be passed from the Model to the View in order to pass it to the client. To solve these problems Struts uses form beans, classes that act as data holders and can be accessed by both Model and View components.

The traditional way of solving this was to bind objects to either session or request objects. When an HTML form is submitted, the web server, that is running a Java-based web application in places all the information form that form into the request as parameters and the handling servlet (or JSP page) extracts that information. However in a large-scale multi-language application than integers or strings, this approach demands quite a lot from the handling class.

When an HTML form is submitted, all the information on it is in string format and if it represents another format, for example a date, it must be converted from a string to a date. However a date can be represented differently in different countries. For example in some countries when writing a date the month is placed before the day of the month – “mm-dd-yyyy”. Hence it would be wise to centralize information extraction and storage and thereby avoid such complications. The form beans and the way they are handled in Struts provide the solution.

Simply put a form bean is a Java class that has all the information stored in it as private variables. To set and retrieve the information inside a form bean, it must provide a

getter and setter methods for each of these variables. Using these methods Struts can populate information on an HTML form, when the form is created, by matching the getVariable and setVariable methods to the name of the form field. For example:

```
<input type="text"  name="login">
```

would be matched to:

```
private String login;
public String getLogin(){
        return login;
}
public void setLogin(String newLogin){
        login = newLogin;
}
```

methods.

When an HTML form is submitted to a Struts application, then the Controller can be configured to create the specific form bean and populate it with the information coming from the client. To do this the submit address of the HTML form must be configured in Struts like this:

```
<form-beans>
        <form-bean name="LoginForm" type="forms.LoginForm"/>
</form-beans>
<action path="/LoginAction"
        type="actions.LoginAction"
        formName="LoginForm"
        formScope="session">
        <forward name="success" path="/login.html"/>
        <forward name="retry" path="/index.html"/>
</action>
```

When an HTML form is submitted to "…/LoginAction.do" page the Controller initializes the forms.LoginForm file and binds it to the session with the name "LoginForm". The Controller then calls bean population process that extracts the information sent by the user, converts it to appropriate data types and saves it into the bean. The bean is then passed to the Model that can run higher level data validation on the data in the bean. When the Controller calls the View the bean can still be easily accessed and any

information originally present in the bean or changed by the Model can be displayed to the client.

6.1. Bean Population

Bean population is the process of extracting information submitted by the user and placing it inside the form bean. This process is divided into three distinct tasks :

1. Extracting data
2. Converting from string to appropriate data types
3. Populating the bean

Since different applications might require different ways of population, Struts offers a default populator that can handle the most common situations, but an application that requires functionality beyond that supported by the default populator can create its own populator class. If no specific populator is defined, Struts uses the default populator. Populator definition is done in Controller configuration file.

```
<populators>
        <populator name="ExamplePopulator"
type="com.oncedone.struts.population.DefaultPopulator">
                <default-value parameter="login" value="login"/>
                <default-value parameter="age" value="10"/>
        </populator>
</populators>
```

When the populate() method is called on a populator, the empty form bean and the HTTP request are also passed to it. First the populator extracts the info user submitted as key - value pairs from the HTTP request. It also adds any default values given in the Controller configuration as fallback for when the specific field was left empty or was not present in the form.

The populator then iterates through the values one - by – one and uses a converter to change the string value to a specific data type that the bean is expecting to receive.

The converter is a class that has a convert() method accepting a string value and the class type that this must be converted to – for example "date". It then does the conversion and returns the date representation of the given string.

After convert in the value the populator calls the setter method of the form bean based on the name that was associated with that value. For example if the HTML was:

<input type=”text” name=”age”>,

the value entered was “10” and

the method setAge(int) was in the form bean,

then the populator converts the string “10” to number 10 and then calls the setAge(10) method based on the name of the input field.

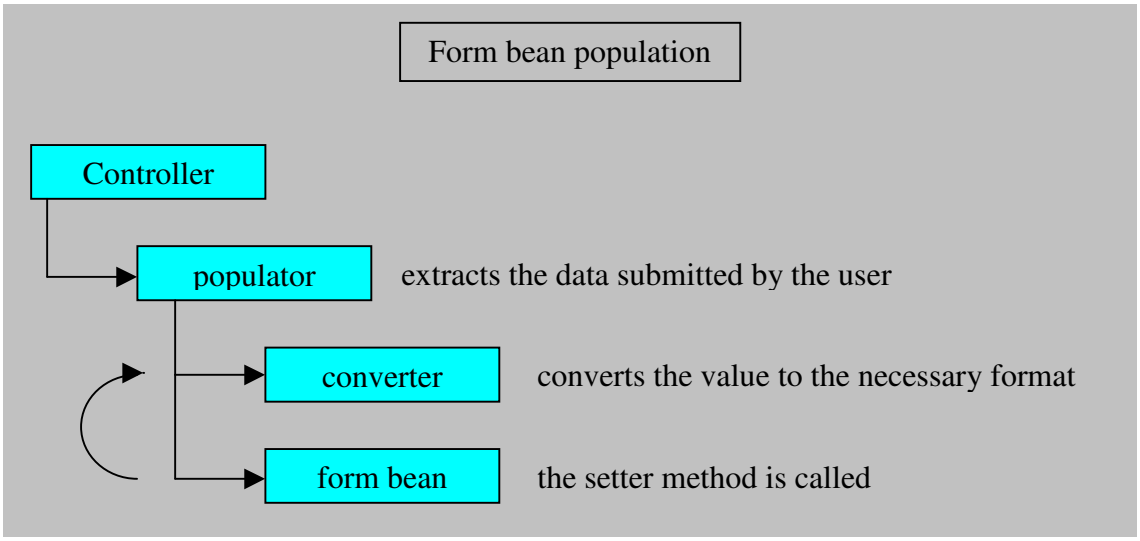

Figure 3. Form bean population

When a conversion error occurs, the error handler is called with the details of the error. The default error handler adds a new error to a stack with an error code “SYS0000” which is usually mapped in an error description file as “Wrong data type”. If any errors have been raised when the populator finishes, the Controller sends the user back to the original page. When the View (the page) is properly designed, it should then show the value or values that the populator was not able to convert highlighted in red and with the error message from the error description file attached to them.

7. The Model

The Model is the business logic of the application. This is the part that actually does the tasks the application is supposed to perform. In most applications the Model is the most reusable and the most resource-demanding component. Hence the Struts framework is designed to produce easily modifiable and highly reusable Model components.

To accommodate such abilities and to offer easy integration with already existing, Struts divides the Model into two separate parts:

1. ActionForms
2. Business logic

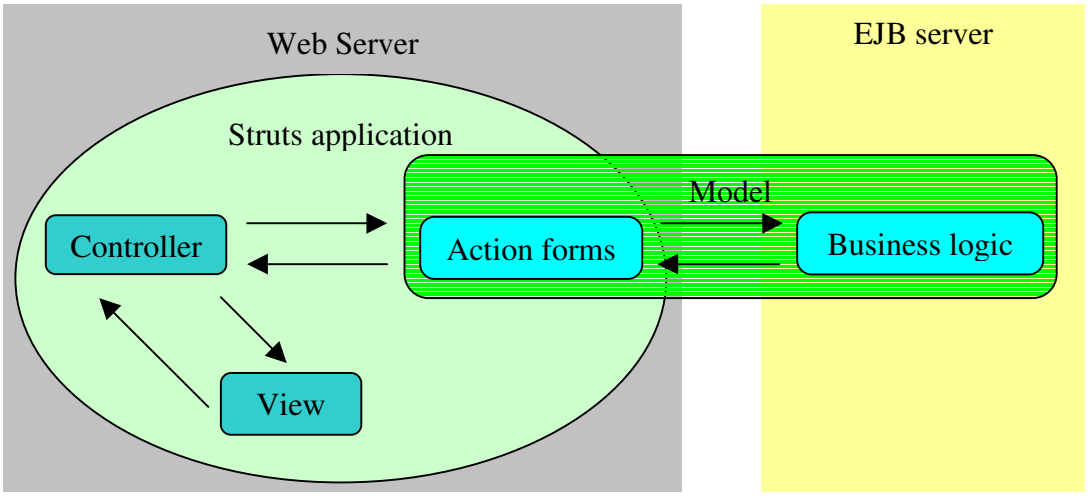

Figure 4. Struts using external business logic

7.1. Action Forms

Action forms are the Struts - specific part of the Model. Actions forms are Java classes that reside in a Struts application in the web server and perform the following actions:

- Form validation
- Error reporting
- Call business logic functions

- Interpret the result(s) returned by the business logic
- Decide which page the client will be forwarded next
- Pass the information to the View to be displayed

When a form is submitted, the first validation occurs during form bean population, however that only verifies the format of the information (textual, numerical, date, etc.). It is clear that before using that information to run business functions, it has to be verified to a much higher level. That verification, if not built into the business logic, is done primarily by action forms. Since action forms are supposed to extract the information from the form beans and present it to the business logic, they are also in the position to verify that information and abort the process, before any potentially damaging actions are performed.

The traditional way of handling an error is to send the user to a special error page. Struts provides an alternative solution. When an error is encountered, the action form can return an ErrorActionForward instead passing the collection of errors that occurred. For example when two entries were incorrect, action form adds these two separate errors into an error collection and returns the collection to the Controller with an ErrorActionForward. This usually sends the user back to the same page that generated the error unless a different fallback path has been defined in the configuration. How the View handles the errors, is explained in the View part of this document.

```
Errors errors=new Errors( );
if( form.getLogin().length() < 4 ){
      errors.addError( new Error( "login", "Error.addUserRequest.loginError" ) );
}
if( form.getPassword().length() < 4 ){
      errors.addError( new Error( "password",
"Error.addUserRequest.passwordError" ) );
} else if( !form.getPassword().equals( form.getPassword2() ) ){
      errors.addError( new Error( "password2",
"Error.addUserRequest.passwordMatch" ) );
}
if ( errors.size() > 0 ){
      return new ErrorActionForward( request, errors );
}
```

This example checks if the login and the password are both longer than 4 characters and that password and the retyped password are the same. If any of these conditions is

present, an error is thrown and the user is sent back to the page that this information originated from.

When an action form has validated the information submitted by the user, it calls the business logic to perform the requested function. In most cases this involves simply calling a method on the business logic class with the parameters being the information submitted by the user.

Since business logic is independent from a web server, the action form has to interpret the result returned by the business logic method or methods and must convert into a form that is usable by the View. For example if the action was to request a bank account balance, the information returned by the business logic would be the amount of money on that account. The action form has to associate that number with a specific text field on the result page.

Sometimes the result of the called business logic function is not only information that is to be displayed to the user, but as a result the user must be forwarded to specific pages. An example of that is when a user logs into a system. The information sent by the user are his username and password and based n the result, the user might be allowed to access the system or might be asked to check the password or to register. Since the business logic is not supposed to handle web-specific functions, that responsibility falls to the action form. The beauty of the Struts solution is that the action form does not have to know or hardcode the pages that it can forward the user to. The URL-s for the pages are in the Controller configuration and mapped to logic names. Hence when any changes in the structure of the application occur only the mapping in the configuration must be changed.

```
<action path="/LoginAction"
        type="actions.LoginAction"
        formName="LoginForm"
        formScope="session">
        <forward name="success" path="/login.html"/>
        <forward name="retry" path="/index.html"/>
</action>
```

For example this configuration creates two logic mappings “success” and ”retry”. And this is how an action form might use this:

```
if ( accessLevel <= 1 ){
        forward = mapping.findForward("success");
}else{
        forward = mapping.findForward("retry");
}
```

7.2. Business Logic

Business logic performs the business logic of the application. This is the most easily reusable component, if basic design guidelines are used to develop a Struts application. T is possible to write the business logic of an application directly into the action forms, but it is strongly recommended to have separate and therefore easily reusable components performing the tasks.

The business logic should be a generic class or a combination of classes performing a specific action. When properly designed, the business logic should be platform independent and also unaware of the method in which it is being deployed. This means that even though a Struts application is a web application, the business logic should never take tasks or use functions that are web server specific. If the necessity for web server specific tasks arises, they should be solved using action forms.

The generic nature of business logic components allows them to be used not only in other Struts applications but also in any future Java-based programs that might not be web-based. And vice-versa already existing Java components can very easily be incorporated into a developing Struts project.

The most commonly used deployment method of business logic is Enterprise Java Beans (EJB).

8. The View

The View is the part of the application that directly communicates with the user. The View usually consists of HTML and JSP pages where HTML pages are used to display static information and JSP pages for dynamic data. The usage of Multilanguage and custom tags allows developers to create sophisticated and highly interactive user interfaces with little effort and resources.

8.1. Custom tags in OD Struts

A custom tag is a way of introducing functionality into a JSP page without overwhelming it with Java code. A custom tag does not exist in HTML specification, but is interpreted by the web server and replaced by HTML code before it is sent to the user. This allows developers to hide complicated pieces of code behind simple and easily usable tags.

```
<%@ taglib uri="/WEB-INF/tlds/od-struts.tld" prefix="od" %>
<HTML>
<body>
<od:form
      name="login"
      action="LognAction.do"
      type="forms.LoginForm"
      method="post"
      labelCatalog="properties.loginPage">
         <od:label for="login" />        <od:text name="login" >
         <od:label for="password" />  <od:password name="password">
</od:form>
</body>
</HTML>
```

This is a simple JSP page asking the user for a login and a password, however it does much more than that. At the beginning of the page od-struts.tld tag library is associated with prefix “od”. When Tomcat processes the page, it searches for all the tags starting with the “od” prefix from the tag library. On this page the tags are:

- form

- label
- text
- password

The tag library associates a specific Java class with that tag name and also specifies which attributes this tag accepts. For a "label" tag the description is:

```
<tag>
        <name>label</name>
        <tagclass>com.oncedone.struts.taglib.html.LabelTag</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
        </info>
        <attribute>
                <name>for</name>
                <required>true</required>
                <rtexprvalue>false</rtexprvalue>
        </attribute>
        <attribute>
                <name>name</name>
                <required>false</required>
                <rtexprvalue>true</rtexprvalue>
        </attribute>
</tag>
```

The "label" tag accepts attributes "for" and "name" and attribute "for" is a required attribute. When the web server encounters the <od:label …/ > tag, it finds the description from the tag library and calls the "com.oncedone.struts.taglib.html.LabelTag" class to perform the actions associated with this tag.

Struts provides several custom tags to help create web pages and to extend the functionality of existing HTML tags. The following is short descriptions of some of the most commonly used:

form – form tag is the equivalent of HTML form tag, but with extended functionality. Forma tag is the required container for most other Struts tags. Its attributes configure the whole form with all its sub tags.

```
<od:form
        name="LoginForm"
        action="LoginAction.do"
```

```
        type="forms.loginForm"
        method="post"
        labelCatalog="properties.mappings"
        labelPrefix="login">
…
</od:form>
```

This example creates an HTML form tag that is later sent to the user, but it also associates the form with a form bean, specifies the property files that the language-specific texts are located and the prefix o the mapping that these texts have.

label – in an html form all the form elements have a label associated with them. In classic solutions this label is usually hardcoded into the HTML file. A label tag is Struts's replacement for that descriptive text. It locates the text it is supposed to display from the property file configured in the form tag. A label is also the default way of displaying errors. This is an example of a label – element pair:

```
<od:label for="password" /> <od:password property="password" size="20" />
```

When an error occurs and the user is returned to the form page where the error originated, the name of the form element causing the error is passed along with that error. When a label tag is writing itself, it checks, if there are any errors with the same name that it has. If there are, it displays itself as a link to the error description.

On this HTML page the label displays the "Password" text and clicking on the link shows the error text

The following tags are Struts extensions of standard HTML tags. When a page is submitted, the Controller extracts the information from the HTML form and puts it into the form bean. Struts tags also support populating the from with the information inside

the form bean. When Struts displays a page with a form created with Struts tags instead of standard HTML tags, it attempts to look up the form bean and the specific value for each form object inside that bean. For example, for a text field with the name "userName" Struts will try to get the value using a function getUserName(). If that value is retrieved, the tag is written with a "value" attribute. If the value were "user" the final HTML code would be: <input type="text" name="userName" value="user" />. This way Struts prefills all forms with existing information.

This a very effective solution coupled with the error reporting process. When an error is sent either by the action form during validation or the converter during bean population, the user is sent back to the page where he entered the incorrect information. When Tomcat is creating this page, all the information that the user submitted, is already in the form bean and therefore placed into the created page. This means that the user does not have to enter anything twice, but can fix the incorrect piece of information and resubmit the page.

text – corresponds to HTML <input type="text" … /> tag.

password – corresponds to <input type="password" … /> tag.

textarea – corrsponds to <textarea …> … </textarea> tag.

submit – corresponds to <submit … /> tag.

message – displays text to users. Message tag creates Multilanguage texts on a web page. Struts allows a single page to be displayed in several languages. To accomplish this no user - visible texts can be hardcoded into the JSP page. A message tag supplies a key that is used to look up the language – dependent text.

messageGroup. Since the message tag only supplies a key for looking up the text it must be in the scope of a messageGroup tag that gives the location of the language file. It is not smart to code the page layout information inside a message tag since this does not depend on language and would create large and unreadable language files. To avoid the duplication of language file information among the many message tags, they are contained within a messageGroup tag.

8.2. I18n

I18N is the nickname for internationalization – the process of making an application support several languages. Struts is designed with internationalization in mind with most custom tags supporting several languages by default and with easy ways to include multilanguage texts into a web page.

The basis for I18N in Struts is Java's multilanguage support class called ResourceBundle. ResourceBundle handles several text files with key-value pairs, choosing the appropriate text file for a specific language. The naming convention for the text files is "name_language_country.properties". For example mappings_en_us.properties would denote a file containing US English mappings. Struts tags use Resource bundle to get the language-specific texts from a collection of text files based on the language and country codes provided either by the client or the server.

The country and language codes are ISO standards. The language codes are based on ISO-639 standard and the country codes on ISO-3166 standard.

Most commonly used tags that display information to the user are "label", "message", "options" and "radio". All of these tags except "message" are inside a "from" tag and thereby associated with a label catalog which is the base directory and name of their language – specific properties files. All of these have a name associated with them that serves as the key in these properties files. The value associated with the key is the actual message displayed to the user.

For example the following tag <od:label for="login" /> has a name "login" and that name is the key in the language file. In the loginPage_en_us.properties file the mapping might be: "login=Enter your username" and in the loginPage_et_ee.properties file the mapping might be: "login=Sisesta kasutajanimi". When the label tag is run, it uses a ResourceBundle to find the appropriate file and then extracts the value using the current language and country settings. If the language was "en" and country "us" the value returned would be "Enter your username".

ResourceBundle tries to find the exact match to the language and country codes, however if there is none, it then attempts to find a file with only the language code. For

example if the file loginPage with language “en” and country “us” is needed and no loginPage_en_us.properties file exists or there is no required aping in that file, ResourceBundle attempts to find the loginPage_en.properties file. If that file is also not found, the file loginPage.properties containing the default language is searched next.

Struts attempts several ways of finding the language and country codes to use. First Struts checks the session object. If there is no language information stored in the session, Struts attempts to get the language from the request. When a browser sends a request for a page, it attaches the desired language to the request. If Struts has found no preferred language from the session or the request, it takes the default language of the web server. Knowing this, it is easy to override the language selection, for example when user has manually selected the language, by putting that language and country code into the session.

9. The Controller

Many of the Controllers functions have already been discussed in this document in other sections. Since Struts operates as a whole, it is difficult to discuss one single component without insight into the whole of the system.

The Controller controls a Struts application, receiving client requests and calling other components. The Controller ties together the otherwise separate View and Model components.

- Receives and routes client requests
- Runs filters
- Creates and manages form beans
- Populates the form beans
- Calls the Model
- Calls the View
- Easily configured by an XML file

Since the Controller provides standard services for the application, it is usually not created by the programmer but only configured. In almost all cases the Struts provided Controller and supporting files are sufficient for any web application, but if the necessity arises for functionality beyond those provided by Struts, all components can easily be rewritten or overridden to include the necessary functionality.

9.1. Filters

A filter is a piece of program that can be called by the Controller before Struts starts processing a user request. When the Controller receives a request by a user, the first thing it does, is check if there are any filters associated with that page in the configuration file. Filters can take major decisions or perform tasks before Struts starts processing the user request. An example of that would be a login filter. In a multi – user environment there are often several user roles and the tasks performed by a role may be performed by no one else. To insure that, every action form accessible by a certain role must check if

the user belongs to that role. This task of ensuring that a user belongs to a certain role can instead be given to a filter, which would free the action forms from repeating the same piece of code and easily changeable without having to worry about updating all instances.

```
<action-filters>
        <action-filter name="LoginFilter" type="filters.LoginFilter">
                <property name="role" value="administrator"/>
        </action-filter>
</action-filters>
```

Struts does not limit the number of filters that can be run a single page even though the configuration of a page only accepts one filter. To solve this Struts has a filter chain concept, where several filters can be combined into a single filter chain and that chain is passed as a filer to a specific page. With the ability to include other filter chains as well as filters this provides a simple and easy solution to filter encapsulation

```
<action-filter-chains>
        <action-filter-chain name="FilterChain1" >
                <action-filter-link name="DebugFilter"/>
                <action-filter-link name="TokenFilter"/>
        </action-filter-chain>
        <action-filter-chain name="FilterChain2">
                <action-filter-chain-link name="FilterChain1"/>
                <action-filter-link name="CheckLogonFilter"/>
        </action-filter-chain>
</action-filter-chains>
```

An action mapping with a filter would look like this:

```
<action-mapping       path="UpdateUser"
                type="com.mydomain.actions.UpdateCustomerAction"
                filterName="FilterChain1"/>
        <forward name="success" value="acceptChanges.jsp"/>
        <forward name="failure" value="error.jsp"/>
</action-mapping>
```

10. A Real-Life Example

To show how a Struts application actually works, I have provided a real - life example. It should be included with his document on a floppy disk, but is also fond on the Internet under the following address:
http://www.tpu.ee/~normakm/movDB.exe

This is the source code of the application in a self – extracting zip achieve. To see the application work go to the following address:
http://212.47.216.220:8080/movDB/

11. Conclusion

At the moment there are two main competing technologies for large – scale client server and web development: J2EE (Java 2 Enterprise Edition) and Microsoft .NET. Struts is compliant with J2EE and supports several design patterns associated with application development in Java.

For companies Struts offers a lot. The companies can use people with skills otherwise lacking in web development and compile a full working product from their contributions. Struts the quick development and easy maintenance needed in today's fluid market conditions by allowing the reuse of existing products and providing easily changeable components within an application.

Struts also helps the development of small – scale projects. Having an overview of the entire application, the ability to create a multilanguage environment for the users simply by editing text files outweigh the slight increase in time and effort inherent to Struts web development.

The real beauty of the Struts framework is that one does not have to know it in depth to use it for developing sophisticated web applications. Any person familiar with Java and HTML is capable of creating Struts application acquiring only practical knowledge of the environment.

In a longer perspective Struts is pointing the way for any future tools aiding in the slow and painful development process. The functions of those tools will still include those of Struts even though they might be implemented in a different manner.

12. Kokkuvõte

Internet on muutnud maailma. Ellu jäämiseks peavad firmad oma veebiteenuseid pakkuma seniste staatiliste HTML lehekülgede asemel dünaamiliste lahendustena. Tooted ja teenused peavad olema kiiresti turule paisatavad, võimalikult mitmekülgsed, dünaamiliselt turu vajadusele muudetavad, võimalikult odavad valmistada ja hallata. Nende eesmärkide saavutamiseks on loodud mitmeid vahendeid, mis lihtsustavad ja lühendavad tarkvaraarendusprotsessi samas ohverdamata loodavate toodete mitmekülgsust. Üks sellistest vahenditest on Jakarta Struts.

Strutsi loojaks oli Craig R. McClanahan, kes mais 2000 annetas selle Apache Software Foundation -ile. McClanahan on jäänud siiani Strutsi projekti juhatajaks, kuigi tegutseb aktiivselt ka teiste Apache projektidega. Struts on üks Apache Jakarta projektidest, mille eesmärgiks on luua kvaliteetseid serverilahendusi Java platvormile, mida arendataks vabas ja koostööaldis keskkonnas.

Struts ei ole mitte valmisprodukt, vaid raamistik, mille koodi võib soovi korral muuta. Struts on vabavara ja seda võivad kasutada era- ja ärikliendid nii alg- kui nende poolt muudetud kujul oma toodete komponendina.

Rääkides Strutsist baseerun ma OD Strutsil, mis on Profit Softaware OY poolt kasutusele võetud ja arendatav Strutsi implementatsioon.

Strutsiga seotud tehnoloogiad

Java – programmeerimiskeel. Java on üldotstarbeline objekt-orienteeritud programmeerimiskeel, mille arendamisel on eesmärgiks olnud võimalikult vähe arvuti arhitektuurist sõltuda. Seetõttu on Java programmidel tähelepanuväärne omadus töötada erinevate operatsioonisüsteemidel ja erinevatel standarditel ehitatud arvutitel. Java programme jooksutab virtuaalne masin, mis arvuti arhitektuurist sõltumatu koodi tõlgib arvuti masinkoodi ning käivitab selle. Tõlkimisprotsess aeglustab märgatavalt programmide käivitamiskiirust, kuid selle kao teeb tasa keele mitmekülgsus ja lihtne õpitavus.

Servlet – Tehnoloogia, mis lubab veebiserveril luua dünaamilisi veebilehti. Uuemad tehnoloogiasuunad eelistavad suurema osa tööst teha keskserveris ja klientide masinaid mitte koormata. Servlet on programm, mis töötab veebiserveris, ja loob igale kliendile oma HTML lehekülje. Selle tehnoloogia eelis tavalise HTML lehekülje ees on, et igale kliendile saab kuvada erineva lehekülje, näiteks laoseisu päringu hetkel.

JSP - Java Server Pages. JSP on edasiarendus Servleti tehnoloogiast. Luues Servleti abil HTML lehekülge, peab sinna rida-realt kirja panema. See on tülikas ning pärast raskelt loetav ning muudetav. JSP lehekülge hoitakse lähtekoodilisel kujul. Kui kliendilt tuleb päring JSP leheküljele, siis veebiserver loeb vastava lehekülje, kompileerib selle ja käivitab. Töö lõppedes jääb lehekülg veebiserveri mällu ja järgnevad päringud samale lehele on sama kiired kui servlet tehnoloogia lahendus.

EJB – Enterprise JavaBeans. Tehnoloogia, mis lubab Javas kirjutatud äriloogikat installeerida ja käivitada spetsiaalsetel EJB serveritel. Samas toetab EJB ka kasutajakoormuse jagamist mitme arvuti vahel. Kui server, kust äriloogikat küsiti tegeleb millegi muuga, siis võib äriloogikat jooksutada hoopis teine server, kellele on installeeritud sama äriloogika komponent ja vastata esimese serveri eest.

Struts ja mudel 2

JSP tehnoloogia ilmudes võeti see mitmete arendajate poolt kiiresti kasutusse. Kasutusmudelit, kus kogu veebirakendus on üles ehitatud JSP lehekülgedele kutsutakse mudel 1. Varsti selgus, et ei ole lihtne kogu äriloogikat JSP lehekülje sisse kirjutada ning sellise lehekülje haldamine muutub peaaegu võimatuks. Selle probleemi lahendamiseks kasutab Struts MCV (Model Cotroller View) lahendust, ka tuntud kui mudel 2

Struts on modulaarne struktuur. Struts jaotab loodava veebirakenuduse kolmeks üksteisest funktsionaalselt eraldi seisvaks komponendiks: mudel, kontroller ja vaade.

· Mudel on äriloogika - programm või programmid, mis teostavad teenust, mida kliendile pakutakse. Näiteks andmebaasiliides, mis tagastab pliiatsite arvu laos või programm, mis kannab rahasumma arvelt A arvele B.

· Vaade on JSP lehekülg, mis suhtleb kliendiga. Sellel on vormid, mida kasutaja täidab ja sellele kuvatakse kasutajale vajaminev info.

· Kontroller on keskne servlet, kes käivitab vastavaid Mudeli programme ja tagastab saadud tulemuse vastavale JSP leheküljele, mida siis kuvatakse kasutajale.

Kontrolli liikumine Strutsi rakenduses

Klient kontakteerub veebiserveriga küsides lehte "HelloWorld/HelloWorld.do". See tähendab HelloWorld.do lehekülge HelloWorld rakenduses. Veebiserver kontrollib oma konfiguratisiooni ja leiab, et kõik .do leheküljed tuleb saata Kontrollerile. Kontroller leiab, oma konfiguratsioonist, et vastavat lehekülge teostab Java klass "strutsDoc.HelloWorld.HelloWorld" ja käivitab selle. Antud näites saadab Mudel kliendi "welcome" lehele. Kontroller leiab oma konfiguratioonist, et "welcome" lehele vastab URL "HelloWorld.html" ja suunab kliendi sellele lehele. Klient päris "HelloWorld.do" lehekülge ja talle tagastati tema teadmata "HelloWorld.html".

Andmete liikumine ja hoidmine Strutsi raamistikus

Enne kui uurida põhjalikumalt Strutsi moodustavate komponentide olemust, tuleb selgitada, kuidas liigub info Strutsi raamistikus. Selleks tuleb selget vahet teha kahel liikumissuunal: Vaade - Klient - Kontroller ja Kontroller - Mudel - Kontroller - Vaade.

Kuna kliendini jõuab HTML lehekülg, siis on info liikumine Vaatelt kliendini piiratud HTML võimalustega. See tähendab, et Mudeli poolt tagastatud info peab olema Vaate poolt kirjutatav HTML dokumenti. Info liikumine kliendilt Kontrollerini on võimalik kahte teed pidi, kas klient vajutab lingile (või kirjutab brauserisse aadressi), või täidab oma brauseris HTML vormi väljad ja vajutab "Saada" (Submit) nuppu. Esimesel juhul

saadetakse Kontrollerile HTTP päring, kus on kirjas lehekülg, millele klient soovis minna. Teisel juhul loeb brauser kliendi poolt täidetud lahtrite väärtused ja paneb need HTTP päringusse koos lehekülje aadressiga, kuhu need taheti saata.

Info liikumine veebiserveri sees Kontrolleri Mudeli ja Vaate vahel toimub Java objektide kaasabil. Nendeks objektideks on erilised Java klassid - vormi oad (form bean). Igale vormi oa muutujale peab vastama kaks funktsiooni, tuntud kui getter ja setter funktsioonid, millest üks, "get", tagastab vormi oa vastava muutuja väärtuse ja teine, "set", muudab vormi oas oleva muutuja väärtust.

Sel viisil tekib andmehoidla, kus andmeteks on vormi oa muutujad ning vormi uba kasutavad programmid saavad neid muutujaid funktsioonide abil lugeda ja muuta. Kui vormi uba esimest korda luuakse, siis asetatakse ta veebiserveri mällu temale määratud kehtivuspiirkonda (scope). Kaks kõige kasutatavamat on päring ja sessioon.

Vormi oa täitmine

Kliendilt tulev info on tekstilisel kujul isegi kui see tähistab näiteks aastaarvu. Enne kui äriloogika seda kasutada saab, tuleb see konverteerida õigele kujule. Kuna konverteerimisprotsess on tavaliselt standardne, siis pakub Struts selle standardset lahendust. Strutsi komponent “poulaator” hoolitseb kliendilt tuleva info eraldamise eest ja selle muutmise eest õigele kujule konverteri abil. Konverter on Java programm, mis suudab tekstilist infot tõlgendada vajalikule kujule, või kui see pole võimalik, siis teavitada sellest klienti. Näiteks kui aasta tekstiväljale sisestati tähti numbrite asemel, siis ei ole seda võimalik numbriks muuta ja kliendilt tuleb antud infot uuesti küsida.

Mudel

Struts jagab Mudeli kaheks alamkomponendiks:

- Tegevus (Action) - otsustab "mida teha".
- Äriloogika (EJB - Enterprise Java Beans) - teostab vajalikku tegevust.

Strutsi raamistikus on püütud toetada komponentide taaskasutatavust. See tähendab, et komponenti, mis on loodud mingi programmi või veebirakenduse koosseisu, võib taaskasutada ka teistes programmides. See eeldab, et komponent on piisavalt üldise arhitektuuriga. Tema sisend, väljund ja tegevus ei tohiks sõltuda situatsioonist, kus ta käivitatakse. Kuna Mudel peab kasutama veebiserveri funktsionaalsust, näiteks milline kasutaja sisestatud info on rakenduse jaoks mittesobilik ning suunama kasutajat erinevate lehekülgedele, peab äriloogika lahutama veebiserveri kesksest funktsionaalsusest.
Tegevus teostab järgmisi toiminguid:

· Kasutaja poolt sisestatud info verifitseerimine.
· Äriloogika välja kutsumine
· Äriloogika poolt tagastatud info põhiselt otsustamine, millisele leheküljele kasutaja edasi saata.
· Info tagastamine Vaate komponendile, millele klient suunatakse.

Näiteks kui Tegevus kontrollib kasutaja sisselogimist, siis võib see toimuda järgmiselt:

· Kontroller käivitab Tegevuse ja annab talle sisselogimise infot sisaldava vormi oa.
· Tegevus loeb kasutajanime ja salasõna vormi oast ja käivitab äriloogika.
· Äriloogika kontrollib kasutajanime näiteks andmebaasist ja tagastab kas tõese (kasutaja lubatakse sisse logida) või väära.
· Tegevus kontrollib tagastatud väärtust ning kui see oli "tõene", siis suunatakse kasutaja edasi. Samas võib Tegevus Vaatele edastada ka kasutaja nime, et saaks kasutajale nimeliselt viidata, näiteks "Teretulemast tagasi Mikk".
· Kui Tegevusele tagastati "väär", siis suunab Tegevus kasutaja leheküljele, kus öeldakse, et sisselogimine ebaõnnestus.

Sel viisil muutub kasutaja sisselogimist kontrolliv äriloogika komponent ülejäänud veebirakendusest iseseisvaks ja seda saavad kasutada ka teised samalaadset teenust tarbivad veebirakendused ja programmid.

Tegevus annab käivitusjärje oma töö lõppedes tagasi Kontrollerile, kes Tegevuse poolt tagastatud info alusel käivitab Vaate vastava objekti. Info, mida Tegevus tagastab

Kontrollerile, ei sisalda lehekülje aadressi, kuhu kasutaja suunatakse, vaid ainult sümboolset viidet. Igale sellisele viitele on vaste Kontrolleri konfiguratsioonifailis. Näiteks võib sisselogimist kontrolliv Tegevus tagastada: suuna tagasi leheküljele "sisselogimine".

Vaade

Vaade on Strutsi rakenduse osa, mis suhtleb otseselt kliendiga. Vaateks on tavaliselt dünaamilise info näitamiseks JSP leheküljed ja staatilise info jaoks kasutatakse HTML leheküljed.

JSP kasutamine koos kasutaja poolt loodud märgenditega (custom tags) laiendab veelgi Struts toetab peale tavalise tekstivälja ka muid HTML vormi elemente. Täielik nimekiri asub http://jakarta.apache.org/struts/struts-html.html. Suurem osa neist käituvad sarnaselt nende HTML kujule.

Peale selle toetab Vaade ka mitmekeelseid veebirakendusi. Selle teeb võimalikuks Java vastav funktsionaalsus, mis on realiseeritud ResouceBundle klassi poolt. ResourceBundle klass lubab samal programmil kuvada erinevas situatsioonis erinevas keeles kasutajaliidest. Infot selle kohta, millist keelt kasutada, on võimalik saada mitmest allikast. Leheküljel võib olla keele valik, kust kasutaja valib endale sobiva keele, iga HTTP päringuga saadetakse serverile ka keel, mille kasutaja on valinud enda brauseris vaikimisi keeleks. Veebiserverile võib anda vaikimisi keele, mida kasutatakse, kui muul viisil ei olnud võimalik kasutaja keele eelistust teada saada. Seejärel väärtustatakse sessiooni muutuja "Locale" kujul keel - kahe tähega (ISO-639 standardi järgi) ja maa - kahe tähega (ISO-3166 standardi järgi). Näiteks "EE, ee" tähendab eesti keelt Eestimaal, "EN, us" tähendab USA inglise keelt ja "EN, gb" tähendab Inglismaa inglise keelt.

Näiteks kui JSP lehel on kirjas <pre:label for="tekstiväljaNimi" /> ja keel on "EN, us" ja nimesid otsitakse "Nimed" failist, siis otsitakse kõigepealt "nimed_EN_us.properties" faili. Kui leitakse selline fail, siis võetakse nimi sellest failist, näiteks "tekstiväljaNimi=Text field" ja kuvatakse kasutajale. Kui aga keeleks oli "ET, ee", siis otsitakse faili "nimed_ET_ee.properties", kus on hoopis "tekstivälajaNimi=Tekstiväli".

Kuna ei ole reaalne, et rakendust kõigis keeltes esitataks, siis faili "nimed_ET_ee.properties" puudumisel otsitakse faili "nimed_ET.properties" ja kui ka seda ei leita, siis faili "nimed.properties". Nii on garanteeritud, et ükskõik, millises keeles klient ka ei räägiks, saab ta veebirakendust igal juhul kasutada.

Kontroller

Kontrolleri funktsioonid on eelnevalt juba loetletud. Kontroller on Strutsi raamistiku keskne komponent, mis käitub liidesena teiste veebirakendust moodustavate osade vahel. Kuna Kontrolleri tegevused on standardsed, siis ei ole tavaliselt vaja muuta Strutsi enda poolt pakutud Kontrollerit, vaid piisab selle käitumise juhtimisest konfiguratsioonifailiga.

Filtrid

Enne kui Kontroller käivitab vormi oa täitmise või Mudeli, kontrollib ta, kas antud leheküljega on seotud mingeid filtreid. Kui on, siis käivitab Kontroller kõigepealt filtrid. Filtrite abil on lihtne kirja panna korduvaid tegevusi, mille muidu peaks teostama Mudel. Näiteks võib filter kontrollida, kas kasutaja on sisse loginud ja kui mitte, siis suunata ta sisselogimise leheküljele ja seda enne, kui Kontroller on käivitanud Mudeli või Vaate. Selline lahendus garanteerib, et kogemata (või meelega) valele leheküljele sattunud inimene ei tekita rakenduses olevale infole kahju.

13. Used Literature

1. http://jakarta.apache.org/tomcat/index.html
2. http://jakarta.apache.org/struts/index.html
3. http://java.sun.com
4. http://java.sun.com/products/servlet/index.html
5. http://java.sun.com/products/jsp/index.html
6. http://www-106.ibm.com/developerworks/library/j-struts/
7. http://drdb.fsa.ulaval.ca:8080/dr/html/struts/strutsPres_en.html
8. http://www.javaskyline.com/learnejb.html
9. http://www.din.de/gremien/nas/nabd/iso3166ma/codlstp1/
10. http://www.loc.gov/standards/iso639-2/langcodes.html