3. Implementation

Contents

From the previous chapters, two rounds of framework selection process, and analysis were done. The chosen frameworks for prototype implementation are Seam, Wicket, JSF, Struts2. The prototype implementation is based on the existing system (Customer and Contract Management System). All main functionality such as create, retrieve, update, delete, and assign remains, but some part of the flow and user interface layout are redesigned. The purpose of the prototype implementation is to understand the architecture, background processes, artifacts produced under the same system, and proof of concept for each frameworks selected from the previous step. The new system flow of the prototype [Figure 3.1] derived from the original flow shown in [Figure 2.2]. After did the system flow analysis, some flow were removed due to ambiguity and complexity. The new system flow aimed to make the system easier for user to use, understandable, and more productive.

For example, according to the original system flow, in order to assign customer to contract, or other way around, there are two ways: navigates through detail of the customer/contract, then navigates through assign contract/customer link, or create customer/contract, then though the same process as the first way. In the new system flow, the customer/contract detail page and assign customer/-contract are combined together, and the result from the assignment is refreshed and shown automatically after the assignment selection and confirmation.

The existing system divides into three sub-projects: muster, client, and ear [Figure 3.2]. These three sub-projects responsible in three different areas. The muster project contains whole Business layer and everything necessary for Data layer communication such as location of the database system, username/password of the database, query, entity beans (required when query using the Object Relational Mapping (ORM)), and etc. In prototypes development, there is no need to change any part of this project. Since, this master thesis's goal is to investigate the presentation layer so, the backend is built once and used in every prototypes. The client project reflects the Presentation layer. This project is the focused project in this master thesis. The architecture and the artifacts

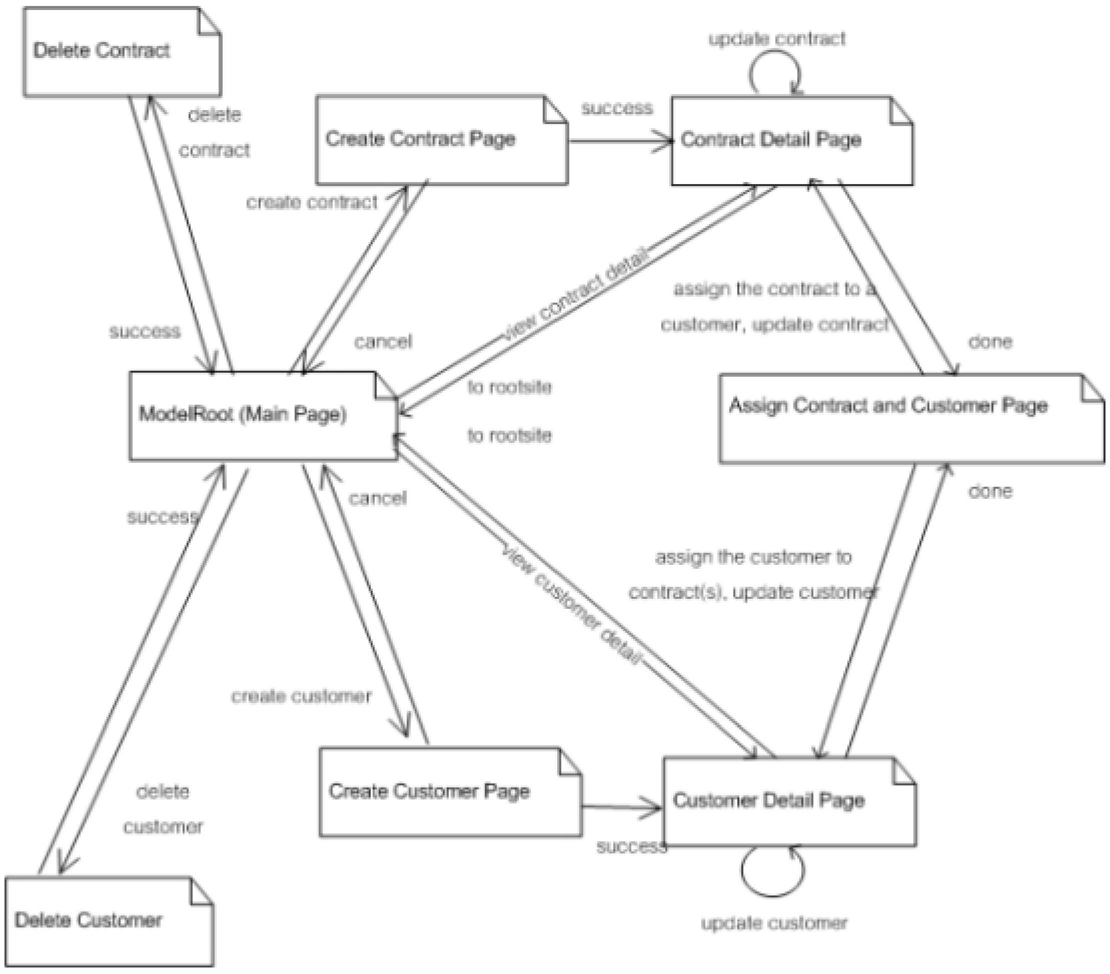
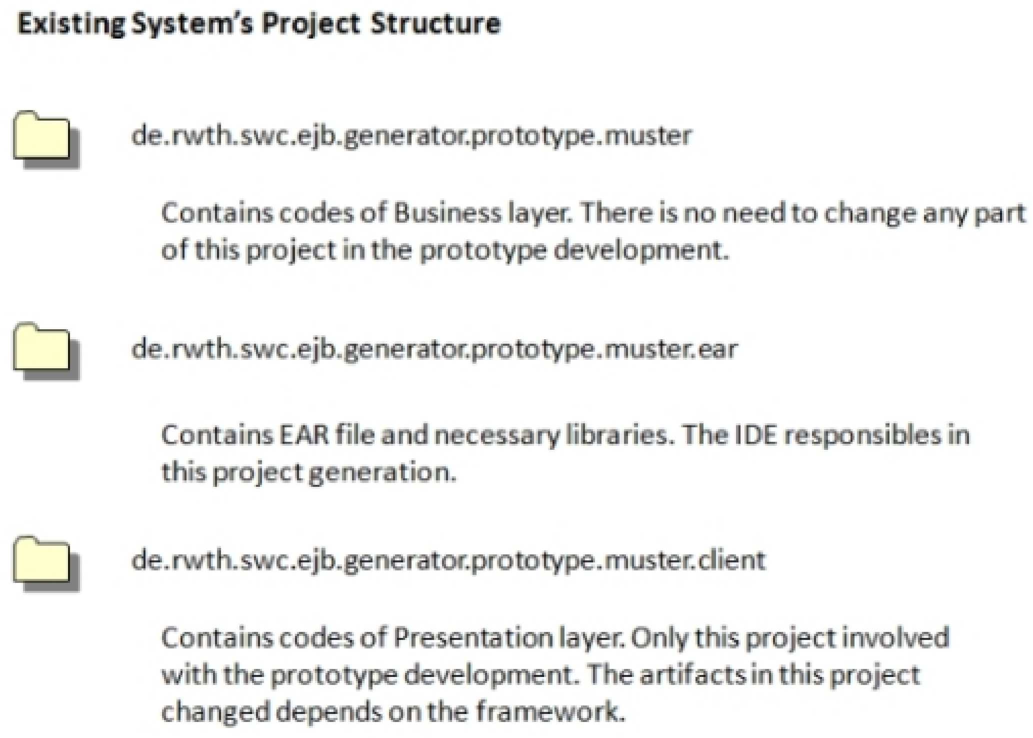


Figure 3.1.: New Page Flow Diagram

in this project are changed depend on the framework used in each prototype. The ear project contains EAR file, which need to be publish to the web server in order to makes the prototype available. Normally, this project managed by the IDE. Whenever the whole system compiled, the ear file and the required libraries are updates, then the ear file need to be published to the web server.

In the client project [Figure 3.3], Like other ordinary web application, the views (JSP) located in the WebContent folder and the web.xml, which is the default configuration file for every web application project located in the WEB-INF folder. The Servlets are in the src folder. The Action Handler translates the request into action and delegates to the corresponding action class. The Action classes share some similarity by inherit from the super class, Action. Those action classes communicate with the Business layer by calling the methods in the Application Facade, the only portal to the Business layer (muster project). The detailed muster project structure explained in the appendix of this research paper.


Existing System's Project Structure

de.rwth.swc.ejb.generator.prototype.muster

Contains codes of Business layer. There is no need to change any part of this project in the prototype development.

de.rwth.swc.ejb.generator.prototype.muster.ear

Contains EAR file and necessary libraries. The IDE responsibles in this project generation.

de.rwth.swc.ejb.generator.prototype.muster.client

Contains codes of Presentation layer. Only this project involved with the prototype development. The artifacts in this project changed depends on the framework.

Figure 3.2.: Existing System Project Structure

In the following section of this chapter, failure of Seam prototype implementation is also introduced, then JSF, Wicket, and Struts2 implementation including details information are described consecutively.

3.1. JBoss Seam

Seam is a powerful open-source full-stack framework for building web application based on Java. Seam integrates other technologies and frameworks such as :

- Full JSF-based AJAX supported framework: RichFaces, IceFaces.
- Presentation Layer: JSF, Wicket, Google Web Toolkit (GWT).
- Business and Data Layer: Java Persistence API (JPA), EJB 3.0, Hibernate.
- Business Process Management: Java Business Process Management (jBPM).
- Security Framework: Drools.

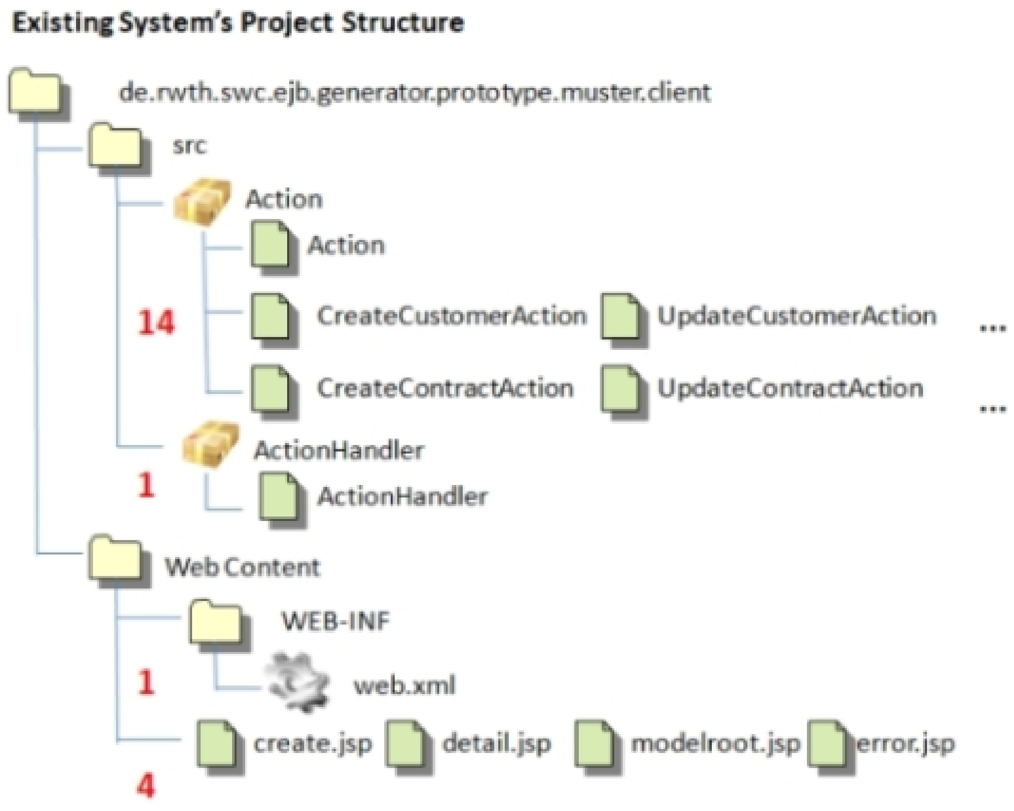


Figure 3.3.: Client project structure

Since, Seam provides alternative several presentation layer development frameworks fully integrated with EJB covered backend, which is the proposed architecture from the beginning, and two of three presentation layer development frameworks integration provides by Seam are the chosen frameworks (JSF, Wicket). Altogether with many enhancements provided by Seam, instead of develop three prototypes independently, at that point, integrated Seam to the whole proposed architecture, then replaced only the presentation layer with the chosen frameworks was the wiser decision [Figure 3.4]. The benefits of Seam integration other than the enhancements mentioned in chapter two is the ease of prototype development. Seam tied Wicket and JSF seamlessly to the backend and does not required any Action beans. Also, it is simple to develop the next prototype after the first one by changing only view technology without changing anything at the backend.

Unfortunately, Seam prototype implementation was failed. The cause of failure suspected to be the incompatibility of Seam and IBM Websphere 7.0 Application Server (WAS). Even the JBoss community's resource already stated the Seam installation on IBM Websphere 7.0 Application Server [KMR$^+$11], but in practical, it is not working in the existing environment. In consequences, the prototype implementation plan changed back to the original plan, which is

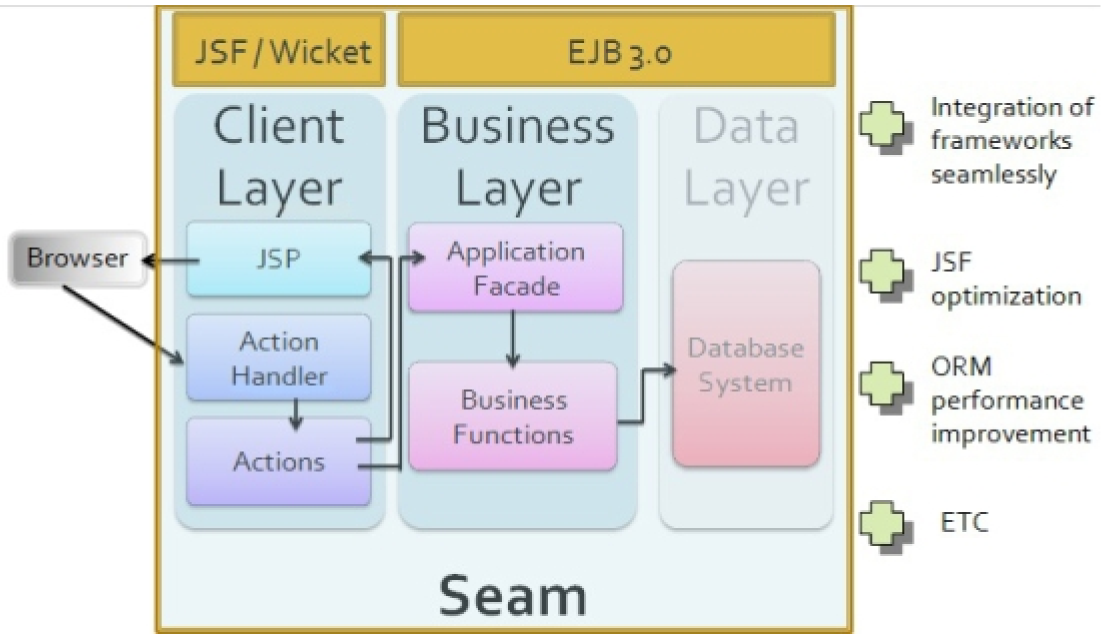


Figure 3.4.: Seam integrated architecture

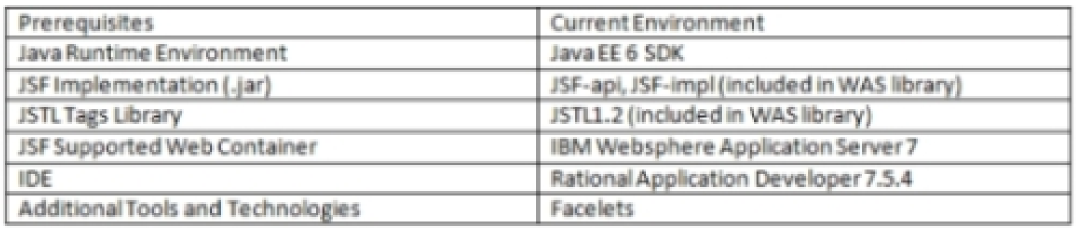
develop the remaining prototype independently without Seam.

3.2. Java Server Faces (JSF)

Java Server Faces (JSF) is a Java-based web framework, which simplify the development integration of web-based, server-side user interface. JSF is also a request, component-driven MVC web framework developed under JavaEE and JSR official standard.

JSF's component architecture not just enables standard JSF UI widgets (buttons, hyperlinks, textfields, checkboxes, and etc.), but also sets the stage for third-party components libraries. JSF components are event-oriented, so JSF allows client-generated event such as on click, on roll over, on release, on type, and etc. If compares to other technology or framework currently out there, the concept of JSF is similar to Microsoft Visual Studio .NET.

Even the official latest stable release of JSF is version 2.0, but the risk from failure of environments incompatibility which occurred in Seam prototype implementation shouldn't be overlooked. Therefore, the prototype was developed using JSF 1.2 implementation, which already contained in WAS and already tuned the compatibility with WAS.



Prerequisites	Current Environment
Java Runtime Environment	Java EE 6 SDK
JSF Implementation (.jar)	JSF-api, JSF-impl (included in WAS library)
JSTL Tags Library	JSTL1.2 (included in WAS library)
JSF Supported Web Container	IBM Websphere Application Server 7
IDE	Rational Application Developer 7.5.4
Additional Tools and Technologies	Facelets

Figure 3.5.: JSF Prerequisites

3.2.1. Working Environment and Tools

JTSL is one of the prerequisites because, JSTL used as a default JSF component renderer until JSF 2.0 and newer version. Facelets is the default renderer for JSF 2.0 and newer version, and required, in order to achieved the second priority requirement for the lower version.

3.2.2. Architecture

JSF has a very complex architecture, which consists of several design patterns working behind the scene [Jos05]. Developers only need to know five components: FacesServlet, UI Components, JSP, Managed Beans, and Navigation Rules. [Figure3.5]

Developers only need to know five components: FacesServlet, UI Components, JSP, Managed Beans, and Navigation Rules.

The FacesServlet is the front controller of the system. All the requests from the client have been translated into mapped action and delegated to the corresponding page and the response from the system is sent back to the requested client. Every JSF pages contain JSF components both basic components (JSTL tags, Facelets), which represent only standard HTML, or third party libraries (PrimeFaces, ADF Faces, Trinidad, IceFaces, RichFaces), which provide advance widgets like trees, grids, tabs, or personal custom components. Each component's value might bound to the value in Managed Bean, and some components have event listeners, which call the methods in the Managed Bean, when triggered. Those methods in the Managed Bean handle the communication with the backend. The Navigation rules manage all page flows of the system.

3.2.3. Basic Concepts and Life Cycle

Similar to all other web applications [Figure 3.6], JSF life cycle starts from client sends request by navigates link, button, or enter a URL to Web Server. After that, the server responses back as an webpage.

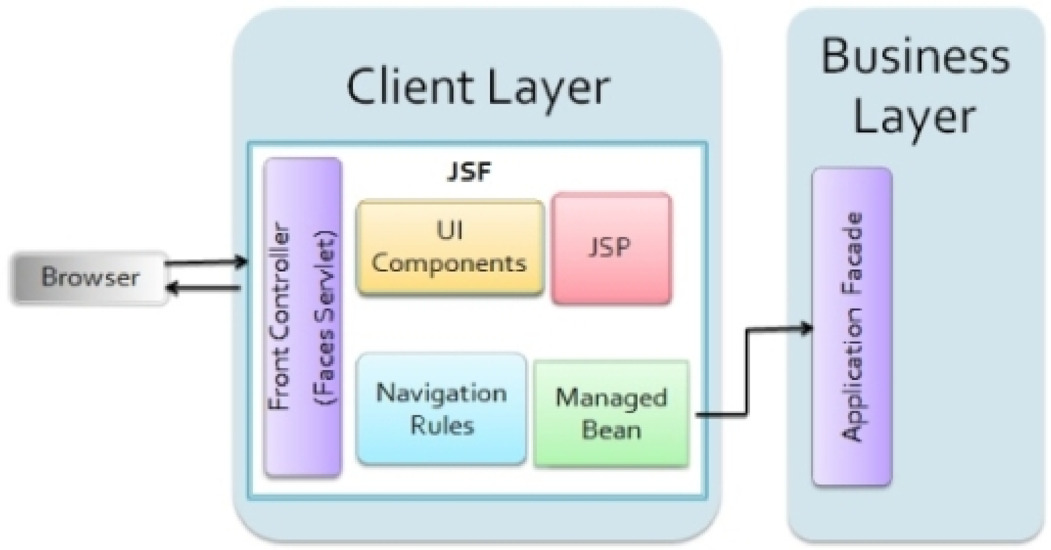


Figure 3.6.: JSF integrated architecture

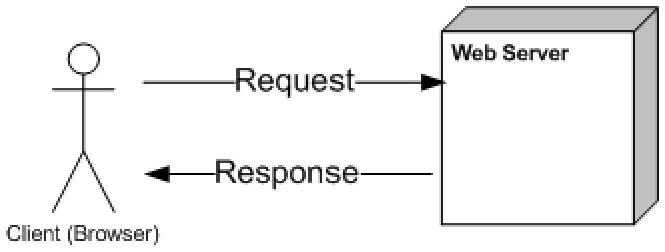


Figure 3.7.: Simple web application life cycle

To be more specified, if takes a closer look, JSF life cycle divides into 6 phases: restore view, apply request values, process validations, update model values, invoke application, and render response [Figure 3.7].

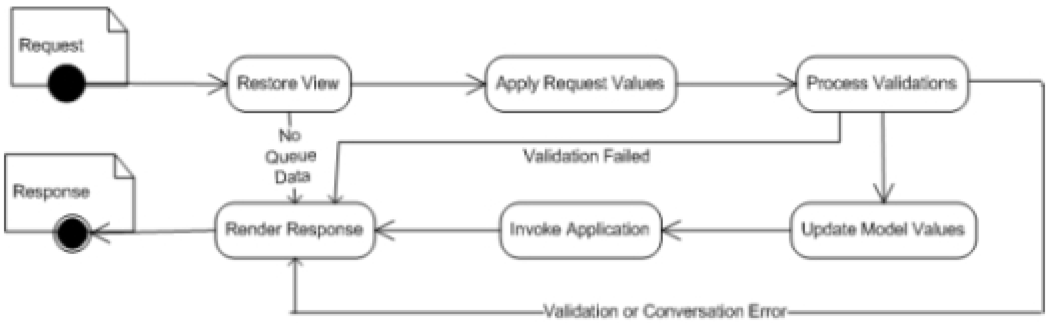


Figure 3.8.: JSF life cycle

1. Restore View The request from client sends to FacesServlet in the Restore View phase, which is the first phase. The main task in this phase is building

the Component Tree. There are two possible cases. First, if the request is a new request (done by URL submission), the Component Tree will be built and stored in FacesContext because, there will be no further value submitted with the request. In this case, the next phase will be Render Response. Second, if the request is request by postback (done by form submission), the Component Tree will be loaded from the FacesContext then go to the next phase, Apply Request Values, with submitted value.

2. Apply Request Values In this phase, the component objects in Component Tree will be iterated over and bind the request values to the responsible components.

3. Process Validations Since submitted values from the web pages are Strings, data validation is necessary. Attached validators perform correctness check on the submitted values. If the validation failed, the next phase will be Render Response and displays the previous page with error messages so the user can correct the invalid input values. Otherwise, the next phase will be Update Model Values. 4. Update Model Values The submitted values which pass the validation will be set in Managed Beans through setter methods.

5. Invoke Application In this phase, all the action methods which bind to the components (Command Buttons, Command Links) will be executed. These action methods will pass request values to the Business Logic Layer and return the result for the next phase.

6. Render Response After the get the result from the Business Logic Layer, the action methods will return String value. These return values will be used in page navigation defined in the Navigation Rules. Finally, the response page will be encoded and send back to the user.

3.2.4. Project Structure and Artifacts Overview

There are totally, ten artifacts in JSF prototype [Figure 3.8]. The artifacts divided into three types: Managed Beans, JSF pages, and configuration files.

There are three Managed Beans in the prototype. These Managed Beans responsible in supporting the user interface components of the JSF pages by providing attributes and methods, which bind to each component. For example, a textbox component binds to an name attribute. After do the form submission, the value user input in the textbox will be stored in name attribute in the Managed Bean via setter method (setName). After that, if the user refreshes the page, the name attribute that just stored in the Managed Bean will appear in the textbox. The value in the textbox is set by getter method in Managed Bean (getName). Also, the form submission component is bound to a method in Managed Bean.

There are two configuration files, one is the web.xml, the default configuration

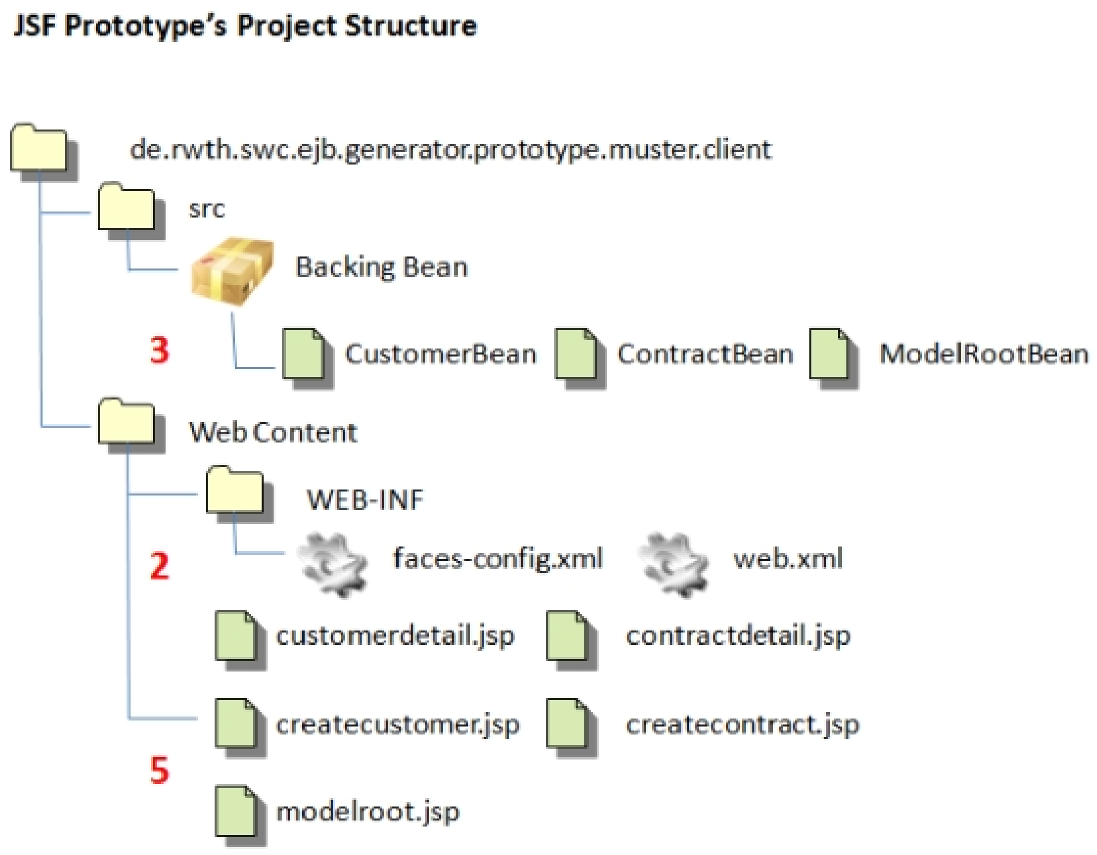


Figure 3.9.: JSF project structure

file, the other is faces-config.xml, the main configuration of JSF. In JSF 1.2, all Managed Beans are declared and mapped in the faces-config.xml. The complexity of the faces-config.xml is very low. Only three simple to five lines of code are needed in order to define a Managed Bean and its session scope. In JSF version 2.0, which aimed to be nearly zero-configuration framework, only one or two lines of annotation replaced, and these Managed Beans declaration is not required anymore. The Navigation rules are declared in faces-config.xml or its own file, navigation-rule.xml. The Navigation rules are rules to control the flow of the user interface navigation likes from which page, if the output is matched an string specified in a rule, move to which page defined in that rule, otherwise, move to an error page. These rules defined as an xml syntax with a diagram-like user interface support from IDE [Figure 3.9].

3.2.5. Migration Steps

These following steps must be done in order to migrate the system from the existing system to JSF prototype system. Only the presentation layer need to

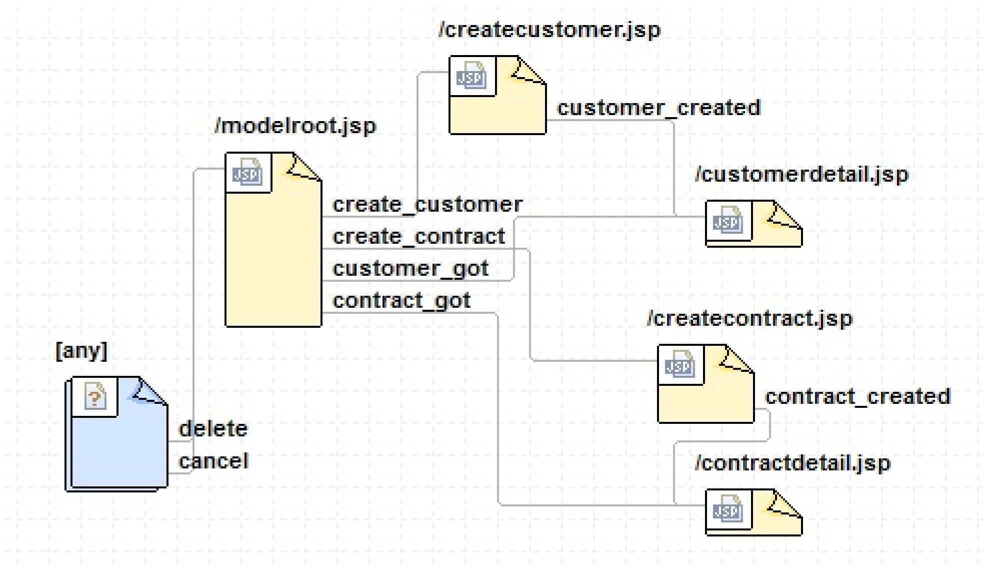


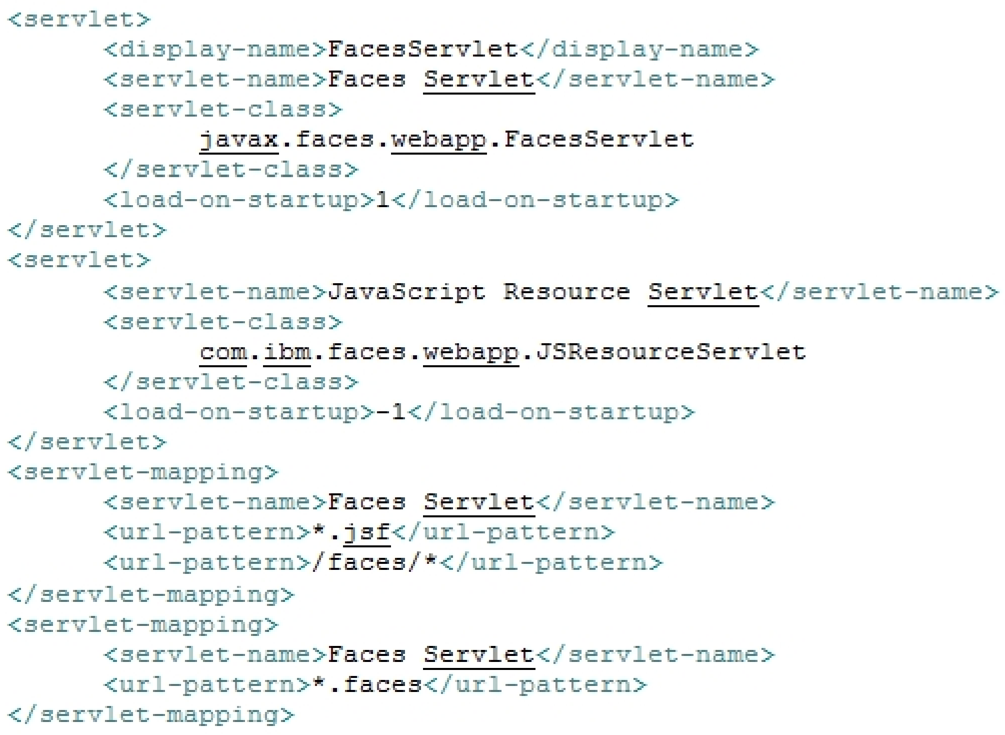
Figure 3.10.: JSF Navigation Rules (Eclipse IDE)

be changed. 1. Prepared the environment (JSF's prerequisites) 2. Copy all required libraries (JSF's prerequisites) to the client project in the WEB-INF's lib folder. 3. Add faces-config.xml to the WEB-INF folder. In some IDE such as Eclipse, faces-config.xml is added automatically, when the JSF project is created. 4. Create Managed beans and mapped them into faces-config.xml or by annotation (JSF2.0 or newer). 5. Create views (JSF pages) composed of user interface components, JSF core tags, and JSP/JSTL tags. 6. Defined Navigation rules in faces-config.xml or navigation-rule.xml using diagram view supported by IDE. 7. Configure web.xml to make it realized Faces Servlet (Front Controller) by adding these lines.

8. Publish EAR to the web server.

In order to fulfills the 2nd priority requirement, additional view technology, Facelets, is needed. Facelets is an official view technology for JSF. Facelets supports all JSF UI components, template inheritance, and focuses completely on building the JSF component tree, reflecting the view for a JSF application. Although, Facelets also aimed to improves JSF application in many aspect [Hig06], one example of those aspect is the synchronization between JSP and JSF lifecycle [Ber04]. Based on the example of [Figure 3.10], the top block of codes is the parent template inherited by the second, and the second inherited by the third. The result of this template inheritance is the great user interface reusability provided by Facelets, which answers the second priority requirements.

There are many additional libraries for Testing JSF application such as JSFUnit, InfoQ, and etc. All of them provides complete integration testing and unit testing inside the container, which provides the developer full access to the Managed beans. However, the independent unit testing on the view is not easy because



```
<servlet>
      <display-name>FacesServlet</display-name>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>
            javax.faces.webapp.FacesServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
</servlet>
<servlet>
      <servlet-name>JavaScript Resource Servlet</servlet-name>
      <servlet-class>
            com.ibm.faces.webapp.JSResourceServlet
      </servlet-class>
      <load-on-startup>-1</load-on-startup>
</servlet>
<servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>*.jsf</url-pattern>
      <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>*.faces</url-pattern>
</servlet-mapping>
```

Figure 3.11.: JSF's web.xml)

of JSF architecture does not provides complete separation of view and logic. Also, there are plenty of good IDE supports for JSF such as auto-completion, component tags validation, and simplified interface for configuration.

Since this prototype developed using JSF 1.2, migrates this prototype to the newer version might be one of future plan. In JSF 2.0, there are many improvements, which should be considered for future development such as:

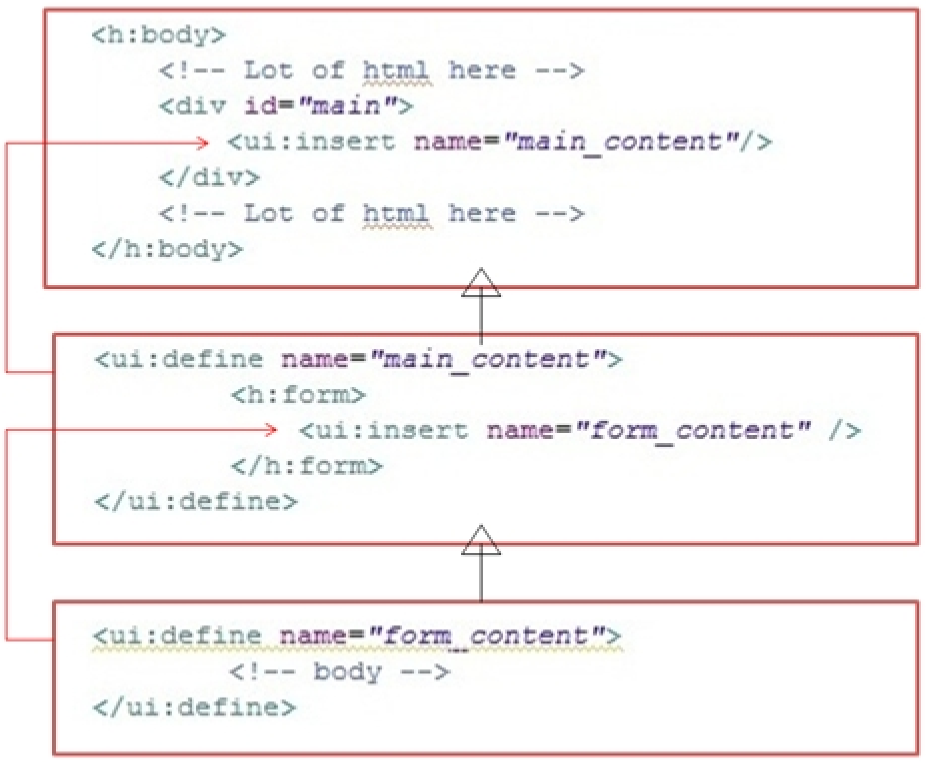
- Allow for zero configuration web applications. In JSF 2.0, configuration files (web.xml, faces-config.xml) is not required anymore. Annotations will be used for the necessary configuration data. For instance, instead of defines Managed Beans in faces-config.xml, only one line of annotation is needed in Managed Bean class.

- Add one more phase in request processing lifecycle to specifically for JavaScript and AJAX which means JavaScript library now becomes part of the JSF specification.

- Allow bookmarkable JSF pages and fix URL problems.

- Performance improvement.

- Support new standard for passing values from page to page.



```
<h:body>
    <!-- Lot of html here -->
    <div id="main">
        <ui:insert name="main_content"/>
    </div>
    <!-- Lot of html here -->
</h:body>
```

```
<ui:define name="main_content">
        <h:form>
            <ui:insert name="form_content" />
        </h:form>
</ui:define>
```

```
<ui:define name="form_content">
        <!-- body -->
</ui:define>
```

Figure 3.12.: JSF and Facelets inheritance example)

- Enable components to have a client-based lifecycle in addition to, or instead of the server-based request/response lifecycle. This lifecycle would enable user actions such as drag-and-drop, master-detail, and sub-dialogs on a single page interface web application.

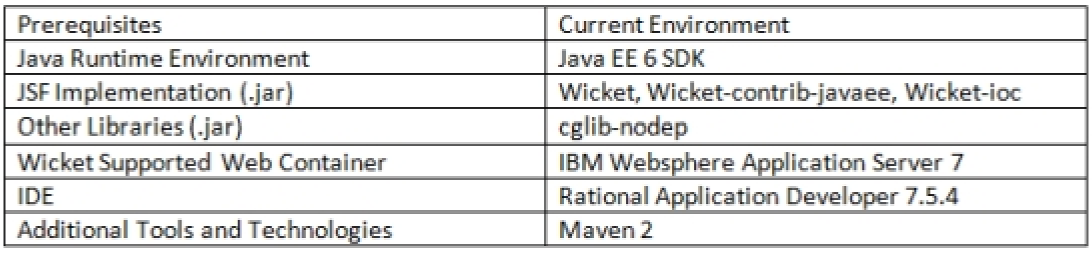
3.3. Apache Wicket

Similar to JSF, Wicket is a light-weight, component-based web application framework. Wicket's goal is to makes web application development looks more like desktop application development by eliminate the complexity of all server side state. Wicket manages all server side state automatically which means the developer will never directly dealing with requests/responses, URLs, or any sessions.

With complete separation of view and logic, a POJO data model, and no XML required (zero-configuration), Wicket simplified the web development [Nad08] by swapping the boiler-plate, complex debugging and brittle code for powerful, reusable components written in plain Java and HTML.

The latest statistic shows that Wicket is one of the most remarkable framework. With increasing user every year, well-known as one of the most active community, and frequently update released, makes Wicket becomes one of the prototype in this master thesis.

3.3.1. Working Environments and Tools



Prerequisites	Current Environment
Java Runtime Environment	Java EE 6 SDK
JSF Implementation (.jar)	Wicket, Wicket-contrib-javaee, Wicket-ioc
Other Libraries (.jar)	cglib-nodep
Wicket Supported Web Container	IBM Websphere Application Server 7
IDE	Rational Application Developer 7.5.4
Additional Tools and Technologies	Maven 2

Figure 3.13.: Wicket Prerequisites)

First, Maven 2 is needed because, the Wicket project structure is based on Maven Archetype structure generation. Maven generates the project structure and managed the dependency for Wicket. Second, Wicket-ioc library is not required, but in order to use the dependency injection to communicates with the Application Facade from the presentation layer, it is needed.

3.3.2. Architecture

Wicket's architecture is very simple. It is mainly composed of HTML templates bind tightly to POJOs with the same name [Figure 3.11]. These paired HTML templates and POJOs represent web pages. Minimum number of artifact to build a page is two: one HTML page, and a POJO. In consequence, huge amount of artifacts are produced. However, Wicket's component-based architecture greatly reduces the complexity of artifacts. The developer may use basic Wicket components or even, create custom components using the same method as building a page. As a result, more than half of the artifact especially, HTML files have few simple lines of components' interface declaration, and POJOs are well-structured, highly reusable, and easy to understand with inheritance structure.

The flow is quite simple. The Request Cycle acts like a front controller, delegates the request to the corresponding page, then the user interface components in HTML template will be initialized by POJO bound to that HTML.

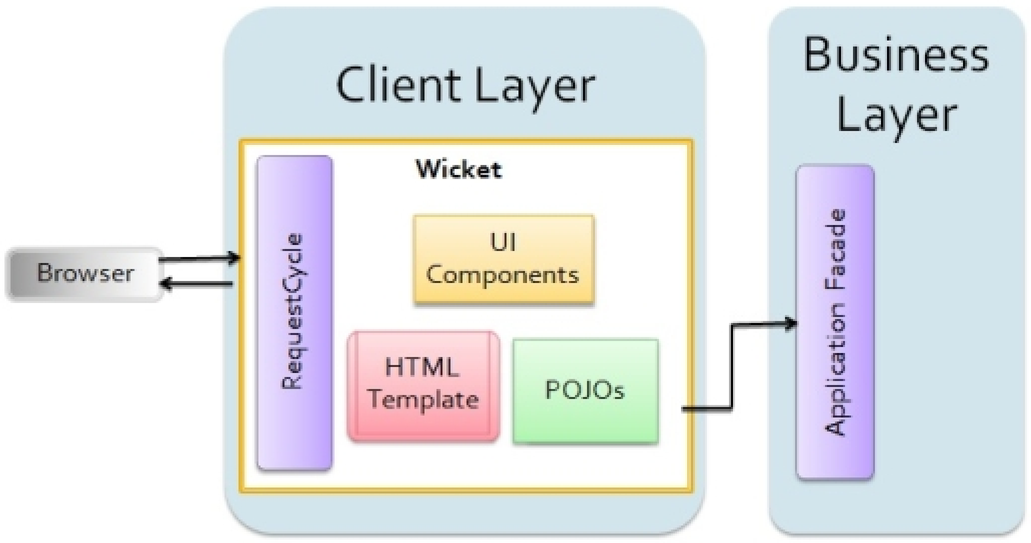


Figure 3.14.: Wicket integrated architecture)

3.3.3. Basic Concepts and Life Cycle

Wicket's life cycle composed of three main steps: application loading, request processing, and rendering [Figure 3.12]. Start with the Wicket filter is initialized and Request Cycle object, which handles the request delegation is created. The request from user will be forced to passes the pre-request processing and pre-render components before the component rendering. At the component rendering state, if the component checking is set to enable in web.xml, the component render status checking will be performed. During the rendering state, if any model value is changed, the request will be forced to the pre-render state again. Finally, the response will be passed through the post-render components and post-request processing before sends back to the user.

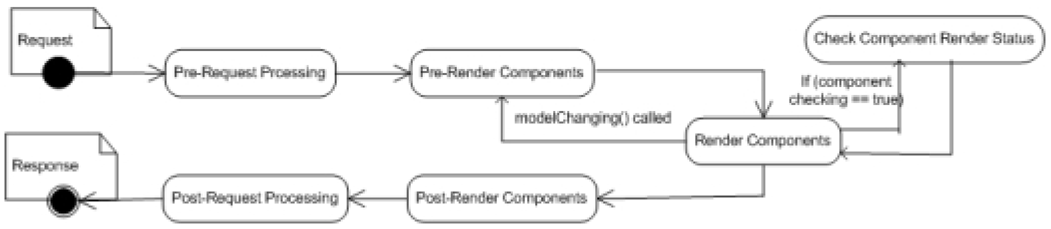


Figure 3.15.: Wicket life cycle)

3.3.4. Project Structure and Artifacts Overview

There are totally, twenty-eight artifacts in Wicket prototype [Figure 3.13]. The artifacts divided into four types: custom components, pages, data provider and configuration files.

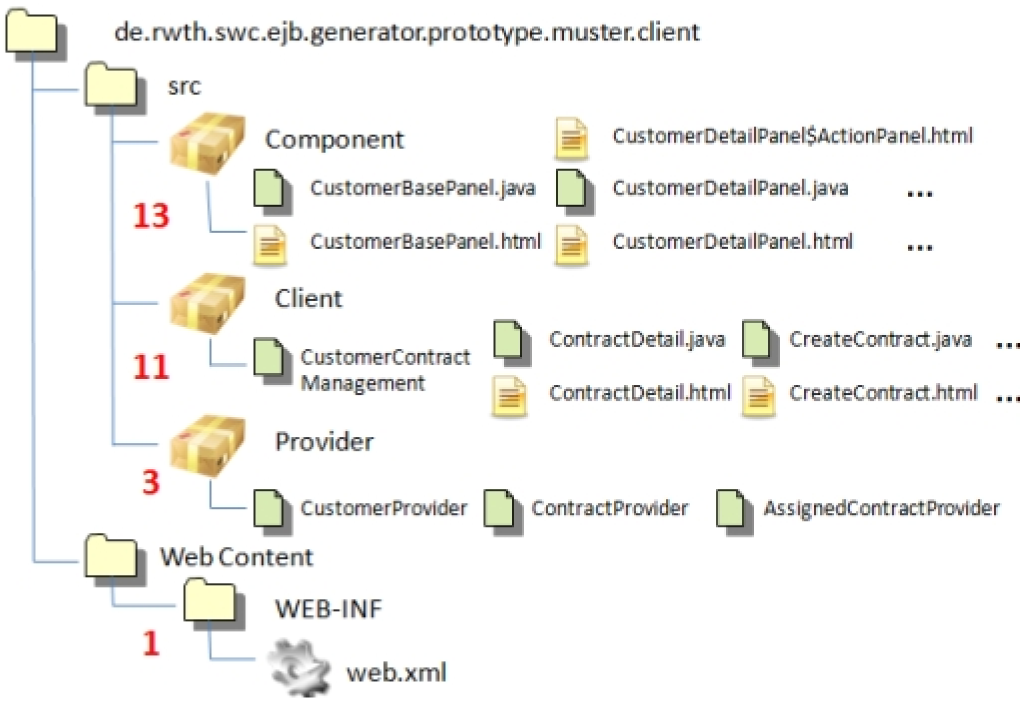


Figure 3.16.: Wicket project structure)

The Component package contains custom components of the prototype such as data table displaying name of the customers and contract identifier of the contracts bind to each customer, or a column for any table which contains hyperlinks to edit and view the detail of the object of that row. In order to reuse the components on the pages in Client package, those components given as the example need to be placed on a basic wicket component, Panel. As described in the previous session, one page (component) consists of at least, one HTML template and one POJO. Some component may has another private small sub-component declared inside the POJO. In this case, another HTML template under the name format: [POJO name]$[Sub-Component Name].html is required.

The Client package contains the prototype pages. Those pages are simply the combination the custom components declared in Component package and other small details. Also, Wicket filter class, CustomerContractManagement is needed to be implemented and mapped in the configuration file. The filter class responsible in overall project control, for example, enable dependency injection for the Application Facade and allow user to access the system with nicer URL.

There is only one configuration file, which is the default web application configuration file, web.xml. Only few lines to declared the Wicket filter class and mapped the Application Facade EJB reference is needed.

3.3.5. Migration Steps

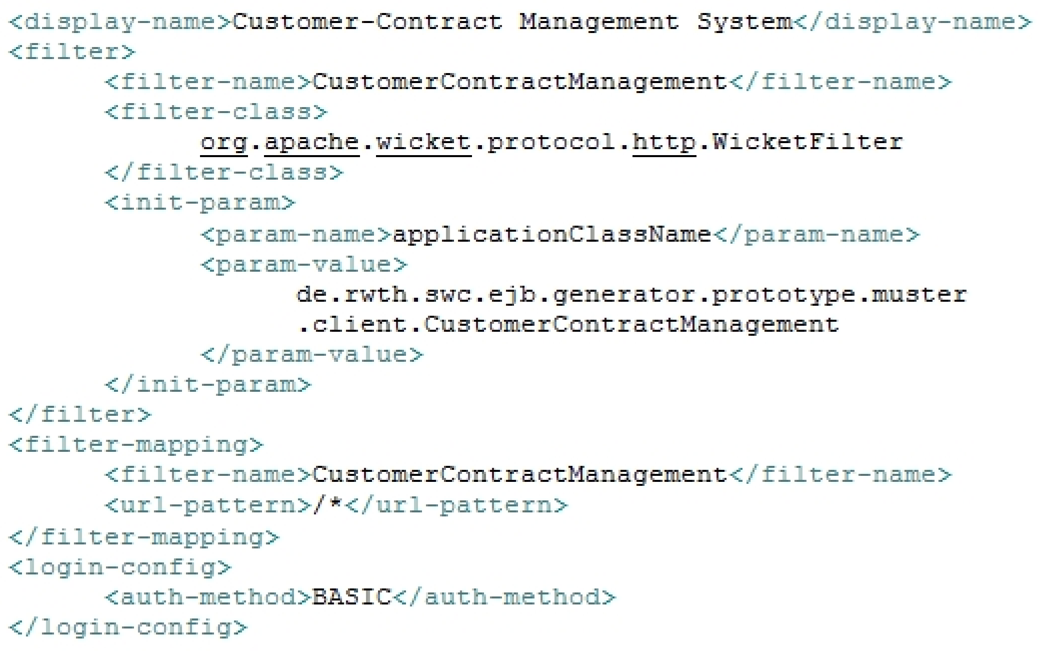
These following steps must be done in order to migrate the system from the existing system to Wicket prototype system. Only the presentation layer need to be changed. 1. Prepared the environment (Wicket's prerequisites). 2. Install Maven 2. 3. Generate a Wicket project using Maven Archetype followed this generation command :

mvn archetype:create -DarchetypeGroupId=org.apache.wicket -DarchetypeArtifactId=wicket-archetype -quickstart -DarchetypeVersion=[wicket version number] -DgroupId=[group id] -DartifactId=[project name] -DarchetypeRepository=https://repository.apache.org/ -DinteractiveMode=false

4. Import the project into the IDE and copy all required libraries (Wicket's prerequisites) to the client project in the WEB-INF's lib folder.

5. Create HTML template and POJOs with the same name, or defined own custom components.

6. Configure web.xml to make it realized Wicket filter (Front Controller) by adding these lines.

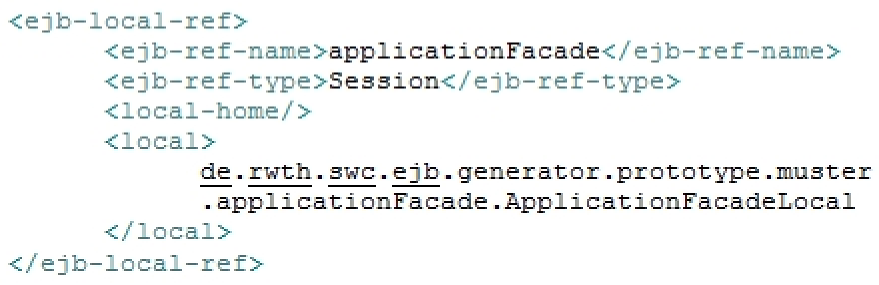


```
<display-name>Customer-Contract Management System</display-name>
<filter>
      <filter-name>CustomerContractManagement</filter-name>
      <filter-class>
            org.apache.wicket.protocol.http.WicketFilter
      </filter-class>
      <init-param>
            <param-name>applicationClassName</param-name>
            <param-value>
                  de.rwth.swc.ejb.generator.prototype.muster
                  .client.CustomerContractManagement
            </param-value>
      </init-param>
</filter>
<filter-mapping>
      <filter-name>CustomerContractManagement</filter-name>
      <url-pattern>/*</url-pattern>
</filter-mapping>
<login-config>
      <auth-method>BASIC</auth-method>
</login-config>
```

Figure 3.17.: Wicket's web.xml part1)

7. Map the Application Facade EJB reference by adding these following lines to web.xml :

8. Publish EAR to the web server.



```
<ejb-local-ref>
      <ejb-ref-name>applicationFacade</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <local-home/>
      <local>
            de.rwth.swc.ejb.generator.prototype.muster
            .applicationFacade.ApplicationFacadeLocal
      </local>
</ejb-local-ref>
```

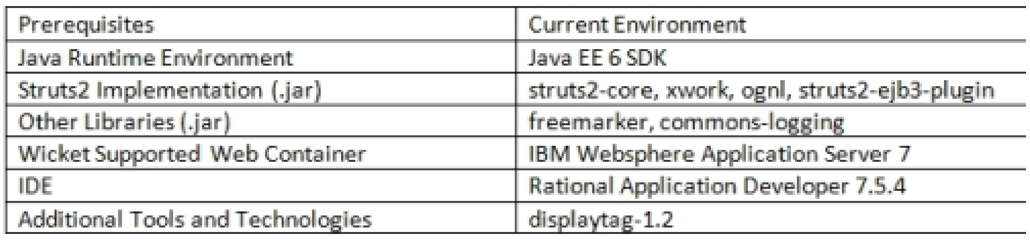
Figure 3.18.: Wicket's web.xml part2)

3.4. Apache Struts2

Apache Struts2 is a enhanced version of a combination of two frameworks: Apache Struts, and OpenSymphony WebWork. Struts1, the original version of Struts2, once a very first Java web framework, which revolutionized Java web development and resulted as thousand of Struts-based application deployed worldwide. Even though, Struts2's architecture and concept have improved a lot from Struts1 [Str11], but the goal and purpose are still the same. Struts is an request/response, action-based MVC framework, which makes web application development easier, faster, and more productive.

Struts2 is totally, different with Struts1, the core features are all implemented with interceptors, value stack concept, OGNL expression, and Struts2 tags to work around the application data, many annotation and conventions, which makes Struts2 is more easy to use and understand.

3.4.1. Working Environments and Tools



Prerequisites	Current Environment
Java Runtime Environment	Java EE 6 SDK
Struts2 Implementation (.jar)	struts2-core, xwork, ognl, struts2-ejb3-plugin
Other Libraries (.jar)	freemarker, commons-logging
Wicket Supported Web Container	IBM Websphere Application Server 7
IDE	Rational Application Developer 7.5.4
Additional Tools and Technologies	displaytag-1.2

Figure 3.19.: Struts2 Prerequisites)

The Struts2-core is the framework library. As the prerequisite for Struts2, the framework itself is built on XWork2 framework and it used Object Graph Notational Language (OGNL) to access object properties. Easiest way to gain access to the Application Facade in the business using dependency injection, an unofficial library, struts2-ejb3-plugin is needed [ldt11]. A famous view technology, Freemarker is used to create UI tag templates by default. There are several view option integration supported by Struts2 such as JSP/JSTL, Tiles, Velocity, Excel, XSL, PDF, and etcetera. Finally, the only additional library, displaytag 1.2 was brought into Struts2 prototype because, this widely-used library provides datatable-like component, which is powerful, good-looking, and easy-to-manage and manipulate the data and graphic user interface.

3.4.2. Architecture

In standard web application development, the client submits the request to web server via web form (view), processes by Servlet, interacts with database, and response back as an web form. This approaches are often considered inadequate for large project because, it mixes business logic with presentation and makes maintenance and testing difficult.

The goal of Struts is to separate the model from view and controller [Figure 3.14]. Struts2 introduced Action bean as the controller to facilitate the writing of templates for the view or presentation layer (typically in JSP, but XML/XSLT, Tiles, FreeMarker and Velocity are also supported). The web application programmer is responsible for writing the model code, and for creating a central configuration file struts-config.xml that binds together with model, view and controller.

Requests from the client are sent to the controller in the form of "Actions" defined in the configuration file; if the controller receives such a request it calls the corresponding Action class that interacts with the application-specific model code. The model code returns an "ActionForward", a string telling the controller what output page to send to the client. Information is passed between model and view in the form of special JavaBeans. A powerful custom tag library allows it to read and write the content of these beans from the presentation layer without the need for any embedded Java code.

Interceptor is the core concept of Struts2, which provides great code redundancy reduction. Many Action beans share common concern. Several Action beans need similar input validation. Some needs a file upload to be pre-processed. Another Action beans might need double form submission protection. Interceptors can execute code before and after the Action bean is invoked. Features like double-submit guards, type conversion, object population, validation, file upload, page preparation, and more, are all implemented with the help of Interceptors. Each and every Interceptor is pluggable, so developers can decide exactly which features an Action bean needs to support. Also, if a lot of Action beans are plugged with the same pattern of interceptors, Interceptor stack can be defined in the struts.xml to provide even more convenient. For example, every Action beans which related to very important task need to apply double-submit guards, user session validation, and encryption interceptor. Instead of spending three lines to apply all required interceptor on every Action beans, registering an interceptor stack contained three required interceptors, and spent only a line for applying the stack to the Action beans is more productive.

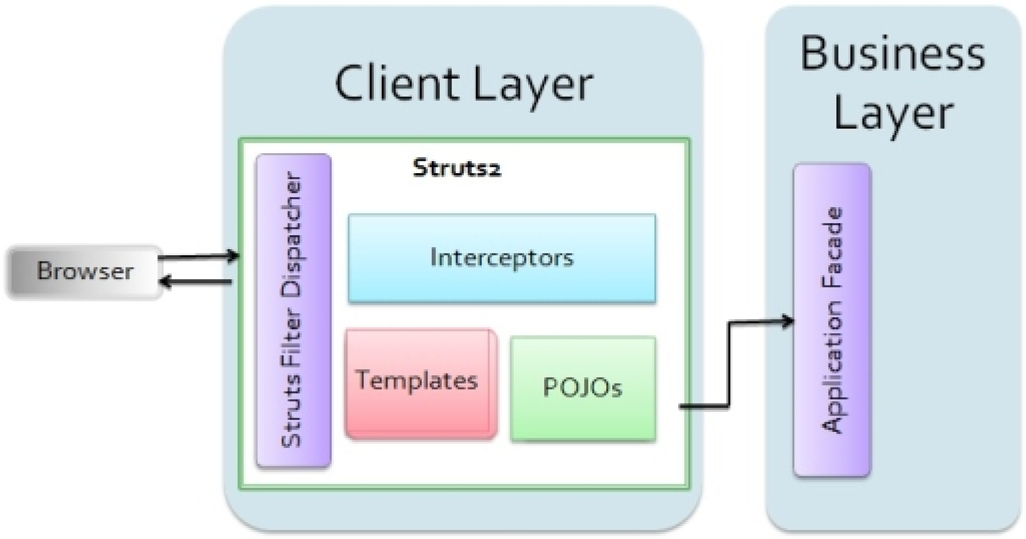


Figure 3.20.: Struts2 integrated architecture)

3.4.3. Basic Concepts and Life Cycle

These following five steps describe the request processing lifecycle of Struts2 showed in Figure 3.15 : 1. The request is generated by user and sent to Servlet container. 2. The Servlet container invokes Servlet Filter Dispatcher, which get the name of the responsible action from the configuration file (struts.xml). 3. One by one Interceptors are applied before calling the action. Interceptors performs tasks such as logging, validation, file upload, double-submit guard, and etc. 4. Action is executed and the Result is generated by Action. 5. The output of Action is rendered in the view (JSP, Freemarker, Velocity, etc) and the result is responded back to the user.

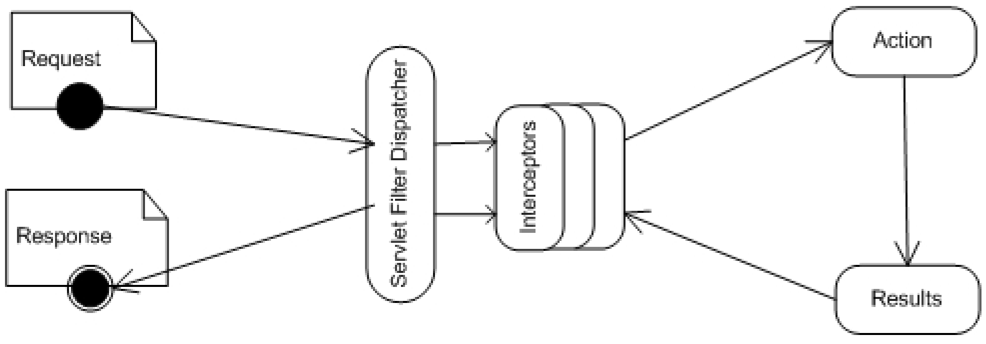


Figure 3.21.: Struts2 life cycle)

3.4.4. Project Structure and Artifacts Overview

There are ten artifacts produced by Struts2 framework [Figure 3.16]. Those artifacts categorized into three category: Action beans, views (JSPs), and configuration files.

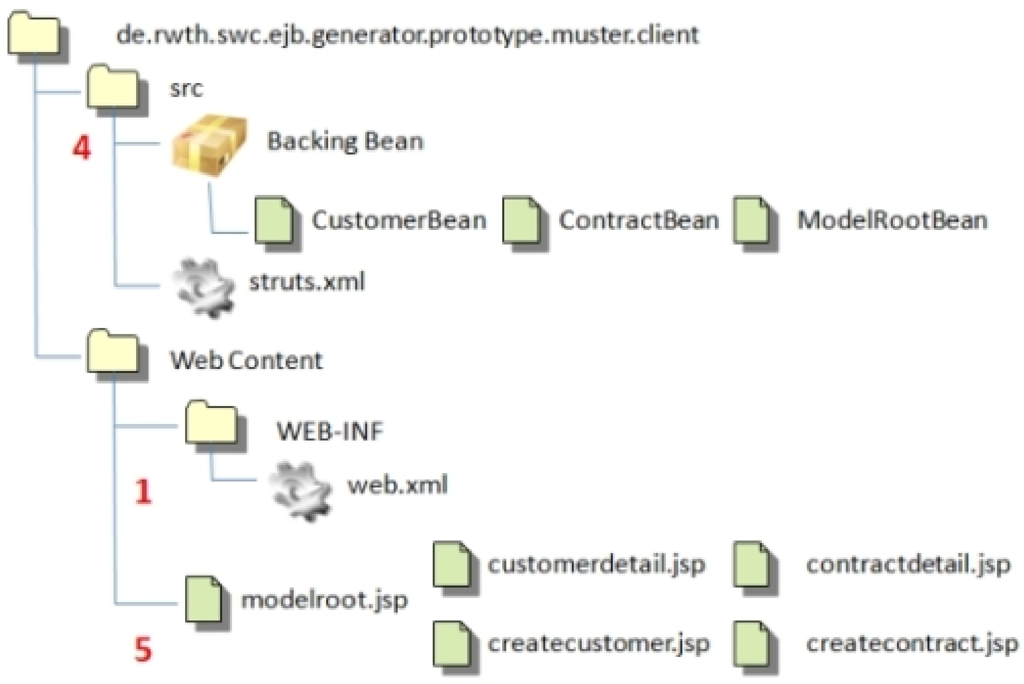
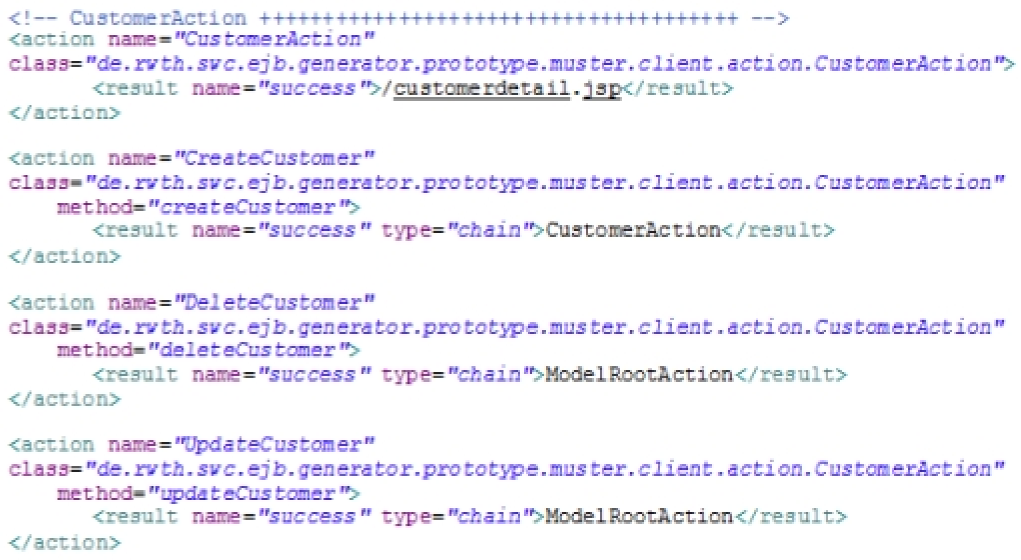


Figure 3.22.: Struts2 project structure)

There are three Action Beans in the prototype. These Action Beans are similar to Servlets. The differences are these Action beans also responsible in supporting the views of the JSP pages by providing attributes and methods, which bind to struts tags and displaytag. For example, a datatable provides by displaytag binds to customer name and contract identifier attribute. After process the first page request, the datatable shows all customer name and contract identifier. After do the create customer form submission, the value user input in the textbox will be stored via setter method (setName). After that, if the user refreshes the page contained datatable, the new customer value is also displayed in the datatable. The value shows in the datatable is set by getter method in Action Bean (getName). Also, the form submission tag is bound to a method in Action Bean. Unlike JSF, every method that bound to the view must be registered in struts.xml.

There are two configuration files, one is the web.xml, the default configuration file, the other is struts.xml, the main configuration of Struts2. All Action beans, methods inside Action beans which bound to the view, and page flow are all registered in struts.xml without any supports. For example, based on struts.xml below, there is only one Action bean registered, which is Customer-Action, if there is nothing wrong with the attributes initialization, the client will be redirected to customerdetail.jsp. Create, Delete, and UpdateCustomer are CustomerAction's methods registration. In consequences, these three methods are allowed to be called on any view.



```xml
<!-- CustomerAction ++++++++++++++++++++++++++++++++++++++++ -->
<action name="CustomerAction"
class="de.rwth.svc.ejb.generator.prototype.muster.client.action.CustomerAction">
       <result name="success">/customerdetail.jsp</result>
</action>

<action name="CreateCustomer"
class="de.rwth.svc.ejb.generator.prototype.muster.client.action.CustomerAction"
    method="createCustomer">
       <result name="success" type="chain">CustomerAction</result>
</action>

<action name="DeleteCustomer"
class="de.rwth.svc.ejb.generator.prototype.muster.client.action.CustomerAction"
    method="deleteCustomer">
       <result name="success" type="chain">ModelRootAction</result>
</action>

<action name="UpdateCustomer"
class="de.rwth.svc.ejb.generator.prototype.muster.client.action.CustomerAction"
    method="updateCustomer">
       <result name="success" type="chain">ModelRootAction</result>
</action>
```

Figure 3.23.: Struts2 action mapping xml)

3.4.5. Migration Steps

1. Prepared the environment (Struts2's prerequisites)

2. Copy all required libraries (Struts2's prerequisites) to the client project in the WEB-INF's lib folder.

3. Add struts.xml to the src folder. In some IDE such as Eclipse, struts.xml is added automatically, when the Struts project is created.

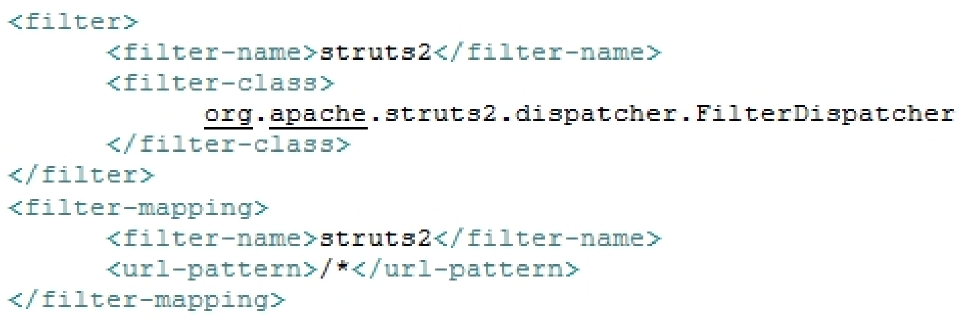
4. Create Action beans and mapped them into struts.xml or by annotation. In our prototype, Action beans were mapped in the struts configuration file because, the Action mapping also include many other parameter such as method name, result name, result type, and the result page, which totally, sums up to 3-4 lines of annotation scattered on top of the classes and methods declaration. These annotations considered as one factor that increases artifact complexity.

5. Create views (JSP pages) composed of Struts core tags, 3rd party libraries

(displaytag), other view technology(optional) and JSP/JSTL tags.

6. Defined the page flow in struts.xml.

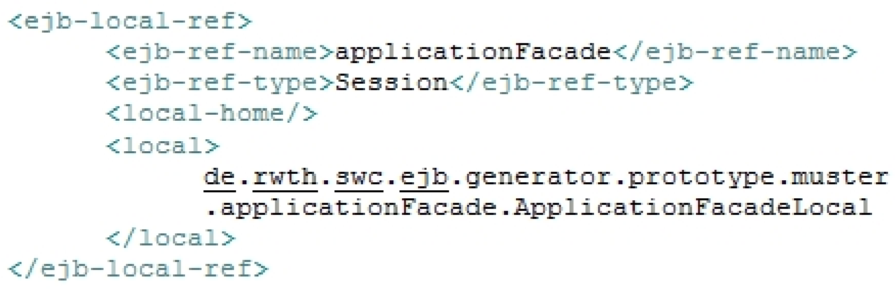
7. Configure web.xml to make it realized Struts Filter Dispatcher (Front Controller) by adding these lines:



```
<filter>
      <filter-name>struts2</filter-name>
      <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
      </filter-class>
</filter>
<filter-mapping>
      <filter-name>struts2</filter-name>
      <url-pattern>/*</url-pattern>
</filter-mapping>
```

Figure 3.24.: Struts2's web.xml part1)

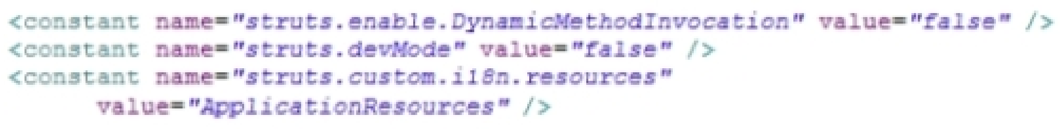
8. Map the Application Facade EJB reference by adding these following lines to web.xml :



```
<ejb-local-ref>
      <ejb-ref-name>applicationFacade</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <local-home/>
      <local>
            de.rwth.swc.ejb.generator.prototype.muster
            .applicationFacade.ApplicationFacadeLocal
      </local>
</ejb-local-ref>
```

Figure 3.25.: Struts2's web.xml part2)

9. Adding these lines to struts.xml is a requirement for WAS 7.0 integration:



```
<constant name="struts.enable.DynamicMethodInvocation" value="false" />
<constant name="struts.devMode" value="false" />
<constant name="struts.custom.i18n.resources"
      value="ApplicationResources" />
```

Figure 3.26.: Struts2's web.xml part3)

10. Publish EAR to the web server.