e-Assessment of Workplace Health & Safety

Martin René Pedersen

Kongens Lyngby 2007
IMM-Eksamensprojekt-2007-52

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Summary

Danish companies with employees are obligated to assess the health and safety environment work. The assessment is a time demanding but important task.

This thesis documents a masters project aiming at developing a web based assessment system for this work. The system should comply with the working environment law in force and at the same time be simple and foreseeable to the user.

The thesis introduces Java Enterprise Edition as the platform technology, Extreme Programming as the development methodology and the planning and implementation of the first release of the e-assessment system.

Resumé

Alle virksomheder med én eller flere ansatte skal måle og vurdere håndteringen af virksomhedens arbejdsmiljøog sikkerhed - en APV. At gennemføre en APV proces er et tidskrævende men vigtigt arbejde.

Denne rapport dokumenterer et eksamensprojekt med sigte påat udvikle et web baseret system til håndtering af virksomhedens APV. Systemet skal opfylde gældende krav til APV og samtidig skal systemet være simpelt og let overskueligt for den almingelige bruger.

Rapporten introducerer Java Enterprise Edition som teknologi, Extreme Programming som udviklingsmetode og planlægning samt gennemførelse af første iteration af APV systemet.

Preface

This thesis was prepared at Thermo Fischer Scientific, Roskilde and is the documentation of a project aiming at developing a electronic system for assessment of workplace health and safety. The system should be web based and the usability of the user interface is crucial to the employees.

The thesis is divided into three parts.

Part 1 describes the Java Enterprise Edition platform. The platform includes various technologies as servlets, Java Server Pages, Java Server Faces and Enterprise Java Beans which makes it a very powerful platform for large web and enterprise applications.

The application server used in this project is JBoss Application Server which includes the Apache Tomcat server for servlets and Java Server Pages. The server also includes the new EJB3.0 specification for Enterprise Java Beans and along with the JBoss Seam framework for connecting Enterprise Java Beans with Java Server Faces this makes a powerful runtime platform.

Part 2 describes Extreme Programming which is a relatively new software development methodology. It was devised in the late 1990s combine well known practices of software development in a new way and with new practices. Making high quality software should be fun and rewarding - and this is the main goal of Extreme Programming.

Part 3 describes a web based system for e-assessment of workplace health and safety. The system is described and planned using the Extreme Programming Methodology. The project is divided into three iterations and the first is implemented in Java Enterprise Edition

As the basic aspects of Java Enterprise Edition is described in the first part no knowledge of this technology is required to read the report, but as the platform is quite complex the reader is presumed to know the Java platform at an advanced level.

Lyngby, June 2007

Martin René Pedersen

Acknowledgements

A lot of people have been very helpful during this project.

I would like to thank the employees and the management at Thermo Fischer Scientific (Nunc A/S) for participating in this project. Projects with students always has a risk as you never now what the result will be. Eventually I think we will come up with a useful and timesaving solution.

I would especially like to thank the safety organization for excellent feedback and criticism as the project and the web layout were presented to them at a meeting. I would also like to thank Mogens Andersen and Asger Dahl - the health and safety managers at Thermo Fischer Scientific. Their help and patience were crucial for the understanding of the working environment legislation and the requirements for the assessment system.

I would also like to thank my supervisor Jens Thyge Kristensen who always has more several points of view and useful feedback.

And last but not least I would like to thank Tina for her never-ending support and patience with me.

Contents

Part I

Java Enterprise Edition as Technology

Chapter 1

Java Enterprise Edition

Java Enterprise Edition is one of the most comprehensive and flexible platforms available. It offers the entire Java API (which is enormous) and has several opportunities for each part of the application. The latest version is Java Enterprise Edition 5 released fall 2006 and features at lot of updates and additions compared to the rather old version 2. In the following JEE will be used and refers to Java Enterprise Edition 5.

Sun provides the JEE platform for developing applications but not the middleware connecting the platform with the custom made application. In this project a web server[1] is needed and there are several possibilities from different vendors. JBoss Application Server is one of the most popular and widely used JEE web servers and will be the one used for this project.

Figure 1.1 shows an overview of the entire JEE platform. The grey arrows indicates how calls from clients are processed through the system. The following sections describes the most important technologies of JEE - but is by no means exhaustive as that would require several hundreds of pages. More information on the specific technologies is found in the books and websites referenced in the bibliography.

[1]There are other servers providing a platform for running traditional client/server applications as e.g. a financial system with a server and dektop client.

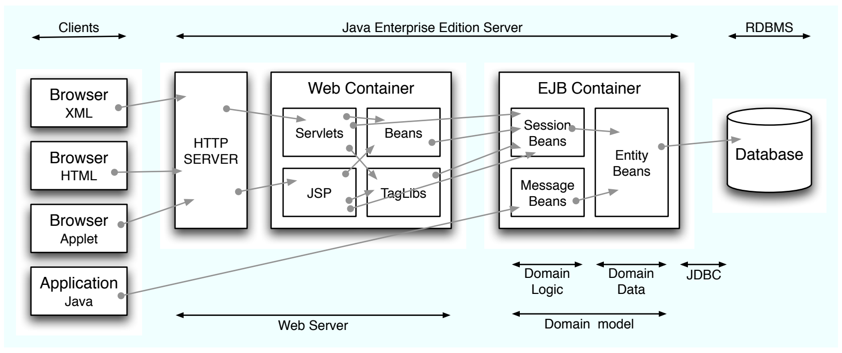


Figure 1.1: An overview of Java Enterprise Edition.

1.0.1 Architecture of a JEE application

In the design of JEE web applications there are two commonly used architectural patterns referred to as *Model 1* and *Model 2*.

Model 1 is the simpler of the two. When a request is made by a client either a servlet or a JSP processes the request. It is responsible for validating input, handling business logic and generating the response to the client. The pattern is very simple and not suitable for large applications.

Model 2 builds on the Model-View-Control (MVC for short)[2] architectural pattern. In the pattern a client makes a request to a controller implemented as a servlet. Based on the request the controller decides which view to pass the request to. The view may be either a Java Server Pages (JSP) or a Java Server Faces (JSF) which again invokes the methods of the model which are made up of Enterprise Java Beans (EJB). The model might access a database and the produces a response object which is passed to the view. The view then generates a response which is sent back to the client.

[2]The MVC pattern was first described by Trygve Reenskaug in 1979 and used in the Smalltalk language. The pattern separates the layers of an application into the *model* containing the business logic of the domain, the *view* which renders the work of the model into e.g. a user interface and a *controller* which processes events, invokes changes on the model and responds to the events

1.1 Web application

Web application might not be the best heading for this section as this only describes servlets, Java Server Pages and Java Server Faces whereas Enterprise Java Beans are covered in the next section. Nevertheless the three technologies are packed and deployed inside the Web container of the JEE server and they are not dependent on the EJBs (e.g. using the Model 1 pattern does not include EJBs), so they actually are able to form an entire web application.

1.1.1 Servlets

One of the most central elements in the JEE platform is servlets. A servlet is a java class which extends the web server and enables the server to respond to new kinds of requests. This is accomplished by letting the java class implement the `Servlet` interface through extending the abstract class `GenericServlet`. As many applications are web applications a special abstract class `HttpServlet` which extends the `GenericServlet` class is used. The `HttpServlet` class implements some HTTP specific functionality (by overriding the verb+service()+ method of the `GenericServlet` class) and has four methods supporting requests of the HTTP protokol [3]. The four methods are `doGet`, `doPost`, `doPut` and `doDelete` - the latter two are seldom used as most requests are GET or POST requests. The application servlet must override at least one off these methods.

Servlets has a lot of advantages over e.g. web scripting languages like *php*, *asp* and *cgi*. As they the are written in Java they are portable between operation systems, the are powerful through the Java API, they are reliable through the secure memory model of Java and they are an elegant way of developing as they are object oriented and modular. As they extends the web server there is a close integration with the server and the server handles a lot of the trivial stuff like sessions, cookies and logging. There are disadvantages though. If making small applications things might get unnecessarily complicated bringing Java into the development. Generating output for the response is quite tedious (se the examples later) but this may be solved using Java Server Pages as in section 1.1.2.

Servlets were originally developed by Anselm Baird-Smith at Sun Microsystems and later elaborated into a specification by Satish Dharmaraj.

[3]See http://www.w3.org/Protocols for more information on the HTTP Protocol

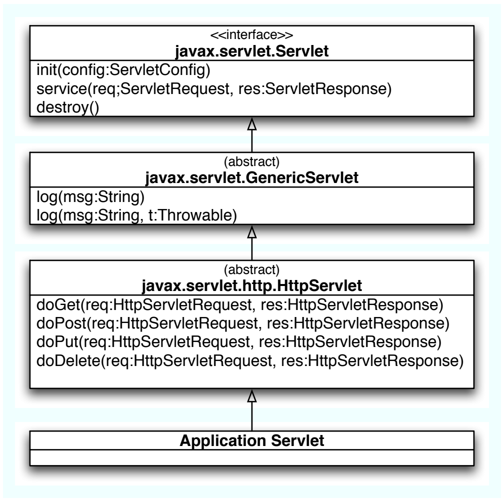

<<interface>>
javax.servlet.Servlet
init(config:ServletConfig)
service(req;ServletRequest, res:ServletResponse)
destroy()
(abstract)
javax.servlet.GenericServlet
log(msg:String)
log(msg:String, t:Throwable)
(abstract)
javax.servlet.http.HttpServlet
doGet(req:HttpServletRequest, res:HttpServletResponse)
doPost(req:HttpServletRequest, res:HttpServletResponse)
doPut(req:HttpServletRequest, res:HttpServletResponse)
doDelete(req:HttpServletRequest, res:HttpServletResponse)
Application Servlet

Figure 1.2: The class hierarchy of the servlet application. Only methods interesting for the lifecycle of the servlet are included.

1.1.1.1 Lifecycle of a servlet

The servlet lives inside the servlet container and is therefore instantiated by the container. This is done either when the servlet container starts up, when the servlet is deployed to the container or the first time a request is sent towards the servlet. The chosen strategy is different for different container vendors.

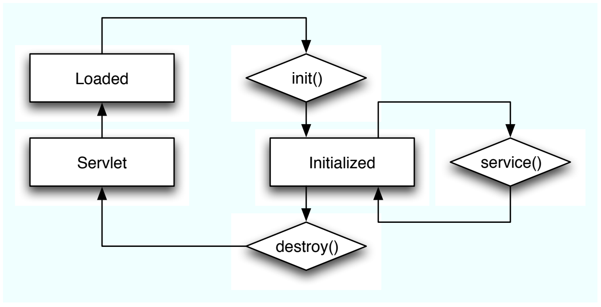


Figure 1.3: The lifecycle of a servlet

Once the servlet has been instantiated it is kept in memory and is in Loaded state. At some point the servlet is initialized by the container which calls the `init(ServletConfig config)` method on the servlet. The method will save the `config` object and call the overloaded no argument `init()` method - and the Servlet is now in Initiated state. Just like the constructor of a simple java class the init method is guaranteed to be called only once per instance.

As the servlet is initialized it is ready to service clients. When the container receives a request it wraps the request in a `HttpServletRequest` object, creates `HttpServletResponse` object and calls the `service(ServletRequest req, ServletResponse res)` method which in turn calls the appropriate `doXXX` method. This method generates a response, wraps it into the `HttpServletResponse` object and returns it to the container. The container unwraps the object and generates the response to the client. `HttpServletRequest` and `HttpServletResponse` are Java interfaces whereas the specific class implementations are vendor specific and unknown to the servlet developer.

A servlet is only instantiated once in the container and the initiated servlet then handles request for all clients which means a servlet may serve several concurrent requests from multiple clients.

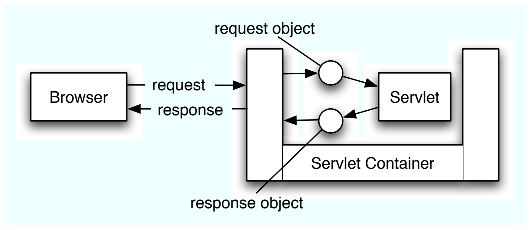


Figure 1.4: The container handles the requests and responses between the client and the servlet.

At some point the servlet instance is destroy by the `destroy()` method called by the container. Once the method has been called the instance will not call it again and at some point, the instance is garbage collected and the servlet enters the Unloaded state (denoted Servlet in figure 1.3).

1.1.1.2 Examples of servlets

Listing 1.1 shows a simple servlet which generates a list of questions from a specific category. From the response the servlet instantiates a `PrintWriter` object which is a text based output response for the browser. A table with all questions is generated and printed to the object. The output is flushed manually at the end of the request which is not needed in this case as it is automatically flushed when the `PrintWriter` object is destroyed.

The servlet only responds to GET requests as only the `doGet` method is im-

```
public class printQuestionsServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
        HttpServletResponse response)
      throws IOException, ServletException {

      // Get category id from request (parsed in URL)
      String categoryId = request.getParameter("categoryId");

      // Get questions for category with categoryId (pseudo method)
      List<Question> questionList = getQuestionsFromDatabase(categoryId);

      // Get a PrintWriter to write output html back to container
      PrintWriter output = response.getWriter();
      response.setContentType("text/html");

      // Print out questions in a table
      output.println("<table>");
      for (Question qs : questionList) {
         output.println("<tr><td>");
         output.println(qs.getText());
         output.println("</td></tr>");
      }
      output.println("</table>");

      // Flush output
      output.flush();
   }
}
```

Listing 1.1: *printServlet.java* - example of servlet printing out a list of questions.

plemented. If a POST request is send to the servlet the container will respond with a *HTTP 405 Method Not Allowed* error.

As mentioned earlier the servlet may also function as a controller in the Model 2 pattern. Listing 1.2 shows an example of a controller servlet. The controller responds to both GET and POST requests as both calls the `processRequest` method. The method find a parameter `action` from the request and loops through a set of *if...then...else* statements to find a matching case. If the matching case was `viewQuestionsFromCategory` the `categoryId` parameter is picked from the request and a list of questions is generated e.g. from a database. The list is set as an attribute on the response object. At the end of the servlet the destination page is set on the request object to forward the container focus to the destination page (which will be an error page by default if not changed during the *if...then...else*).

1.1.1.3 Declaring the servlet

The servlet must be declared for the servlet container to use it and additionally the servlet must be mapped to an url. This is done in an xml format in `web.xml`.

```
public class ControllerServlet extends HttpServlet {
   private static final String ACTION_KEY = "action";
   private static final String ERROR_PAGE = "/error.jsp";

   public void doGet(HttpServletRequest request,
         HttpServletResponse response)
      throws IOException, ServletException {
      processRequest(request, response);
   }

   public void doPost(HttpServletRequest request,
         HttpServletResponse response)
      throws IOException, ServletException {
      processRequest(request, response);
   }

   public void processRequest(HttpServletRequest request,
           HttpServletResponse response)
              throws IOException, ServletException {
      // Get requested action
      String actionName = request.getParameter(ACTION_KEY);
      // Set destination to default error page
      String destinationPage = ERROR_PAGE;

      // Find the requested action and perform
      if (actionName.equals("viewQuestionsFromCategory") {
         String categoryId = request.getParameter("categoryId");

         // Get questions for category with schemaId (pseudo method)
      List<Question> questionList = getQuestionsFromDatabase(categoryId);

         // Parse list of questions to the response object
         response.setAttribute("questionList", questionList);
         destinationPage = "/questionList.jsp";
      }
      else if (...) {...}
      else {}

      // Redirect to destinationPage
      RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher(destinationPage);
      dispatcher.forward(request, response);
   }
}
```

Listing 1.2: *ControllerServlet.java* - controller implemented as a servlet

Listing webxml shows the `web.xml` for the `ControllerServlet`. The servlet is mapped to */testcontroller/** and the wildcard character *** indicates that the servlet matches any url starting with */testcontroller* not matter what is added to the url. The controller servlet may now be requested from a browser using the url *http://localhost:8080/testcontroller?action=viewQuestionFromCategory*.

```
<web-app>
   <servlet>
      <!-- Making the servlet avaliable -->
      <servlet-name>TestControllerServlet</servlet-name>
      <servlet-class>ControllerServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <!-- Mapping the servlet to an URL -->
      <servlet-name>TestControllerServlet</servlet-name>
      <url-pattern>/testcontroller/*</url-pattern>
   </servlet-mapping>
</web-app>
```

Listing 1.3: The mapping of the controller in the web application.

1.1.2 Java Server Pages

Java Server Pages (JSP for short) is a markup and scripting language much like *php* and *asp*. It combines HTML, XML and Java to create dynamic content to a web application. Java and scripts might lead the thought towards JavaScript but there is a huge difference between JavaScript and JSP. JavaScript is sent to the client and then the client runs the script and produces the result. JSP runs on the server and only the result is sent to the client.

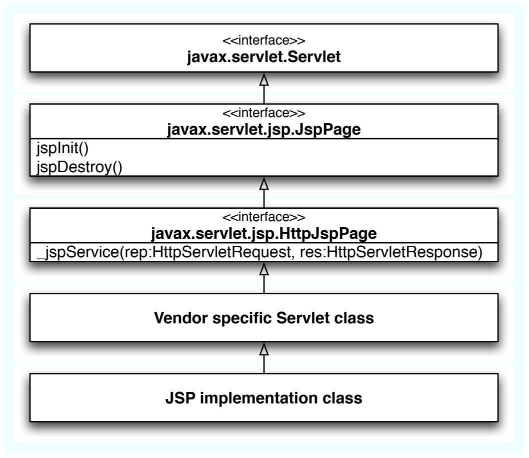
JSP is not the same as servlet either. The difference between servlets and JSP is often describes as servlets being Java code with embedded HTML whereas JSP are HTML with embedded Java code.

When JSP scripts are deployed to the container they are translated into servlet code by the JSP engine which during then translation checks the syntax and validity of the code. The resulting code is a servlet which extends a vendor specific class which in turn extends the `HttpJspPage` interface as in figure 1.5.

After the translation the resulting code is compiled into a servlet which behaves the same way as application servlets described earlier. The lifecycle of the JSP servlet is exactly the same though the methods have new name; `jspInit`, `jspDestroy` and `_jspService`. Architecturally JSP may be seen as an higher-level abstraction of servlets.

A JSP script may be broken down to eight elements.

- **Static code** like HTML.
- **Directives** which are command evaluated during the compilation of the JSP page.



<<interface>>
javax.servlet.Servlet
<<interface>>
javax.servlet.jsp.JspPage
jspInit()
jspDestroy()
<<interface>>
javax.servlet.jsp.HttpJspPage
_jspService(rep:HttpServletRequest, res:HttpServletResponse)
Vendor specific Servlet class
JSP implementation class

Figure 1.5: The class hierarchy of the resulting servlet of a JSP page.

- **Declarations** which can be any valid Java declaration. They are copied as is to the servlet code during compilation.
- **Expressions** which are standard Java expressions that evaluate into a value. They are evaluated once per request.
- **Scriptlets** which are Java code embedded into the page. At compilation they are copied as is into the `_jspService` method of the resulting servlet and are evaluated once per request.
- **Actions** which are commands given to the JSP engine. They are evaluated at request time and are stated in XML format (unlike the other elements).
- **Comments** does not affect the output in any way. They are considered part of the JSP code and does not affect the result or the servlet.
- **Custom tags** which are tags made by the developer or loaded from a library.

1.1.2.1 Scopes in JSP

JSP has four different scopes an application may exist in. All elements in a page exists in one of these scopes - but not necessarily the same. The object are maintained in different object as attribute-value pairs and may be requested through `getAttribute` and `setAttribute` methods of the object.

The four scopes are:

- **Application scope** where objects are shared across all components of the application. The objects are available for the entire application life time. They are maintained in the `ServletContext` instance as attribute-value pairs.
- **Session scope** where object are shared across all request in the same session. The objects are available throughout the session and removed when the session ends. They are maintained in the `HttpSession` object.
- **Request scope** where objects are shared across the elements which are part of the same request. The objects are available during the request and they are maintained in the `HttpServletRequest` object.
- **Page scope** where objects are available in the translation unit in which the are defined. They are maintained in the `PageContext` object.

The default scope if not set in the declaration is Page.

JavaBeans may be used inside the JSP page. The obvious advantage of using beans is a lot of saved work for the developer as JavaBeans provide a standard way of setting and getting properties. But a factor just as important is the use of scopes as the bean may be declared inside one of the above scopes making the bean and the information it holds available to subsequent requests.

```
<jsp:useBean id="question" class="Question" scope="session" />
```

The bean declared in a JSP page using an action like the above. A JavaBean of class Question is set in the session scope. The property of the bean may now be accessed using getters and setters e.g. `<% out.print(question.getText()) %>`.

```
<jsp:setProperty
        name="question"
        property="text"
        value="Do you know the safety procedure?"
/>
```

1.1.2.2 Examples of JSP

Listing 1.4 shows an example of a JSP script. The `c:forEach` loop is bound to a variable `questionList`. The variable is part of the request environment and could e.g. be set by a servlet like the `ControllerServlet` in listing 1.2. The list could also be generated inside the JSP script by putting some scriplet code

```
<!-- Use the jsp jstl library -->
<%@ taglib prefix="c" uri="http:/java.sun.com/jsp/jstl/core" %>

<html>
   <head>
      <title>QuestionList In JSP</title>
   </head>
   <body>
      <table>
         <!-- Run a foreach on all items in questionList -->
         <!-- Inside the loop the current question is bound to qst -->
         <c:forEach items='${questionList}' var="qst">
            <tr>
               <!-- Print the text of current qst -->
               <td>${qst.text}</td>
            </tr>
         </c:forEach>
      </table>
   </body>
</html>
```

Listing 1.4: A list of questions is printed out inside a table on the jsp page.

like listing 1.5 in the script. This is not good programming practice though and should in general be avoided.

The current element of the forEach loop is bound to a variable `qst` and the properties of the object may be referenced by putting the property after the variable separated by a punctuation.

```
<!-- Import the needed java classes to make the questionList -->
<%@ page import="java.util.ArrayList, dk.eapv.domain.Question" %>
<%
   // Prepare questionList
   ArrayList<Question> questionList = new ArrayList<Question>();

   // Create two questions objects and add them to the list
   Question qst1 = new Question();
   qst1.setText("Do you know the safety procedure?");
   questionList.add(qst1);
   Question qst2 = new Question();
   qst2.setText("Are accidents thoroughly investigated?");
   questionList.add(qst2);

   // Add the questionList to the context of the page
   pageContext.setAttribute("questionList", questionList);
%>
```

Listing 1.5: The list of questions may be created inside the jsp page.

1.1.2.3 Tag Libraries

JSP provides the possibility of implementing a lot of functionality into the page with Java code, but as mentioned earlier this is not good programming practice. To avoid this JEE offers the possibility of implementing custom tags and collect them in *Tag Libraries*. The logic is implemented in Java code and then made available through tags in the JSP page. This way the logic is separated out and may be reused in other pages.

The library consists of two thing - a number of Java files with the logic (normally packed into a standard *jar* file) and a Tag Library Descriptor (TLD for short). The TLD is a XML file and contains the meta information which maps the elements of the Java code to the tags used in JSP along with information on attributes.

To use a library it the JSP page must import it. This is done using a JSP directive.

```
<\%@ taglib uri="uriToTLD" prefix="myTL" \%>
```

The uri is the uri to the TLD of the library - not the *jar* file. It may either be root-relative, relative to current position or absolute. The prefix is a reference to the library used inside the script as `prefix:tagname` to call methods or set properties of the library.

```
<myTL:findQuestions selectedCategory="2" storeResultIn="result" />
<c:forEach items="result" var="qst" />
        ${qst.text}
</c:forEach>
```

JavaServer Pages Standard Tag Library or JSTL is a standard tag library of JEE. The library extends the JSP specification and adds standardized functionality for common tasks like loops, conditions and data processing. In listing 1.4 JSTL was used to provide the `forEach` tag - note how it was implemented at the beginning of the script.

In addition to the JSTL there are a lot of open source Tag Libraries available throughout the Internet. It will often be possible to find a library containing all or part of the functionality needed.

1.1.3 Java Server Faces

Though servlets, JSP, custom taglibs and javabeans are powerful technologies which together makes up a powerful platform for developing and running client/server applications a lot of issues are not addressed. Some examples of issues are conversion between java objects and http, validation of input to the application, internationalization and different languages on the website and navigation which is often mixed in between Java or JSP code.

Through the years a lot of different solutions to these issues have appeared but most of them have been either developer- or project specific solutions without the flexibility required for a real framework (most of them based on the Model 2 pattern[4]). This finally lead to the JSF Specification from SUN. The most popular JSF framework is MyFaces MyFacesdevelop as under the Apache project (section 2.1.5).

Conventional desktop application programming has for many years been dominated by the popular term *Rapid Application Development* (RAD for short). The methodology was developed by James Martin[5] and involves the construction of prototypes and the use of Computer-aided software engineering (CASE) tools. A lot of RAD tools have emerged through the years e.g. PowerBuilder, Delphi and Microsofts Visual Studio used for e.g. C#, Visual Basic and others.

JEE did not have a standard Java RAD tool though a few individual tools emerged. The Java Server Faces specification provides these standards and over the past few years several vendors[6] have implemented JSF into their respective RAD tools thereby making a framework to build web based UIs in Java.

Listing 1.6 shows the question list written through JSF. This shows the conversion between the java objects and the resulting http response. The conversion is mapped by the value attribute of the JSF tags. Note that the bindings are expressed in JSF Expression Language which is surrounded by `#{ }` tags (and not `${ }` tags like JSP Expression Language). The code is not much shorter or simpler than the same page and result written in JSP - but this very simple example does not show the real powers of JSF. Note the suffix of the filename remains *.jps* as the JSP engine is still used to translate the page. In the browser though the page is referenced by *questionList.jsf* and *questionList.jsp* will not

[4]Model 2 is a variation of the MVC architectural pattern specific for web applications. In the pattern the browser is considered part of the model

[5]Dr. James Martin has been called *guru of the information age*. He has written over a hundred books on the subject and has been one of the early promoters of fourth generation programming languages like Maple, ColdFusion and Ruby on Rails.

[6]IBMs WebSphere Application Developer, Oracles JDeveloper and Suns Java Studio Creator are the three best known examples of RAD with JSF support

```
<!-- Use the jsf libraries -->
<%@ taglib prefix="f" uri="http:/java.sun.com/jsf/core" %>
<%@ taglib prefix="h" uri="http:/java.sun.com/jsf/html" %>

<html>
   <f:view>
      <head>
         <title>QuestionList In JSF</title>
      </head>
      <body>
         <!-- Use the dataTable tag to print the table with questions -->
         <h:dataTable value="#{questionList}" var="qst">
            <h:column>
               <!-- Print the text of the current question qst -->
               <h:outputText value="#{qst.text}" />
            </h:column>
         </h:dataTable>
      </body>
   </f:view>
</html>
```

Listing 1.6: *questionList.jsp* - The list of questions may be created inside the jsf page.

work.

In the two first lines of the code - two tag libraries are loaded. These are the *JSF HTML Custom Actions* and the *JSF Core Custom Actions* libraries. These libraries contains standard tags for most common visual items used on web pages like the table rendered using the `h:dataTable` tag.

1.1.3.1 Validation in JSF

Validation of input is essential to any application and so is the case for web applications. Neither servlets nor JSP provides a simple standardized way to validate the input sent from forms. JavaScript may be used inside the JSP page to check the different input fields or a validation method may be implemented in either the servlet or the bean holding data. The development of a validation method is often cumbersome and time consuming - and it actually more or less the same tasks over and over again but on different fields.

A large part of the JSF specification considers the creation of forms and the validation of the input of them. This makes developing pages with input forms much easier as the tag library handles everything from rendering fields, validating the input, binding values to beans and displaying error messages.

Listing 1.7 shows a simple bean used to hold users of a system. It has three properties; username, password and a boolean value indicating whether the user is a superuser. The bean has a method for saving the method in the database. Getter and setter methods are left out.

```
public class UserBean {
   private String username;
   private String password;
   private Boolean superUser;

   public String createUser() {
      // Since the username, password and superUser properties
      // is bound to the form in the jsf page the only thing
      // to do here is save the values to e.g. a database.
      try {
         this.save();
         return "success";
      }
      catch (Exception e) {
         return "failure";
      }
   }


   // Getters and setters for properties
   ...
}
```

Listing 1.7: *UserBean.java* - a simpel bean for a user.

To create a new user a simple form must be filled. The page with the form is generated from listing 1.8. The page generates a form `<h:form id="createUserForm">` containing three inputfields; `h:inputText`, `h:inputSecret`, `h:selectBooleanCheckBox` for the username, password and a checkbox to indicate superuser.

The input fields is all bound to a property of the UserBean by their attributes. If the validation succeeds these values are automatically set on the bean and available for the `createUser` method in the bean which is invoked by the submit button rendered from the `commandButton` tag. The method is bound through the tags action attribute.

When the form is submitted the fields are validated according to the validation attributes set on them. All three fields has a `required="true"` attribute indicating a *non null* field. Additionally the `username` and the `password` field tags has a `f:validateLength` tag setting the minimum and maximum number of characters of the field.

Beneath each of the fields is a `h:message` tag. Each tag is bound to the previous field through its `for=""` attribute. If the validation of a field fails, this tag will render an error message.

As simple as it may seem this is everything needed to create a form, validate the

```
<!-- Use the jsf libraries -->
<%@ taglib prefix="f" uri="http:/java.sun.com/jsf/core" %>
<%@ taglib prefix="h" uri="http:/java.sun.com/jsf/html" %>

<html>
   <f:view>
      <head>
         <title><h:outputText value="#{message.createUser}" /></title>
      </head>
      <body>
         <h:form id="createUserForm">
            <h:outputText value="#{message.username}/><br/>
            <!-- Input boks for username - bound to username property
                  of userBean of class UserBean. The box must be filled
                  and minimum length is 2 - maximum length is 20 -->
            <h:inputText id="username" value="#{userBean.username}
                                                     required="true">
               <f:validateLength minimum="2" maximum="20"/>
            </h:inputText>
            <!-- Print errors if validation fails -->
            <h:message id="usernameError" for="username" />
            <br/><br/>

            <h:outputText value=""#{message.password}/>"Password<br/>
            <h:inputSecret id="password" value="#{userBean.password}
                                                     required="true">
               <f:validateLength minimum="4" maximum="20"/>
            </h:inputSecret>
            <h:message id="passwordError for="password" />
            <br/><br/>

            <h:outputText value="#{messagge.superuser} />
            <h:selectBooleanCheckBox id="superuser"
               value="userBean.superUser" required="true"/>

            <!-- Submit Button for form -->
            <h:commandButton id="submit" action="#{userBean.createUser}"
               value="#{message.submitCreateUser}" />
         </h:form>
      </body>
   </f:view>
</html>
```

Listing 1.8: *createUser.jsp* - a JSF page for creating a user. The input elements of the form is bound to a UserBean.

input when the form is submitted and showing error messages on a web page.

1.1.3.2 Internationalization in JSF

One of the other great features of JSF is the language handling. All messages on a screen may be put into a property file and then referenced from the JSF page via the `message` variable.

The property file is a simple text file containing a set of key-value pairs. One or more property files may be bundled together into a language bundle which may be loaded into the JSF page or setup in the *faces-config.xml*. Listing 1.9 shows the english language file for the *createUser.jsf*.

```
createUser=Create a user
username=Username (min. 4 characters)
password=Password
superuser=Super user with all privileges
submitCreateUser=Save User
```

Listing 1.9: *message.properties* - a language file is a flat text file with key=value pairs.

1.1.3.3 Navigation in JSF

The navigation on websites and in web applications has always been autonomous and decentralized to the individual scripts and pages. The consequence is often a loss of overview of the application as this turns out to be rather complex in large applications.

JSF centralizes the navigation by a navigation configuration. In the configuration every page is identified and the possible ways to navigate on from the page is configured. The navigation configuration is saved in *faces-config.xml* and listing 1.10 shows the configuration for the *createUser.jsp* in listing 1.8.

```
<navigation-rule>
   <from-view-id>/createUser.jsp</from-view-id>
      <navigation-case>
         <from-outcome>success</from-outcome>
         <to-view-id>/userList.jsp</to-view-id>
         <redirect/>
      </navigation-case>

      <navigation-case>
         <from-outcome>failure</from-outcome>
         <to-view-id>/failure.jsp</to-view-id>
         <redirect/>
      </navigation-case>
</navigation-rule>
```

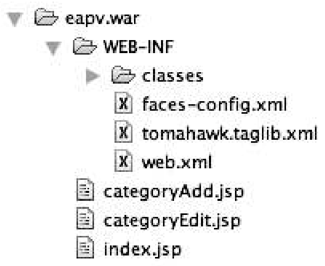
Listing 1.10: *faces-config.xml* contains the navigation configuration for a JSF application.

The navigation configuration has two navigation cases based on the outcome of the `createUser` method in the UserBean in listing 1.7. If the creation succeeds the method returns a string `success` and if the creation fails `failure`. Based on this answer the JSF framework selects a navigation case and hand over control to the specified view. Note that view-ids in the configuration always are set relative to the web application root starting with the `/`.

1.1.4 Packing and deployment

The web application is packed into a *jar* file with the suffix *.war* (Web ARchive). The packed file must have the structure as shown in 1.6.

The web root is the root of the file and the uri used in the browser refers to this root. The archive must have a *WEB-INF* directory in root. The directory should contain all deployment descriptors like the *web.xml*, tag library descriptors and the *faces-config.xml* file is using JSF. The directory should also contain a directory named *classes* which contains the compiled class files from e.g. servlets.



```
▼ eapv.war
    ▼ WEB-INF
        ▶ classes
          faces-config.xml
          tomahawk.taglib.xml
          web.xml
      categoryAdd.jsp
      categoryEdit.jsp
      index.jsp
```

Figure 1.6: The structure of the web application package file

1.2 Enterprise Java Beans

There is no doubt that the biggest difference from JEE 5 to the previous versions is on the Enterprise Java Beans (EJB for short). With JEE a new specification of the EJB has been developed - the EJB3.0 specification.

Enterprise Java Beans forms the model in a Model 2 setup. This is where the business logic is implemented and preform operations on the data of the application.

The biggest problem with the former specifications was the tight connection to the EJB container. Every bean was deployed via a deployment descriptor describing the bean, the methods and the parameters of the bean. This made deployment very cumbersome and error prone causing endless pain to developers. [5] which uses the previous J2EE and EJB2.1 specifications uses several pages on arguing against using them in simple applications.

The EJB3.0 specification is a complete overhaul of EJBs and the EJB container simplifying things extremely. The deployment descriptor is no longer needed (though the meta-data given as annotations may also be set via a deployment descriptor) and the beans have been simplified to simple Java beans making applications simple and elegant.

There are three different kinds of EJBs:

- **session beans** which contains the core business logic.
- **entity beans** which represents the data of the application.
- **message driven beans** which processes asynchronous messages (these will not be described further).

1.2.1 Entity Beans

Previous versions of JEE entity beans were very complicated to use compared to the benefits. The implementation had to follow specific standard and the beans were deployed through deployment descriptors. This made the use of entity beans unnecessarily complicated to use, deploy and test. The consequence were that JEE entity beans were not used. Instead several frameworks emerged fulfilling the persistence task. The best known is probably the Hibernate framework developed by Gavin King but all of them build on the idea of using simple

Java beans (also called Plain Old Java Objects or POJOs for short) in the application and the map these simple objects to databases through some persistence mechanism.

When the Java Community started working on the EJB3.0 specification they invited Gavin King into the specification process. The process ended up with a total reinvention of the entity beans and their role as the persistence were separated into its own specification - the *Java Persistence 1.0*. The specification provides an abstraction layer of persistence which - opposite earlier versions - make the entity beans independent of database and container. This removed much of the complexity of using entity beans and in the new specification entity beans are simply POJOs annotated with `@Entity` on the class and persistence specific annotations on class and methods (see more on annontation in appendix A).

1.2.1.1 The EntityManager

The persistence functionality in Java Persistence 1.0 is handled by the `javax.persistence.EntityManager` interface. Before using the entity manager a *persistence unit* is needed. The entity manager maps a fixed set of classes to a particular database - and this set is the persistence unit. It is configured in *.xml* file named *persistence.xml* and listing 1.11 shows an example.

```
<persistence>
   <persistence-unit name="eapv">
      <jta-data-source>java:/eapvDS</jta-data-source>
      <properties>
         <property name="hibernate.hbm2ddl.auto" value="create-drop" />
         <property name="hibernate.dialect"
                 value="org.hibernate.dialect.MySQLInnoDBDialect"/>
         <property name="jboss.entity.manager.factory.jndi.name"
                 value="java:/EntityManagerFactories/entityManager"/>
      </properties>
   </persistence-unit>
</persistence>
```

Listing 1.11: *persistence.xml* - the configuration of a persistence unit.

In the configuration the name and data source of the persistence unit is set. The data source is configured in yet another *.xml* file which is deployed to the ejb container. This file basically holds information of the uri to the database server and username, password and the name of the database. The persistence unit also defines which hibernate dialect is used based on the database server used for the application and the name of the entity manager used in the project.

As the persistence unit is configured entities may now be managed by the entity manager. An entity is either managed or unmanaged. When an entity is attached to the entity manager it becomes managed and the entity manager keeps track state changes and synchronizes changes with the database. An entity may be detached and become unmanaged and will be when the when the persistence context is closed. A persistence context is a set of managed object instances. There are two different persistence contexts; transactional which lives throughout a transaction and extended which lives longer than a transaction. Transaction-scoped persistence context is the default but extended persistence context may be managed by the application code.

A persistence context may be created by calling `EntityManagerFactory.createEntityManager()` which returns an `EntityManager` representing the context. Using the factory is a little cumbersome and awkward though, so in general it is recommended to *inject* [7] the entity manager directly into the session bean.

```
@PersistenceContext (unitName="eapv")
private EntityManager entityManager;
```

Note the unit name is the name set in the *persistence.xml* file. Objects may then be persisted through the `persist(Object object)` method.

```
entityManager.persist(question);
```

The manager has methods for common persistence tasks like merging and removal of objects but also for using *query language* on the database as described in section 1.2.1.4.

1.2.1.2 Object to relation mapping

When persisting objects to a database the object must be mapped to a database table and the properties of the object to the columns of the table. In the previous versions of the EJB specification this was done in the deployment descriptor of the entity bean thereby putting information about the bean and information about the mapping of the bean to the table two separate places. The previously mentioned frameworks changed this and specifically Hibernate put the information inside the bean class. The information were put into *annotations* inside comments in the code. This practice have been adopted by the EJB3.0 specification but annotations have become part of the language in Java 5.0 making the bean code yet more consistent. The mapping could be done using

[7]Dependency Injection is an architectural pattern where one object uses a second object to provide some particular capacity. It is also called Inversion of Control and is a way to achieve louse coupling between different elements.

XML inside a `<entity-mappings>` element in the *persistence.xml* file. It might be a matter of taste which to use - the xml version of the mappings will not be shown here.

Mapping a class to a table is done using the `javax.persistence.Table` annotation.

```
@Table (name="Schema")
public Class Schema {
...
```

The `@Table` annotation is not required as the entity manager will use the name of the class if the table name is not specified.

The properties of the bean is mapped to the columns of the table. This is done by annotating either the getter or the setter method of the property.

```
private String name;

@Column (name="SchemaName")
public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}
```

As for the class if a property is not annotated with a `@Column` annotation the name of the property is used for the column.

Some properties might have special annotations either because of the way databases work or because a special mapping between the database and the beans is needed. An entity bean must e.g. have a *primary key* which is the identity of the bean. The primary key can map one or more properties and must be a primitive type, a String or a primary-key class composed of primitive types and strings. The primary key is annotated with the `@ID` annotation. The primary key could either be generated by the database or set in the application. If it is generated by the database a `@GeneratedValue` annotation is used in conjunction with the `@Id` annotation.

```
private int id;

@Id
@GeneratedValue
@Column (name="SchemaId")
public int getId() {
        return id;
}

public void setId(int id) {
        this.id = id;
}
```

The persistence mechanism will in most case automatically map the properties of a bean to the correct types in the database where the types are primitive types or primitive wrapper types. This is known as basic mapping and may be annotated with the `@Basic` annotation though it is rarely used as this is the default mapping strategy. It may however be useful if a fetch strategy is needed. A property may either be loaded as `LAZY` or `EAGER`. With an `EAGER` strategy the property is fetched from the database when the entity bean is fetched whereas the `LAZY` strategy fetches the property first time the it is accessed.

The `@Temporal` annotation provides information to the persistence mechanism about the mapping of `java.util.Date` or `java.util.Calendar` properties. Using this annotation allows the bean developer to map these types to either a `TIME`, `DATE` or `TIMESTAMP` field in the database.

```
private Date created;

@Column (name="SchemaCreated")
@Temporal(TemporalType.TIMESTAMP
public Date getCreated() {
        return created;
}

public void setCreated(Date created) {
        this.created = created;
}
```

An entity bean might contain a property that should be persisted to the database and therefore not mapped to the table. The `@Transient` annotation tells the persistence mechanism not to map the property.

These are the general tools for mapping entity beans to database tables. See [1] for information on more special cases of mapping.

1.2.1.3 Entity Relationships

In the previous section entity bean properties of simple types were mapped to a database table through annotations. This works fine for simple concepts but entity beans may also form more complex relationships to other entity beans modeling in order to model more complex real-world concepts. Persisting these relationships is a crucial part of the persistence mechanism.

There are four types of relationship cardinalities. These can be *one-to-on*, *one-to-many*, *many-to-one* or *many-to-many*. Additionally each of these types may be either *unidirectional* or *bidirectional*. If a relationship is unidirectional only one entity knows of the relation to the other entity whereas both entities knows of the relationship if it is bidirectional. This yields seven different relationships.

- One-to-one unidirectional
- One-to-one bidirectional
- One-to-many unidirectional
- One-to-many bidirectional
- Many-to-one unidirectional
- Many-to-many unidirectional
- Many-to-many bidirectional

At first sight one might have guessed that four relationships cardinalities and relationship types would yield eight relationship types - but the many-to-one bidirectional and one-to-many bidirectional is the same relationship.

Fortunately these relationships are similar to the relationships derived from the *relational model* [8] and so makes the persistence easy.

As for the simple types of properties in the entity beans the related types are annotated (or they may be configured in *persistence.xml*).

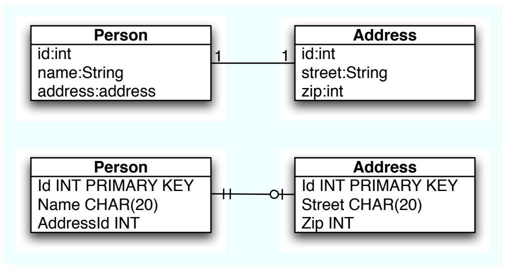


Figure 1.7: Unidirectional relation between two entity beans and the equivalent database relation.

To create a unidirectional relationship the relationship is configured in the entity bean which has a property of the other bean - see 1.7. The property is not annotated with a `@Column` annotation for the property but with a `@JoinColumn` annotation defining the column in the database table holding the foreign key of the other entity.

[8]The Relational Model was proposed by Edgar Codd in 1969 while working at IBM. The fundament for the model is relational algebra where all data is represented as n-ary relations (see [7] for more information on relational model and relational algebra).

```
public class Person {
        private Address address;

        @OneToOne
        @JoinColumn(name="AddressId");
        public Address getAddress() {
                return address;
        }
        public void setAddress(Address address) {
                this.address = address;
        }
}
```

The `Address` bean does not contain any annotations or fields regarding the person or the relationship. A `Person` object contains the correct address but an `Address` object has no knowledge of the `Person` object.

In bidirectional relationships both entities are aware of the relation. Say in the above example it was necessary to find a person from his address then a bidirectional relation is required. This only requires a change in the `Address` bean.

```
public class Address {
        private Person person

        @OneToOne(mappedBy="AddressId")
        public Person getPerson() {
                return person;
        }
        public void setPerson(Person person) {
                this.person = person
        }
}
```

The bean now has a Person property which is mapped by a `OneToOne` relation to the `Person` bean. The `mappedBy` attribute indicates which column in the Person table maps the `Address` entity to the `Person` entity.

`OneToMany` and `ManyToOne` relations are mapped the same way using the `@JoinColumn` annotation. If the relation should be bidirectional the `mappedBy` attribute is used. In both cases the persistence mechanism will add a column to the "One" side of the relation and keep the primary key of the other in this column.

The `ManyToMany` relation is a bit different. In the relational model `ManyToMany` relations cannot be modeled. The solution is to split the relation into two new relations - a `ManyToOne` and a `OneToMany` relation and add an extra table joining the two relations - see figure 1.8

A unidirectional relation is annotated on the entity bean with knowledge of the relation and the other bean.

```
public class Person {
```

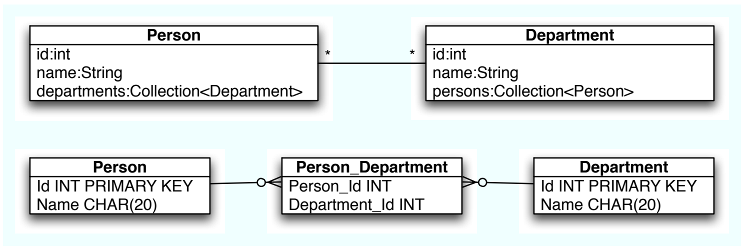


Figure 1.8: ManyToMany relation between two entity beans and the equivalent database relation.

```
        private Set<Department> departments = new HashSet<Department>();

        @ManyToMany
        @JoinTable(name="Person_Department",
                joinColumns{@JoinColumn(name="PersonId")},
                inverseJoinColumns={@JoinColumn(name="DepartmentId")})
        public Set<Department> getDepartments() {
                return departments;
        )
        public void setDepartments(Set<Department> departments) {
                this.departments = departments;
        }
```

The relation may be made bidirectional using the `mappedBy` attribute in conjunction with the `ManyToMany` annotation on the `Department` bean.

```
public class Department {
        private Set<Person> persons = new HashSet<Person>();

        @ManyToMany (mappedBy="departments");
        public Set<Person> getPersons() {
                return persons;
        )
        public void setPersons(Set<Person> persons) {
                this.persons = persons;
        }
```

The relation now resembles the relation shown in figure 1.8. Note that the `mappedBy` attribute is mapped to the `departments` property of `Person` as this property contains the annotations with the mapping information.

The four relation annotations all have an attribute named `cascade()`. The attribute is used to cascade operations performed by the entity manager on the related entities - that is the same operation performed on an entity is performed on the related entities.

```
public class Category {
        private Set<Question>  questions = new HashSet<Question>();

        @OneToMany (cascade={CascadeType.PERSIST});
        public Set<Question> getQuestions() {
                return questions;
        )
        public void setQuestions(Set<Question> questions) {
                this.questions = questions;
        }
```

There are five different cascading types; PERSIST, MERGE, REMOVE, REFRESH and ALL which indicates all four operations.

1.2.1.4 EJB QL

Querying is a fundamental feature of relational databases as this allows the developer to find, calculate and report the information in an almost endless number of ways. This flexibility has made relational databases so popular that most databases today are relational (though a few other types like object oriented databases exists). Queries are formatted in *Structured Query Language*Ê[9] or *SQL* for short.

In Java Persistence queries are performed using both SQL and *EJB QL* which is a language similar to SQL but is constructed to perform queries on Java objects rather than on databases like SQL.

```
FROM Schema scm ORDER BY scm.id DESC
```

The above example shows a typical simple EJB QL query. It select all schemas in the database ordering them by the id of the schema in descending order. Note how query is expressed in terms of the persistence schema. The variable `scm` is bound to the `Schema` object and the properties of the object is referenced with `scm.id`.

The entity manager has a method for creating queries on the database. The method takes a EJB QL string as parameter.

```
Query que = entityManager.createQuery("FROM Schema scm ORDER BY scm.id DESC");
List result = que.getResultList();
```

When the query is executed, the entity manager uses the mapping information described in the previous sections to translate the EJB QL into SQL which conforms with the underlying database and the vendor specific notations. The SQL is sent through JDBC and a vendor specific driver. This way EJB QL is

[9]SQL was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s. It was formally standardized by ANSI in 1986 and subsequent versions have been released as ISO standards.

portable between containers and databases since the entity manager handles the translation.

There are a few specialties that cannot be set through the EJB QL but must be set with parameters. Parameters are set on the `Query` object using the `setParameter()` method and refers to an "anchor" in the query string. Below is an example with a label but it is possible to refer to the number of the anchor.

```
Query que = entityManager.
        createQuery("FROM Category cat WHERE cat.schema=:first");
que.setParameter("first", 3);
List result = que.getResultList();
```

This way of setting parameters are especially needed when setting dates on an object as the `TemporalType` must be set.

```
que.setParameter(String name, Date value, TemporalType temporaltype);

que.setParameter(String name, Calendar value, TemporalType temporaltype);
```

The EJB QL language has all the basic SQL constructions including joins (like `INNER JOIN` and `LEFT JOIN`), functional expressions (like `COUNT`, `DISTINCT` and `MAX`), WHERE clauses with a number of possibilities (like `WHERE ... BETWEEN`, `WHERE ... IN`, `WHERE ... IS NULL`, `WHERE ... MEMBER OF` and `WHERE ... LIKE`), ORDER BY clauses and GROUP clauses. Additionally subqueries are possible so almost anything possible in SQL is possible.

Though EJB QL meets most developer needs some vendor specific queries cannot be performed on a database with EJB QL. During the specification of EJB QL a group of developers foresaw this limitation and provided an API for this.

1.2.2 Session Beans

Session beans model the logic of a domain (opposite entity beans that model the domain data). The logic might be seen as the business processes or workflows of the domain. These processes are implemented as methods in a session bean and often related functionality is gathered in the same bean e.g. administration of users where the bean has method for creating, editing, deleting and listing users. The session bean may use entity beans to represent the data and manipulate the entity beans.

A session bean consists of two elements; a bean interface and a bean class. Clients interact with the bean through the interface and the logic is implemented inside the bean class. The bean interface may either be a *Local interface* or a

Remote interface. The local interface may only be called from inside the ejb package whereas the remote interface may be called from outside - even from other servers through *Java RMI* [10].

There are two types of session beans; *stateless* beans and *stateful* beans.

1.2.2.1 Stateless Session Beans

Stateless session beans are best described as a collection of related methods implemented in a bean. The bean maintains no state across requests. When a client invokes a method on a stateless session bean, an instance is requested from a pool of bean (the Method-Ready pool), the method is executed on the bean and the result is sent to the client. Then the bean is returned to the pool ready to service other requests. It has no knowledge of other beans or of requests before or after the execution.

```
@Stateless

public class UserBean implements User {
    public void createUser() {
        ...
    }

    public void editUser() {
        ...
    }

    public void deleteUser() {
        ...
    }
}
```

Listing 1.12: *UserBean.java* - an example of a stateless session bean.

Listing 1.12 shows an example of a stateless session bean. A stateless session bean is annotated with the `@Stateless` annotation on the class.

1.2.2.2 Stateful Session Beans

Stateful session beans are best described as an extension of the client. When a client invokes a method on the bean, the bean performs the some functionality on behalf of the client and maintains state related to the client. The big advantage

[10]Java Remote Method Invocation (Java RMI for short) is an API which provides remote communication that allows an object running inside one Java Virtual Machine to invoke methods on an object running inside another Java Virtual Machine.

here is that the session bean holds the data instead of sending it to the client and on the next call the client must send it to the session bean an so forth.

An instance of a stateful session bean is dedicated to the same client for its entire lifetime [11] and maintains what is called *conversational state* . The concept may be compared to a conversation between two people having a conversation on a subject. When one responds to a "call" he still knows what the prior "calls" and responses in the conversation were. Stateful beans should not be confused with entity beans though as they are not persistent and though they may access the database the do not model the data in the database.

```
@Stateful

public class SchemaEditorBean implements SchemaBean {
   private Schema schema;

   @PersistenceContext (unitName="eapv")
   private EntityManager entityManager

   public void createSchema() {
      schema.setName("VarFromForm");
   }

   public void addCategory() {
      Category cat = new Category();
      cat.setName("VarFromForm");
      schema.addCategory(cat);
   }

   public void saveSchema() {
      entityManager.persist(schema);
   }

   @Remove
   public void destroy() {}
}
```

Listing 1.13: *SchemaEditorBean.java* - an example of a stateful session bean.

Listing 1.13 shows an example of a stateful session bean. The bean is used to create a schema. First the name of the schema is set, then a category is created and added to the schema and finally the schema is saved. Note that the instance variable `schema` is used in all three methods over three calls to the bean as its state is maintained. Also note the `destroy()` method annotated with a `@Remove` annotation. A stateful session bean must have this notation on a method which is called just before the instances is removed. This may be used for cleaning up before removal.

Stateful session beans may be nested through injection. When an instance of

[11] [1] states that this is a conceptual model. Some vendors may share instances through instance swapping but makes it appear as if the same instance serves the client on all requests

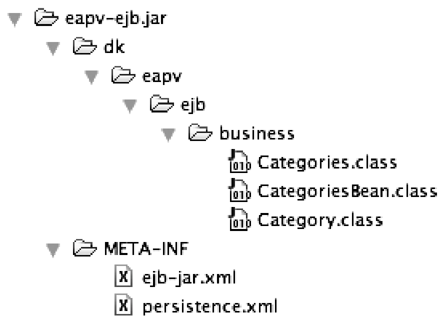
the containing bean is instantiated a unique session is created for the referenced session bean - and when removed the referenced instance is removed too.

Earlier there was a common opinion in the Java community that stateful session beans are a scalability killer as maintaining the state over several request require a vast amount of resources. This probably were the reality in the middle of the 1990s but since then EJB containers has evolved a lot, and today they have extremely sophisticated mechanisms for stateful session bean state replication e.g. through a fine-grained replication where only the attribute values which actually changed are replicated.

1.2.3 Packing and deployment

Enterprise Java Beans is packed into a *jar* file with the suffix *.jar* (Java Archive) as it is considered a java application. The file must have the structures shown in figure 1.9.

The root of the jar file must contain a *META-INF* directory with the persistence configuration in *persistence.xml*. The directory may also contain an *ejb-jar.xml* deployment descriptor file. This is an xml file with the configuration of the EJBs and is only necessary if the configuration is not done using annotations.



```
eapv-ejb.jar
  dk
    eapv
      ejb
        business
          Categories.class
          CategoriesBean.class
          Category.class
  META-INF
    ejb-jar.xml
    persistence.xml
```

Figure 1.9: The structure of the EJB jar package file

1.2.3.1 Packing the entire application

As said earlier the web application *.war* file may be deployed as is if it contains the entire application without EJBs. If EJBs are used in the application the application must be wrapped up into a *jar* file with the suffix *.ear* (Enterprise ARchive). The structure of the package is shown in figure 1.10.

The previously packed files goes into the root along with other files if necessary (e.g. the JBoss Seam application and some JSF packages used along with JBoss Seam).

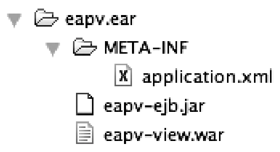


Figure 1.10: The structure of the enterprise application package file

The root folder contains a *META-INF* directory holding a deployment descriptor - the *application.xml*. This file should contain a reference to all files packed into the enterprise package as listing 1.14 shows.

```
<application version="1.4">
   <module>
      <web>
         <web-uri>eapv-view.war</web-uri>
         <context-root>/eapv</context-root>
      </web>
   </module>
   <module>
      <ejb>eapv-ejb.jar</ejb>
   </module>
</application>
```

Listing 1.14: *application.xml* - the deployment descriptor for the enterprise archive.

Chapter 2

Jboss Application Server

As mentioned in the introduction to the chapter on JEE the JEE platform does not provide the middleware between JEE and the custom application. JBoss Application Server is the most widely used web server middle ware for JEE.

The JBoss project was started by Marc Fleury[1] in 1999 founding the JBoss group which in 2004 became JBoss Inc. In 2006 RedHat Corporation bought JBoss Inc. and it is now a division of RedHat.

JBoss has a lot of running project mainly developed by the JBoss community. The JBoss Application Server is the flagship and most of the other project support this - including JBoss Seam framework introduced later and the JBoss Eclipse IDE which is a plugin for the Eclipse Development IDE. The application server includes the EJB3.0 specification in its newest version.

[1]Marc Flery is originally from France. He worked at Sun Microsystems in France befor moving to the United States. He has worked on several Java projects.

2.1 JBoss Seam

Both Enterprise Java Beans and JSF is powerful technologies but unfortunately there is no standardized coupling between them. Both specification have excellent stubs for communication with other technologies but the implementation of the connection is left to the application developers.

JBoss Seam is a framework for coupling the two specification - "sewing" them together with a "seam" so to speak. The project started a few years ago in the JBoss community and one of the active developers is actually Gavin King. JBoss Seam is not an official specification but a specific framework.

2.1.1 How does it work?

The framework is a combination of taglibs and bean packages which sort of encapsulates EJB and JSF. Through reflection (which were implemented in Java 5.0) the framework "inspects" the beans and makes them available to JSF - and vice versa. JBoss Seams uses interceptors for this as the classes have no parent class of interface.

An EJB becomes a JBoss Seam element through a name. The class is simply annotated with a name and then the framework creates a reference for it which may be used from other beans or JSF pages. JSF pages are automatically included in the framework.

```
@Name ("Schema")
public class Schema {
...
}
```

2.1.2 Inversion of control and bijection

The sharing of information between JBoss Seam elements are done through inversion of control - and actually double inversion of control or *bijection*. In the EJB specification objects may be injected into a class but with JBoss Seam objects may also be outjected from a class and available to either JSF or another EJB. Injected objects are annotated with `@In` and outjected objects are annotated with `@Out` (actually they may also be outjected through a special construct named `@Datamodel` which is often used for JSF datatables).

2.1.3 Conversational context

In section 1.1.2.1 JSP scopes were discussed. JBoss Seam have extended the number of scopes with a few where the most important is the *conversational context* (actually the business process is very interesting too but is to be described here).

The conversational context may be compared to - a conversation. The conversation begins, some information is shared and operated on - and at some point the conversation ends. An example of using the conversational context is booking a hotel room. First you search for available rooms, then you choose a hotel, then you reserve the room, then you pay and finally you receive a confirmation of the booking. The smart thing about JBoss Seams conversational context is you may actually have several ongoing conversations at the same time without interfering with each other. So after you search for hotels you may actually start more than one booking conversation. The different conversations are even aware of each other and can share information if useful.

2.1.4 Some advanced features

2.1.4.1 Validation in the Model layer

In section 1.1.3.1 an example of how validation of input could be done in JSF. This way of validating input is simple but lacks one important property - the validation rules are configured in the View layer of the application and this may lead to a lot of problems (and is the wrong way of dividing responsibility). Say the user should be able to change username and password in the application. Since the user cannot himself change his status to superuser a new page is needed - *editUser.jsp*. The page will be exactly the same as the *createUser.jsp* in listing 1.8 except for the `h:selectBooleanCheckBox`. This way *createUser.jsp* has a set of validation rules and *editUser.jsp* has a set of validation rules making the validation rules redundant.

The solution to this problem is obvious - the validation rules must be configured in the Model layer of the application as the model (entity bean) of the user is located here. Unfortunate the EJB3.0 does not contain any validation specification but the Hibernate framework does. Luckily Hibernate is part of JBoss and JBoss Seam has some tools to utilize the validation.

The validation rules are setup on entity beans much the same way as the object-relational mapping was done in section 1.2.1.2 - through annotations. Just as with the mappings the annotations were embedded into comments in previous versions of Java but are now an independent part of the language. The annotations are placed above the *getter* method just as the mapping of the property to a column in the database table as in listing 2.1.

```
@Entity
@Name ("user")
@Table (name="User")

public class User {
   private String username;

   @Column (name="UserUsername")
   @NotNull (message="Username is required")
   @Length (min=2, max=20, message="Username must be 2-20 characters")
   public String getUsername() {
      return username;
   }

   public void setUsername(String username) {
      this.username = username;
   }
}
```

Listing 2.1: Validation annotations is put above the getter method of the property getter method

Each annotation may contain a `message` parameter which will be send to the View layer if validation of the annotations fails. Unfortunately these messages does not support different languages as with JSF validation. Also note the due to the architecture of the current JSF framework the `@NotNull` annotation do not work without the `required` attribute mentioned in section 1.1.3.1.

2.1.4.2 Navigation with JBoss Seam

JBoss Seam has two navigation models. You may either use JSF or Seam navigation rules which is the simple stateless navigation model or the more advanved jPDL and stateful navigation model.

The stateless model was introduced in section 1.1.3.3 as it is implemented as part of the JSF specification. Seam has its own set of rules that may be used instead of the JSF navigation rules - the difference between the two rulesets are small and is probably a question of preference of xml format. The stateless navigation model is often enough for small simple web applications.

The stateful model defines a set of transitions between of named, logical application states. The flow of the interaction with the user may be expressed using the jPDL pageflow definition. This separates the actionlistener methods

```
<page view-id="/createUser.jsp">
        <navigation>
                <rule if-outcome="success">
                        <redirect view-id="/userList.jsp" />
                </rule>

                <rule if-outcome="failure">
                        <redirect view-id="/failure.jsp" />
                </rule>
        </navigation>
</page>
```

Listing 2.2: *pages.xml* contains the navigation configuration for a Seam application.

form the navigation as the methods of the session beans does not return a string indicating the outcome of the method.

The biggest difference on the two models are the behavior of the browsers Back button. As the stateless model is stateless the user may freely navigate around with back, forward and refresh buttons. The responsibility of ensuring the state of the web application lies on the application. The stateful model on the other hand lets Seam handle the responsibility. The state is simply ensured by "disabling" the back button. When it is activated the current page is rendered over again possibly with an error message. Which model to use depends on the application and the developer. The disabling of the back button may be frustrating to the user whereas it may be a feature to the developer as state is maintained.

2.1.5 MyFaces and TomaHawk

The JBoss Seam package includes *MyFaces* which is a implementation of the JSF specification. MyFaces is a project under the Apache project.

The project provides a few taglibs in addition to the JSF implementation among others the *Tomahawk* library which provides standard implementation for tree-menus, calendar and a lot of other useful web layout tools. Additionally it provides some improvements on standard JSF tags e.g. the possibility of making a dynamic number of columns in tables which is not possible from the JSP specification. Figure 2.1 gives an overview of MyFaces used with JBoss Seam (note the *pages.xml* is used). With MyFaces comes a servlet which can be used as the controller in a MVC architecture. The JSF pages is then implemented as *ui:construction* which essentially is just JSF and HTML packed into a surrounding tag block. The servlet uses these block to construct the response sent to the client.

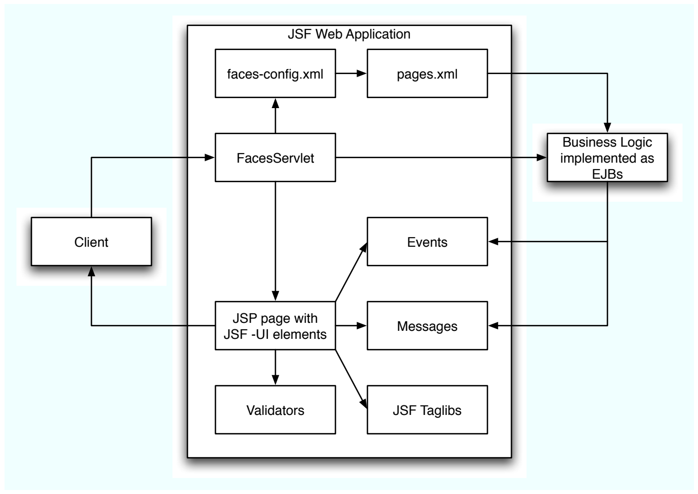


Figure 2.1: An overview of the JSF application and how it interacts with the different elements of JSF.

Part II

Xtreme Programming as Methodology

Chapter 3

Extreme Programming

Extreme Programming (XP for short) is a methodology for engineering software. It was developed by Kent Beck while working on the *Chrysler Comprehensive Compensation System* [1] which throughout the years adapted and and refined the methodology. Kent Beck published a book on the methodology in 1999 and though the project were closed by Chrysler Corporation in 2000 XP has become accepted in the developer community as a true development methodology.

3.1 Background for XP

Making software is difficult - making good software is even more difficult.

According to [10] software development in the 1990s were influenced by two major topics. Internally in the industry *procedural programming* were replaced by *Object oriented* programming as the favorite development paradigm. Externally the rise of the Internet and the "dot-com boom" which emphasized speed-to-market and company-growth as competitive business factors. This lead to re-

[1] C3 was a project run by the Chrysler Corporation. The project started in 1995 and the result should replace several payroll application within the organization with one single system.

quirements which changed fast and the production cycles became shorter. Many of the traditional software development methodologies were not geared for these short cycles leading to many poor quality expensive software projects. The community needed more lightweight methodologies and this was the starting point of *agile software development*.

Agile software development is a conceptual framework for developing software through an evolutionary change throughout the entire life-cycle of the project. Most agile methods develops software in iterations of short time - typically between one to four weeks - and each iteration is a small development project of its own. By the end of each iteration a full functional products is finished thereby increasing the functionality of the overall project. XP is indeed an agile methodology and perhaps the flagship of agile software development.

3.2 What is Extreme Programming?

So what is Extreme Programming? The short explanation is that XP is a methodology that through twelve practices used on a daily basis builds on five values. The goal is a productive development team that makes software of a high quality, delivered at the agreed deadline and to the agreed price - and at the same time software development should be great fun.

3.2.1 Risks of software development

There are a number of risks when developing software. Missed deadlines, defective software, misunderstood requirements and thereby missing and/or unnecessary functionality are just a few of them. Eventually after a few year developers are sick and tired of the development project and start leaving the company and the project.

3.2.2 Four factors of XP

XP builds on four unknown factors; *cost* of developing the software, *time* to develop it, *quality* of the resulting software and *scope* of the resulting. Customer and management may change the first three but only the development group can change the the scope of the project. Often scope is completely neglected but in software development and especially in XP scope is very important. Often the customer only have a weak idea of the exact functionality and which functionality will be most valuable to him. During the development process new ideas come up, the domain changes and when the customer received the first version he realize what functionality the really wanted from this version. The requirements to the software changes during the development process.

Therefore the idea in XP is to split the entire development process up into small iterations typically lasting between one to four weeks. And the end of an iteration a finished part of the software is delivered. Before the development process is started a series of deadlines - one for each iteration - is set, the quality level is decided and the cost of the software is set. This is often decided by the management or the customer. Based on these decisions the development team sets the scope of the software within the limitations and setting the scope of each iteration. As the project changes so does the scope of the following iterations making the process more responsive to the customers needs.

3.2.3 Five values of XP

- **Communication** is important in a software development process. Most of the problems arising in a project can be lead back to lack of communication either among the developers, between the developer and the customer or between the developers and the management.

  XP promote communication through different working habits. Automated test, pair programming and estimating tasks all requires communication.

- **Simplicity** is actually very complicated. Another way of stating simplicity is *develop what you need today - not what you think you need tomorrow.* The core of the statement is that you do not know exactly what you need tomorrow and therefore you should not plan ahead. Code exactly what you need right now - and do it as simple as possible. This way the code might be changed a bit if needed without time wasted.

- **Feedback** is essential to XP. Feedback works in two different time frames. The developers writes test cases for all possible failures they can think of. When running a test cases in the application code the developer gets immediate feedback on the correctness of the implementation.

  The second time frame is on a longer term. On every release the customer tests the application and holds it up against the user stories to ensure the functionality is as expected. This feedback gives a picture of the projects current condition and the plan for the project may be adjusted accordingly.

- **Courage** is hard to define. It requires courage to not think about the future but only code what you need today and not thinking about what you need tomorrow or in a week. It requires courage to refactor code which possibly means you have to change entire parts of the application. It requires courage to throw away code and start over with a new implementation. In general a lot of the practices in XP requires courage as they are pushed to the extreme.

- **Respect** was not one of the original values but have become apparent during the maturing of the methodology. Developers must have respect to each other e.g. by not integrating code pieces that break existing tests or delay other developers. Developers also must respect their work and aim at high quality and the best design for a solution.

3.3 Practices of Extreme Programming

Extreme Programming has twelve practices grouped into four areas.

3.3.1 Shared Understanding

3.3.1.1 Coding Standard

Since all developers are going to change different parts of the system and refactor the previously built parts the development team must agree up on a common coding standard which is used for the entire project. The standard should demand less possible work and still be consistent. The standard should also emphasize the procurement of the code.

Often the standard conventions specified by the language vendor is used but the team may decide on another standard.

3.3.1.2 Collective Code Ownership

If a developer finds an error or a possibility to improve the code, the developer is obligated to do so. All developers are responsible for the entire code and everybody knows at least the overall functionality of the different parts.

In the earlier days the ”rule” were no ownership. If a developer needed to change the code he just did without considering the rest of the system. The code evolved fast but got unstable just as fast. Then individual ownership emerged where only the developer who wrote a piece of code were allowed to change it. Other developers could propose changes and then wait for it to be implemented. The code now were stable but evolved slowly. Collective ownership is the combination of the to making fast evolving and stable code.

3.3.1.3 Simple Design

Simplicity in the design of the system is central in Extreme Programming. When implementing the functionality at hand the simplest possible approach should be chosen. Only program exactly what you need to implement the functionality without thought of future needs.

By making the design as simple as possible the cost of changing direction in the middle of the project is much less expensive since no preparations for new functionality has been made. Along with refactoring the simple design ensures a very flexible code.

3.3.1.4 System Metaphor

A metaphor is a designation used to describe something to ensure a common understanding of it. Everybody in the development team should have a common understanding of all the metaphors of the system so they will be able to communicate without misunderstandings. The metaphors is also used in the code - and the development team should agree on a common way to name classes and methods so everybody knows what e.g. a method does from its name.

3.3.2 Fine Scale Feedback

3.3.2.1 Pair Programming

All code is produced by two developers working on one task at one workstation. The two developers has two roles. The one with the keyboard thinks about how to implement the current functionality the best way while the other thinks in the bigger picture of the entire application. Roles are traded regularly.

The pairs are not fixed which ensures that all members of the development team has some understanding of the entire application.

3.3.2.2 Planning Game

Planning software is a ongoing process throughout the project with a dialog between the possible and the desirable. Neither business considerations nor technical considerations should be dominating. The customer must decide the scope of the project, the order of priority, the content of each release and the dates of the release. The development team must decide on estimates of functionality, consequences of decisions, the development process and the detail planning. The business considerations cannot be decided on by the customer alone as the technical considerations might have an impact on these.

3.3.2.3 Test Driven Development

Every functionality in the application is tested. This is done using automated unit tests which is constructed to test specific units of the code. The test is written before the actual application code to stimulate the developer to think of every possible way the functionality could fail. When the developer cannot think of more ways the code may fail the code is finished.

Tests is not needed for every function in the code. If the code cannot fail there is no need for a test - e.g. it is not necessary to test simple properties of beans (unless the have some special functionality or conditions which is rarely the case). The entire collection of test improves the trust of the application since every bit for expected behavior.

3.3.2.4 Whole Team

Whole Team is sometimes also known as Customer Presence. The customer should be present in the development group for discussions and questions on all aspects of the customers requirements to the application. This insures the inside knowledge of how the application should be used and respond.

In this context the customer is the person who should use the application - and not necessarily the one paying for it. If the application is a customer service system then the customer should be a customer service assistant. There are by the way different opinions on whether the customer should be in the group physically at all times or should be at hand at all times.

3.3.3 Continuous Process

3.3.3.1 Continuous Integration

New code is integrated and tested every few hours - and at least once a day. If the test fails all problems is solved at once to ensure the integrity of the current version in the repository.

Different developer teams might have their own version with various changes saved locally. By integrating small pieces of code it is easier to find and solve problems. Additionally integration problems are solved when the code still is fresh in the developers memory instead of postponing it to a major integration step.

3.3.3.2 Design Improvement / Refactoring

When implementing some functionality developers should consider if changing some of the existing code will make the implementation easier and more versatile. After every new implementation developers should again consider if changing code will simplify things. This is known as refactoring.

Refactoring is a necessity in Extreme Programming due to the principle of only implementing the needs for the current functionality without thinking ahead on what might be needed later in the project. This principle may lead to messy and duplicate/similar code which makes the application hard to maintain and develop further.

3.3.3.3 Small Releases

The entire project should be split up into a number of small releases. Every release should contain as little additional though complete functionality as possible compared to the previous release - but enough to add value to the product.

Making small releases has several purposes. First of all the customer can follow the development process and see the product as it evolves. For every release some new features has been added to the product and this strengthens the confidence in the development team. Also delivering a piece of work which is finished and ready to ship to the customer improves the moral of the development team. Besides that progress of the entire project is easy to follow.

3.3.4 Programmer Welfare

3.3.4.1 Sustainable Pace

A working week should therefore be between 35 and 45 hours depending on the individual. Overtime is not allowed more than one week at a time.

Software developers are human beings like everybody else. Therefore they have the same needs for spare time to sleep, family, sports or any kind of activities not related to work. The idea is simply that people performs best when they are rested.

In [6] Kent Beck also makes a remark on vacation as americans opposite europeans rarely has more than two or three days of vacation in a row. Longer vacations are just as important to the welfare of workers.

Not all twelve practices are invented by Kent Beck during the development of XP. Several of them build upon well known principles of software engineering. The twelve practices are carefully selected so that they support the weaknesses of one or more of the other practices. E.g. you might claim that pair programming will never work as it is too slowly and the two developers might not agree on the solution or the code. But if the coding standard is set, there is no arguments on trifling matters. If the working week is 40 hours long everybody are rested as the day begins reducing the risk of trivial discussions. And if the two developers start by coding the test-case that should test the functionality they will gain a common understanding of the functionality supported by the system metaphor and a simple design. And then the development process will not be slow and there will be no arguments on the solution. On the contrary the solution will be implemented in the best way as two developers have approved the implementation [2] The same way the other practices interact making XP a versatile and strong methodology.

3.4 Strategies

3.4.1 Planning

In [6] Kent Beck states that

[2]Similar descriptions of interaction between the practices may be found in [6].

Planning is all about guessing what it will be like to develop a piece of software with a customer.

Note that he on purpose uses the term *with* the customer. The sentence might be read as planning with the customer or developing with the customer - and this is exactly the purpose of planning. In XP both the planning and the development is done *with* the customer.

The planning is built on five fundamental principles of XP [3]

- Only plan a short period of time
- Accept responsibility
- The developer doing the work should estimate it
- Ignore dependencies between different parts of the system
- Prioritize the work

The planning process is called the Planning Game in XP as it might be compared to a game.

- **The goal** of the game is to maximize the value of the software produced during the game.
- **The strategy** of the game is to invest as little time and resources as possible in the development process by developing the part of the system with greatest value to the customer as fast as possible (though still considering the programming- and design strategies agreed upon to ensure quality). When the first delivery has been delivered the customer will know what then becomes the next part with highest value.
- **The pieces** are *history cards* with *user stories*.
- **The players** are the customer and the development team. The development team are the those developing the software whereas sometimes it might be unclear who the customer is if the software is an off-the-shelf item not developed for a specific customer. Then focus-groups or representatives from the customer mass (e.g. expert users who knows the system in and out) might be used as the customer player.

[3] These are the most important principles for the Planning game. More principles are covered in [6].

- **The moves** are divided into three phases; the *exploration phase* where decisions on system functionality is made, the *commitment phase* where the decisions on priority is made and *steering phase* where the project is led in the correct direction as reality affects the plan.

The planning game is divided into two parts each containing the three phases.

3.4.1.1 Release planning

The *release planning* part of the game is focused on determining what requirements should be included in which release and when the releases should be delivered. Both the customer and the developers takes part in this planning.

Exploration phase This phase focuses on gathering the requirements and estimating the work needed on every requirement. The customer has a problem which should be solved by the software. The problem is described as a user story by the customer telling what the system should do. When a story is finished the development team estimates the time it takes to develop and implement a story. If it is not possible to estimate the story the customer may have to split the story into several small stories.

Commitment phase The commitment phase focuses on committing to the development. The customer sorts the user stories according to the value of the story. The developers sort the stories according to the risk based on how accurate they can estimate the risk. The development team the tells the customer at what speed they can perform the project and then the customer select the scope of the first delivery. The customer may either set a deadline and then choose stories that according to the speed of the developers - or the customer may choose the stories and then set the deadline accordingly.

Steering phase Within the steering phase the the plan may be updated due to the lessons the developers and customers learn - thereby steering the project in the correct direction. Updates are changes in the plan such as different priorities of stories, lower development speed, new stories or even new estimates of all stories in the project.

3.4.1.2 Iteration planning

The *iteration planning* part of the game focusses on planning the activities of the development team. The customer is not involved in this part.

Exploration phase The exploration phase is used to extract tasks from the stories and estimate their implementation time. Tasks are smaller than hole stories and often a task compliment more than one story. If a task is to difficult to estimate the task should be split up into smaller tasks.

Commitment phase In the commitment phase a programmer commits to a task. As the programmer is responsible for the task he should also estimate the task. Each programmer sets their load factor which essentially is the effective days for development divided by the number of workdays (e.g. if a week has 40 hours and 8 are used for meetings then there will be less than 32 hours for development giving a load factor of 40/32 or 4/5 which is for development days out of five.). Each developer compares the estimates of the tasks with their load factor and the tasks are balanced out on all members of the team.

Steering phase The implementation of the tasks is done during the steering phase. The procedure is get a card, find a partner (for pairprogramming), design the task, write a unit test for the solution, write the code, run the test, refactor the code and end by running functional tests according to the requirements of the user story.

The release planning is done at the beginning of the project and the iteration planning is done during each iteration (which actually resides inside the steering phase of the release planning). An iteration is typically lasts three to four weeks.

3.4.2 Development

The development strategy is probably the most radical of the strategies compared to traditional methodologies. The development enforces continuos process, collective code ownership and pair programming as described in the XP practices above. As describes these practices are very new and often hard to implement for experienced users.

3.4.3 Design

The design strategy of XP short - simplest possible implementation. There are four rules which encapsulate this in short:

- The system should communicate everything you would like to communicate - including both application code and test.
- The system cannot contain duplicate code
- The system should have as few classes as possible
- The system should have as few methods as possible

This way of developing will work due to the short iterations and small releases, as it is cheaper to change simple design than complex design.

3.4.4 Test

The test strategy may be encapsulated in

- The test cases is written before the application code and thereby leading the way for the implementation.
- The test cases is derived from the customer requirements (user stories)
- The developer writes unit and integration test.
- The customer writes function tests

Part III

e-assesment system

Chapter 4

Introduction to the project

4.1 Assessment of safety and health

All companies in Denmark with at least one employee are as of year 2000 obligated to prepare a written assessment of the safety and health conditions at the workplace due to the *Consolidated Danish Working Environment Act* (see appendix B). In Danish this assessment is named an *Arbejdsplads Vurdering* or APV for short. The purpose of the APV is to ensure that the work on safety and health in the company encompass all significant working environment problems. This implies that the APV is a tool for the company and it is not to be reported to any governmental institution though *Danish Working Environment Authority* oversee that companies observe the statutory order.

The company is obligated to involve the safety organization of the company in the process of planning, implementation, follow-up actions and revision of the APV. The scope of the APV depend on the complexity of the work, technical equipment, chemical substances or materials, working methods and the size and arrangement of the company.

Companies a free to choose the methodology used for the APV but it should be well suited for mapping out the significant problems. Some companies choose a questionnaire while other interview all employees depending on the culture and the size of the company. The Danish Working Environment Authority has devised 60 checklists with a set of questions aimed at different business sectors. Whatever methodology is chosen it should contain five elements:

- Identification and mapping of working environment conditions.
- Specification and assessment of working environment problems
- Include the sickness absenteeism of employees in order to assess if working environment problems affect this.
- Preparation of action-plans for the purpose of solving the problems - and an order of priority of these.
- Guidelines for following up on action-plans

The APV must be revised if the work or the working method changes such that it affects the working environment. This may be triggered by new knowledge or a workplace accident.The APV must be revised at least every third year, since revising the APV on a regular basis helps the company on systematizing the ongoing work with safety and health.

4.2 Thermo Fischer Scientific and APV

Nunc [1] is one of the world leaders in the production of high tech disposable plastic ware for biotechnology, pharmaceutical and research laboratories. The company was founded in 1953 in Roskilde and now resides just outside Roskilde with a large production facility with numerous casting machines producing high quality products round the clock. The company have around 500 people employed in production, development, sales and marketing, distribution and administration. In december 2006 the company became part of the american Thermo Fischer Scientific group and this name were adopted shortly after though all products is marketed under the NUNC brand.

At Thermo Fischer Scientific the APV process is implemented through a questionnaire based on the checklists devised by the Danish Working Environment Authority. Though the questionnaire is aimed at the plastic producing industry the checklist is very general and therefore is not necessarily an effective tool for assessing and improving the working environment. At Thermo Fischer Scientific there is a lot of focus on the safety and health of the employees and therefore

[1] Nunc is latin for now and symbolizes innovation

a better tool is needed. The department responsible for safety and health have devised their owen questionnaire with 125 questions divided on 16 categories. Each question may be answered with one of the following: *Not relevant*, *Problem*, *OK* or *Good*. This gives the safety organization a more nuanced picture of the working environment condition.

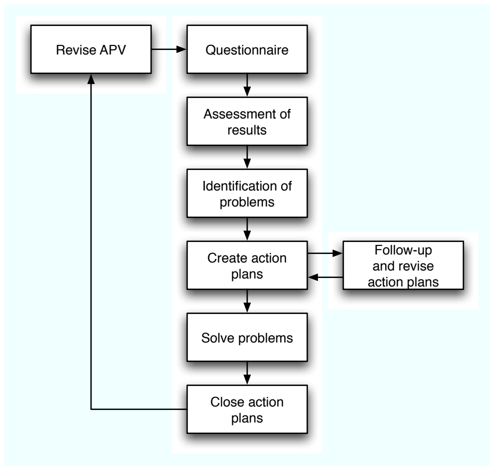


Figure 4.1: An overview of the APV process at Thermo Fischer Scientific

The questionnaire is a twelve sheet paper schema which every employee must fill out. The company is divided into a number of departments and each department has a safety responsible or a safety group. Their job is to collect all schemas from the employees in the department and extract the results from the questionnaire by counting the number of answers for each question. This is a cumbersome and time demanding work which often ends as a low priority task in a busy company.

The objective of this project is to develop a electronic system for the assessment of safety and health for the company. The environmental act allows the written assessment to be in an electronic format as long as it fulfills the requirements of the act and supports the APV process. (see figure 4.1 for an overview of the process).

4.3 Existing application

As the APV has been mandatory for companies for seven years and carrying out a manual APV is a time demanding job several electronic solutions are available.

The three most important are:

- **Orbicons APV** which probably is the largest product available. The standard checklists devised by the Danish Working Environment Authority is available and it is possible to construct your own schemas based on the questions from the checklists. The solution is web based and has three levels of user access: administrator, safety group and employee. Unfortunately you are not able to add your own questions which makes it inflexible - and additionally the subscription price is quite high. Orbicon[2] is part of the Danish Company Hedeselskabet.
- **NemAPV** is a web based solution developed *defgo.net* [3]. The specification states that the solution contains the standard checklists but nothing about the possibility of creating your own questions. The specification does not state anything on results or the possibility of dividing the assessment based on department. Additionally it is not possible to write remarks to answers.
- **APVplus** is developed by Særkon Kommunikation [4]. The solution contains a lot of the working environment information available from the Danish Working Environment Authority and the working environment act. The system is not web based but consists of a database placed on a central server, a client used to fill out the questionnaire and an administrative client to extract results.

Common for most of the electronic solutions (including the three mentioned above) are the method chosen. They are based on a questionnaire with the standard questions from the checklists without the possibility to add new questions or response possibilities. Though the schemas are a little different and there are different possibilities on extracting results and adding action-plans the different solutions available are based on the same process making the solutions almost the same.

[2]www.orbicon.dk
[3]www.defgo.net and www.nemapv.dk
[4]www.apvplus.dk

Chapter 5

Project specification

The objective of this project is to specify, develop and implement an electronic system for assessment of health and safety based on the needs of Thermo Fischer Scientific. The sections in this chapter states the overall specifications for the project and system. The more functional specifications is discussed in the next chapter.

5.1 General considerations

5.1.1 User characteristics

The system should have three types of users; *administrators*, *safety group members* and *ordinary users*. Administrators of the system will probably be the safety and health managers, the system administrator and production managers. Common to the administrators is they are all experienced computer users with understanding of navigation web sites.

The departments and the management should select a group of members in the department as the safety group. These users are - like ordinary users - often unskilled workers who might not be very experienced with computers.

Additionally Thermo Fischer Scientific employ many people with different ethnic backgrounds some of who might have difficulties understanding danish technical terms and guidelines.

5.1.2 Assumptions and constraints

It is assumed that Thermo Fischer Scientific has examined the solutions available for making the APV process electronically and have found no suitable solution - and that it will feasible to develop an individual solution for their needs (*a feasibility study*).

It is assumed that computers with web browsers are available to all users of the system.

5.2 Requirements

The developed solution must comply with a set of requirements. The requirements are stated by the different stakeholders who all have different interests in the system. These stakeholders include

- **Users** using the system (includes safety group members)
- **Administrators** administrating the system and using the results to administrate the working environment. Administrators also reports to the management.
- **Operation manager** is responsible for the operation of the system on a daily basis. This is the system administrator of the IT department.
- **Developer** who develops and maintains the system.

5.2.1 Non-functional requirements

Non-functional requirements are requirements that are not directly functionality related. They are very general and are related to the operation and dependability of the system.

5.2.1.1 Safety and reliability

The system should have a maximal uptime and be available to the users at all times. The system must register all entries without errors and notify the user is he acts erroneously.

The system should be secured against unauthorized access through a login and password system. The functionality of the system should be secured through a privilege system with three different user types.

5.2.1.2 Interaction

The system should be web based and interact with the user through a web interface thereby being accessible on all computers in the company. The web interface should be structured in a simple and intuitive manner making it easy for even unexperienced users to use the system. The development should aim at using keyboard interactivity only.

5.2.2 Domain requirements

Domain requirements are requirements extracted from the domain of operation. These are often related to technical or legal matters of the domain.

5.2.2.1 Technical requirements

The data of the system must be saved in a Microsoft SQL Server database, as the company has one of these available. Additionally the system must run on a server in the company thereby only being accessible through the internal company network.

The system should not be limited to the currently specified functionality but be possible to extend with new functionality.

5.2.2.2 Legal requirements

The system functionality must comply with the requirements set by the Working Environment Act (see appendix B).

5.3 Technology

For this project the Java Enterprise Edition 5.0 platform is chosen. JEE is the enterprise edition of the Java programming language adding features for networked enterprise applications. The platform builds on top of the traditional Java language and thereby have access to the entire Java API.

The e-assessment system will run on a JBoss Application Server which is one of the most widely used JEE servers. JBoss AS is written purely in Java which makes it platform independent. Additionally it is easy to install. It comes in a package which is unpacked. Inside the package is a *run* script which starts the server - and within a few minutes the server is up and running. The current version includes EJB3.0 and requires a Java 5.0 runtime environment.

On top of the JBoss AS the JBoss Seam framework is used. The framework connects the Enterprise Java Beans of the application with the Java Server Faces view in a neat and simple way. Additionally it bring some extra functionality and improved context handling making the server environment very powerful.

5.4 Methodology

Extreme Programming programming is a relatively new development methodology which distinguishes it self from older methodologies in the focus on coding and testing as soon and as little as possible rather than analyzing the entire project, then code and end up with testing the application. This makes the it easy to change the requirements through the development process without affecting to much work already done. The aim is that it should be great fun to develop high quality software.

The rest of this part of the report will be structured according to XP with a planning, development, implementation and test though the content of the section might not follow the XP guidelines strictly.

5.5 System metaphor

The terms stated below will be used throughout the documentation of the APV system. The terms recur in the application code - many of them as classes.

- **Question** - A question part of the questionnaire and should clarify possible problems - e.g. *Are accidents investigated to avoid recurrences?*.
- **Category** - The questions is divided into categories e.g *Accidents*. A question may only be related to one category.
- **Schema** - The categories with questions make up a schema. A category may only be related to one schema. A schema is similar to the standard checklists devised by the Danish Working Environment Authority. The schema may be used for several revisions of the APV.
- **Review** - A revision of the APV is done through a review. A review is based on a specific schema and may be carried out in several departments. If two different departments use two different schemas in the same revision, two reviews should be created connecting the departments with the correct schema.
- **Department** - The company may be divided into several organizational sectors - these are named departments. Reviews and action-plans are connected to departments making the extraction of results and statistics flexible and easy to inspect.
- **Value** - The severity of a problem clarified by a question is ranked by a value - e.g. *Problem* or *Good*.
- **Answer** - An answer is related to a specific question made by one employee. The answer contains a value and a remark to the question if set.
- **Evaluation** - An evaluation is the collection of answers to a schema made by one employee. The evaluation is connected to a review (and thereby to a schema) and the department of the employee.
- **Result** - The result of a review collect all answer of the evaluations related to the review. The resulting values for each question is calculated from the answers as a percentage of all answers.
- **Action-plan** - If the result of a review identifies a problem, the safety group should create an action-plan. The action-plan is related to a specific question and the result for a specific department. It has a deadline and a responsible for taking action.

Follow-ups on action-plans are modeled as an new action-plan with relation to the old version.

- **User** - A virtual user has access to the APV system. A user could be related to a physical user or a safety group depending on the company policy.
- **User Type** - The system has three user types - administrator, safety group member and user. The users privileges in the system is dependent on the user type.

Chapter 6

Planning

6.1 User stories

The non-functional and domain requirements to the APV system is stated in section 5.2. These sets the basic technical requirements to the system but states nothing about the functionality. The functionality requirements or user requirements are described through user stories written by the customer.

To give an overview of the user stories, they are grouped together in nine groups based in on their related functionality. The original groups were

- User administration
- Group administration
- Privileges administration
- Department administration
- Schema administration
- Evaluation

- Results
- Action-plan administration
- Statistics

6.1.1 Changes in user stories

A lot of the user stories have changed from the first version to the current. During the implementation of the functionality of the first iteration new experiences were gained and this caused some drastic changes. The major changes is described here.

6.1.1.1 Splitting user stories

Originally the *Create schema* (C.3.1) were simply stated as one long story. It described a long and rather complex sequence of steps with a lot of digressions from the core functionality. The story were split up into eight user stories which specifies a simpler functionality required. The benefit is a more precise description of the individual tasks and an individual order for priority for the stories as not all of them are equally valued. E.g. creation of category and questions inside the schema have a much higher value to the customer than retiring these, as the customer actually can create a schema and use it for a review without the possibility of retiring.

6.1.1.2 Review added

The user stories did not exactly specify how the evaluations entered by the users and a schema was connected. The original thought were to simply relate a schema to a department and thereby to the users.

This soon turned out to be a bad solution as the schema can only be used for one revision of the APV. As a consequence a new element - the *review* - were added along with user stories for it (C.4). This gave a looser coupling between the schema, the department and the evaluations entered. One of the benefits is the possibility of using the same schema on several reviews - and thereby on several revisions of the APV. This opens the opportunity of detailed statistics over several revisions as the same questions are evaluated over again.

6.1.1.3 Groups and privileges removed

The original set of user stories had two additional groups besides the above mentioned. The idea was to divide users into group with different privileges to the system. The privileges would then be set either on the user, the group of the department. During the development of the the development of the *Create a review* (C.4.1) and through conversations with the safety and health manager this construction appeared to be to complex for the needs.

Therefore the groups and privileges were removed. Now users are given a user-type (*administrator*, *safety group member* and *user*) which determine system privileges and users are connected to at least one department, which determines which reviews the user can access to either carry out an evaluation or only see the results. This simplification resulted in a simpler user administration though it still satisfy the requirements of administrating access.

6.1.1.4 New requirements

Half way through the first iteration the project were presented to the entire safety organization at Thermo Fischer Scientific. The discussion of the project caused two new requirements that was not covered in the original set of user stories.

In the APV system a schema is created and used in an electronic format. As not all employees in the production sectors of the company have access to computers the schema is also published on paper and the results are entered into the APV system by the safety group of the department. The paper schema has so far been implemented in Microsoft Excel - but as the APV system already implements a schema, it should be possible to print the schema e.g. as a *pdf* file. The functionality is described in C.3.3.

The american owners of the Thermo Fischer Scientific group are very focused on statistics. Their preferred chart is Pareto Charts[1] and therefore it should be possible to plot the results of an APV preview as such. The functionality is described in C.6.2.

The final user stories can be found in appendix C.

[1]A Pareto Chart is a bar chart where the values are arranged in descending order. It is named after Vilfredo Pareto who was a french-italian sociologist, economist and philosopher.

6.2 Value and Risk

The customer value of the functionality each user story describes is assessed from the amount of work it would simplify during the APV process and how it interacts with other user stories. The highest value functionality is easy to point out whereas the lower priority is difficult to prioritize among. In general *create* and *edit* functionality is prioritized higher than *retire* and *delete* as they are useful to the customer in the preparation of and during the APV process whereas nothing is retired or deleted from the system until the entire process is ended.

The risk assessment of a story is based on experience and the presumed complexity of implementing the functionality. Each story is given one of three values:

- **Low Risk** are relatively simple implementations. Typical examples are creating, editing, retiring and listing objects.
- **Medium Risk** are more complex implementations where the developer has some idea of how this might be done.
- **High Risk** are either very complex functionality, special implementations or cases where the developer have to search for a solution. These cases are often time consuming.

Table 6.1 shows all user stories of the APV system sorted by the value they bring to the system. Along with the value the risk is also stated. Both customer value and risk is noted on each user story in appendix C.

6.3 Releases

Based on the user stories the development of the APV system is divided into three iterations. The horizontal lines in table 6.1 indicates which user stories is in which iteration. The distribution of the work amount and time required is very difficult to estimate, but based on prior experience loose estimates have been made. The three iterations is estimated to last four weeks each - probably with an extra week for the first release caused by the setup of the project and configurations.

Customer Value	**User story**	**Developer Risk**
1	View Results of a review	High
2	Enter an evaluation	Medium
3	Create a schema	Medium
	Create a category	Low
	Create a question	Low
4	Edit a schema	Medium
	Edit a category	Low
	Edit a question	Low
5	Create a review	Low
6	Create an action-plan	Low
7	Close an action-plan	Low
8	Create a follow-up action-plan	Medium
9	List action-plans	Low
10	Create a user	Low
11	Edit a user	Low
12	Print Schema	High
13	Log in	Medium
14	Log out	Low
15	Close a review	Low
16	Retire a review	Low
17	Create a department	Low
18	Edit a department	Low
19	Create a new revision of question	Low
20	Retire a question	Low
21	Retire a category	Low
22	Retire a schema	Low
23	Delete user	Low
24	Extract results of review...	Medium
25	Extract statistics of action-plans...	Medium
26	Extract statistics of schema...	Medium
27	View results of a review as Pareto Chart	High
28	Retire a department	Low

Table 6.1: The user stories of the APV system listed in customer value order.

6.3.1 First release

The first release implements the core process of an APV revision. In the manual APV process used at Thermo Fischer Scientific the most time consuming part is collecting and calculating the result thereby mapping the problems in safety and health. An automated result calculation would bring high value to the process.

To calculate the results the system must have some input from the process. This is done by user entering evaluations. Evaluations is answers to a schema containing question grouped into categories and therefore the functionality to create schemas, categories and questions along with editing them is required. The result, evaluations and schemas are related through a review and therefore this is also required.

As the results of a review is divided on and connected to departments these are needed too. As the departments resemble physical departments these rarely changes and in the first release these will be modeled as static data in the database.

In addition to the described functionality the layout of the web interface will be produced and implemented into the system with this release. This way the user interface can be evaluated early in the process and the customer sees some progress on the system implementation. The consequence is that parts of the interface will be disabled.

6.3.2 Second release

The second release implements two main functionalities - action-plans and user/privilige administration.

The action-plans are part of the core APV process but not as highly valued as the results of a review as the action-plans are the consequence of the result pointing out a problem. The action-plans could might as well be handled on paper or in a text document.

The user administration is mainly a supporting functionality which administrates what different users can do inside the system. These privileges are related to a user-type of the user logged in. Both the user-types and the privileges on different functionality are static and built into the APV system. Figures 6.1 6.2 and 6.3 shows the functionality each of the user-types are privileged to use.

Along with a system for administrating the users the functionality to enforce the privileges must be implemented in all parts of the system. This is a quite big operation and therefor the number of user stories completed in this release is limited.

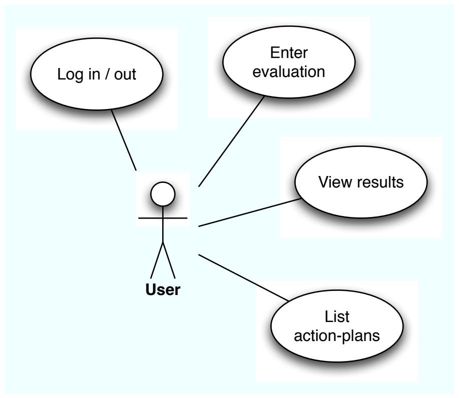


Figure 6.1: The functionality a user is privileged to use

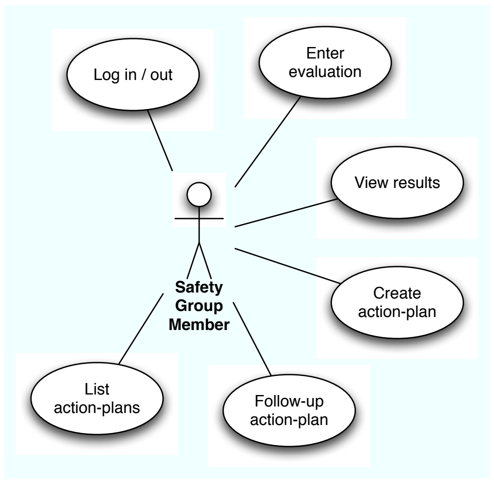


Figure 6.2: The functionality a safety group member is privileged to use.

6.3.3 Third release

The third release contains the rest of the functionality described in the user stories collection. This release has three main areas.

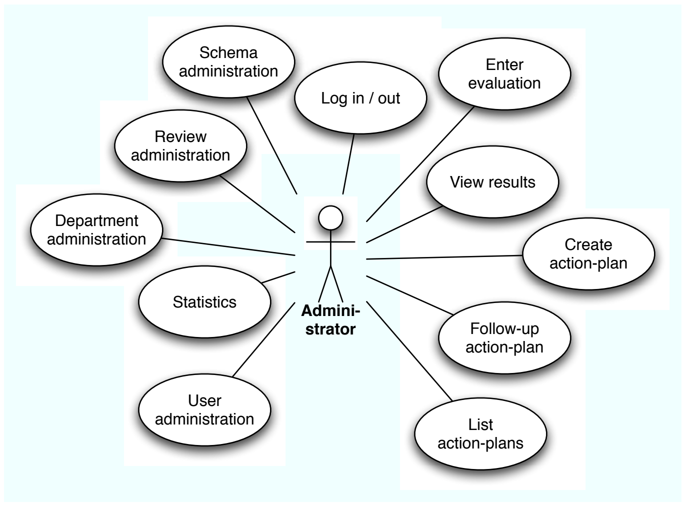


Figure 6.3: An administrator has privileges to all functionality in the APV system.

The functionality of creating and editing elements of the APV is contained in the two previous releases. This release implements functionality for retiring the elements. The term *retire* should be understood as *removed from the active part of the application but not deleted* or perhaps as *archived*.

The administration of departments is postponed to this release as it - as stated earlier - is an almost static type.

Output is another keyword for this release as the functionality for extracting statistics to comma-separated files is implemented along with the possibility of printing the APV schema. These user stories are marked with *medium* or *high* risk due to the work required. The information must be generated and then formatted to either comma-separation for the statistics or a *pdf* file for the schema and then sent to the client. Especially the amount of work required in formatting the output is very uncertain.

CHAPTER 7

Design

7.1 The web application

The web application of the APV system is the user interface where the user interacts with the system. Therefore the design of the web application is an important matter for the success of the entire system.

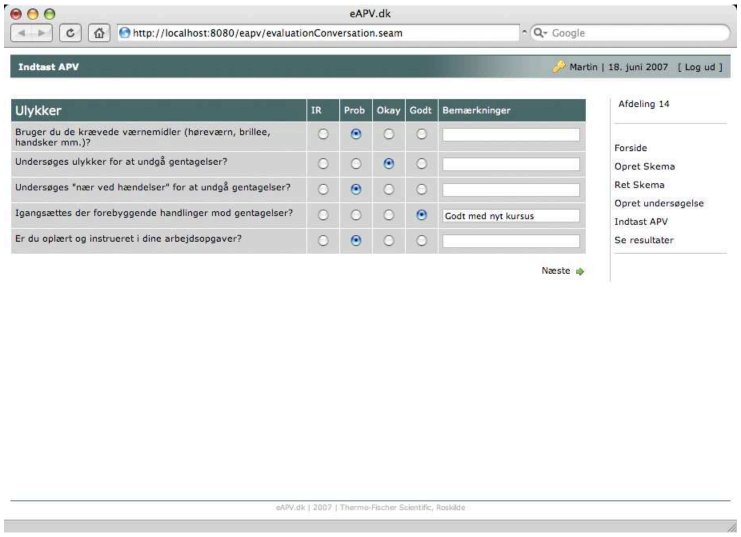
7.1.1 Requirements for the web application

When designing the web layout for a web application, it is important to bear the purpose of the application and the target users in mind. Different purposes requires different layouts for different users.

The purpose of the APV application is as an internal tool in the company providing value to the APV process. It is to be used as an application along with a financial system or a document management system.

The users of the system spans from experienced users to users with little or no experience - and even users without good danish reading skills. As the questions is formulated in danish the application elements should not take the focus from the questions.

On the basis of these two premises a simple and concise web layout should be created. No superfluous elements should draw the focus away from the APV process. Therefore web layout will contain a small header indicating the current task, a simple menu for navigation and a centralized area for the core functionality. The colors are kept in white, grey and a dark green with black text and there will be no banners, dynamic elements or anything else disturbing the eye. The web layout for the APV system is shown in figure 7.1.


eAPV.dk
http://localhost:8080/eapv/evaluationConversation.seam
Google
Indtast APV
Martin | 18. juni 2007 [Log ud]
Afdeling 14
Forside
Opret Skema
Ret Skema
Opret undersøgelse
Indtast APV
Se resultater
Næste
eAPV.dk | 2007 | Thermo-Fischer Scientific, Roskilde

Ulykker	IR	Prob	Okay	Godt	Bemærkninger
Bruger du de krævede værnemidler (høreværn, brillee, handsker mm.)?	○	●	○	○	
Undersøges ulykker for at undgå gentagelser?	○	○	●	○	
Undersøges "nær ved hændelser" for at undgå gentagelser?	○	●	○	○	
Igangsættes der forebyggende handlinger mod gentagelser?	○	○	○	●	Godt med nyt kursus
Er du oplært og instrueret i dine arbejdsopgaver?	○	●	○	○	

Figure 7.1: The web layout for the APV application.

7.1.2 Implementation of the web application

The user interface is implemented using the *MyFaces* JSF library distributed with JBoss Seam. This makes a strong framework with lots of preimplemented functionality.

The application will use the MyFaces Servlet described in section 2.1.5 as the *controller* of the application. The *view* is constructed with *xhtml* files which contains JSF and HTML code inside a `ui:construction` block.

7.1.3 Template and elements

The MyFaces Servlet uses a template to construct the basic layout of the web page e.g. the top bar with logged on user and date. The template includes three `ui` blocks from a website:

- **pagename** is a small block which only contains the name of the page. The block is written in the top bar of the page.
- **content** is the main container for the content of the page. This block contains the forms and lists that make up the functionality of the application.
- **sidebar** is a small content block used in the right lower side of the page. It is used for e.g. a clickable list of categories in the schema editor and when viewing the results of an APV review.

An `ui` block is defined using the `ui:define` tag with a name attribute.

```
<ui:define name="pagename">
   <h:outputText value="#{messages.PageHeadFrontpage}"/>
</ui:define>
```

The above defines the content of the `pagename` block which in this case is a text from a language file (the name of the frontpage).

In addition to the three block defined in the emphxhtml file for a view page, the template file includes the `sidebar.xhtml` file. This file is inserted in the upper right corner of the page and contains the department selector and the application menu. The department selector selects the department used for the current session. The available departments are based on the departments the currently logged in user is related to and his privileges (as administrators have access to all departments). The menu likewise is generated dynamically based on the privileges of the user and the currently selected department (though this is not fully implemented in release 1). The three `ui` blocks and the menu `sidebar.xhtml` is indicated in figure 7.2 and the code for the template can be found in appendix D.1.1.

The flow of the application may be illustrated as in figure 7.3.

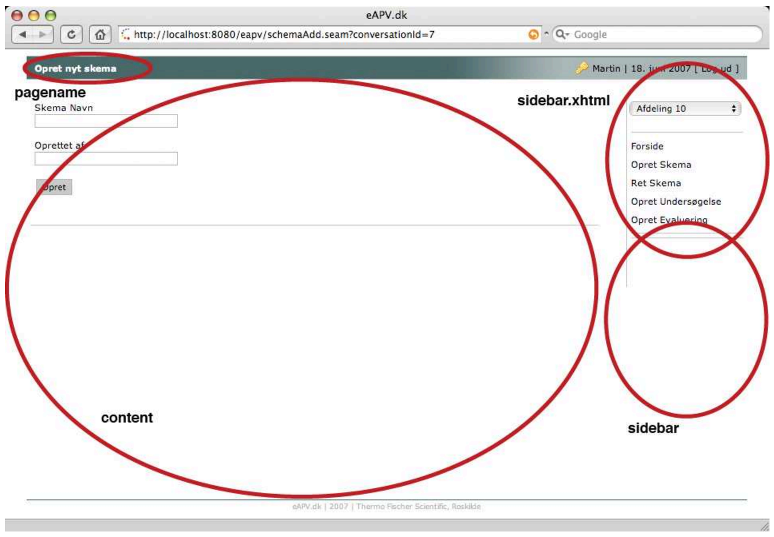


Figure 7.2: A web page is composed by three elements which is included into a template file.

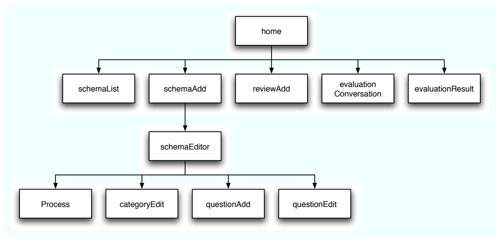


Figure 7.3: The flow of the application may be illustrated as above. The names refer to the filenames of the *xhtml* files.

7.2 Domain Data

The domain data is implemented as entity beans. The different entities is derived from the user stories written by the customer as the thing some operation is used on. Figure 7.4 shows the entity beans of the APV system and how they are related.

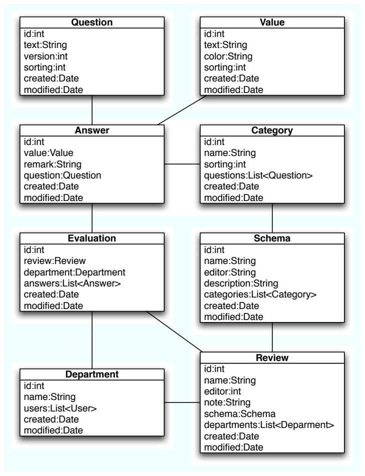


Figure 7.4: The domain data is modeled in entity beans connected as shown.

As described in section 1.2 the beans are simple POJOs with some properties and a getter and setter method for each property. In addition to these method all beans contains a method named `updated`. This method is called from all setter methods and sets the `modified` property to the current date and time. When ann entity is created it additionally sets the `created` property too. The method may be seen as a simple changes tracking mechanism of when the entity was updated last. This may be done in a more lightweight way (the method is called every time a setter method is called) but this is simple and it works.

All entity beans have an overriding `equals` method. The method is mainly used along with the `entityconverter` taglibrary which converts object to string for the id of JSF select and button tags (see e.g. appendix D.1.11). The idea of the method is simple - if two objects have the same `id` - they are the same.

Besides these methods which are common for all beans `Schema` have a `addCategory` method and `Category` have a `addQuestion` method (see appendix D.2.7 and D.2.2). These methods are used to add a new category and question respectively to the tail of the lists inside the entities and assign it the correct sorting value.

7.2.1 Database implementation

The database layout resembles the class diagram shown i figure 7.4 though it has four extra tables. These tables model the relations between the entities and are:

- **Category_Question** which is created as the relation between the two classes is a bidirectional one-to-many.
- **Evaluation_Answer** which also is a bidirectional one-to-many relation.
- **Schema_Category** which also is a bidirectional one-to-many relation.
- **Review_Department** which is a many-to-many relation between the two classes

During the development the Database Management Server (DBMS for short) have been the swedish *MySQL Database*. This is a robust and rather efficient open source database and still small enough to run on even small computers. The requirements specify the system must use a Microsoft SQL Server as data backend. The shift to another DBMS is not a problem though as the Java Persistence Specification acts as a *facade* on the DBMS hiding all the "ugly" SQL stuff away and providing a nice, high-level interface. The only thing needed to change DBMS is finding an JDBC driver for the new DBMS and changing the *dialect* in the *persistence.xml* file.

7.3 Domain logic

The business logic of the APV system is implemented in enterprise session beans. The core functionality is implemented in four beans:

- **SchemaEditorBean** which has all the functionality for creating and editing an APV schema. The bean have methods for creating and editing categories in the schema - and creating and editing questions in a category.

  The beans resides in a conversational context starting the conversation as a new schema is created or a schema for editing is chosen from a list.

The conversation runs over the entire editing and ends when the schema is saved.

- **ReviewAddBean** creates a new APV review. The bean only has one business method which persists the injected review. The bean is a stateless session bean.
- **EvaluationConversationBean** handles the evaluation process. It is implemented as a conversational bean and the conversation starts when the bean creates a set of answer objects for the evaluation process - and it ends after the last page of questions have been filled out and submitted.

  As the different states in the conversation is simply different lists of questions map of question-answer pairs is outjected and send to the same page file over and over again.
- **EvaluationResultBean** calculated the results of each question in an APV review based on the currently selected category. The bean is implemented as a simple stateful bean which outjects the result to the same page over and over again.

Looking at the flowchart in figure 7.3 there are five possible paths from the start page *home*. Four of form exactly the core functionality implemented with the four core session beans.

7.4 Refactoring to list factories

The fifth path from the start page in figure 7.3 is through the *schemaList*. This page lists the schemas in the system and by selecting one of these a conversation with the *SchemaEditorBean* is started. This kind of lists are used in various parts of the application. The evaluation conversation e.g. uses a list of questions in the category the user currently is entering.

The list of categories used by the evaluation conversation is also used by the evaluation result part. The lists are exactly the same - and so is the code generating it.

Due to the Design Improvement practice before implementing the method providing the list over again the previously implemented functionality should be examined. This leads to the *Extract Method* refactoring moving the code from original methods to a new method. Due to the architecture of JEE and JBoss Seam the method is moved to a new session bean only containing the functionality of generating a list and providing the selected object. More refactorings lead to the implementation of such a "list factory" for the `Schems`, `Category`, `Question`, `Department` and `Review` objects (and with the next release a factory for the `Value` object is implemented).

Chapter 8

Test

8.1 Unit testing JEE

One of the main goals of revising the EJB specification was to simplify testing of enterprise applications. In EJB 2.1 the beans were tightly connected with the container through deployment descriptors thus making testing very complicated. Due to the tight connection applications had to be deployed into an application server cutting of the developer. There were no way to test the application directly and often applications were only tested through the client (e.g. a browser). Over the past five years or so a lot of attempts to solve the problem have emerged. The most popular is probably Cactus[1] which is a framework that allows developers to write JUnit tests and deploy them to the application server and then executed via a web interface. This way the internals of the application inside the container is exposed to JUnit and it is possible to test applications. This works but the downside is that the applications server has to be running to test and for every change in the code the application must be redeployed. Additionally Cactus is a non commercial project and the development effectively stopped at J2EE 1.3.

As beans in EJB 3.0 is simply POJO's and they do not depend on the container it is possible to test the EJBs outside the container.

[1] http://jakarta.apache.org

Entity beans are straight forward to test but often they do not contain critical functionality with different outcomes depending on the input. Simple getter and setter methods are normally not tested. In the APV system the equals method of the entity beans are tested.

Session beans on the other hand contain a lot of processing and data manipulation. It is easy to test simple method which do not interact with the database. Methods interacting with the Java Persistence is a bit trickier to test - but not difficult.

The trick is to exploit that Java Persistence uses the entity manager for al kinds of communication through the persistence functionality to the database. By creating our custom version of the entity manager and overriding the built in, full control of the communication with the ”database” is accomplished. These kinds of object which simulate some other objects functionality but giving the user full control is known as *Mock objects*.

8.2 Unit testing JBoss Seam

Testing JBoss Seam adds a little more complexity to the unit testing of session beans, as objects are injected and outjected through the framework. The solution is quite simple though as the injection resembles setter method on the internal property holding the injected object and outjection resembles getter methods. By implementing getter and setter method appropriately in the session beans, JBoss Seam element can be tested too. All session beans in the APV system is tested this way.

Chapter 9

Conclusion

9.1 Java Enterprise Edition

A more lightweight platform like *php* or *asp* could have been chosen. Most of the functionality would perhaps be easier to implement in these platforms.

There are two reasons for choosing JEE.

- With the JBoss Application Server the entire application could be packed into a single package. The package could be used on any platform with a Java 5.0 RE simply by extracting it and starting the application server. The server and the code stays within the folder it was extracted to and no system files are needed or installed. This makes the application simple and versatile to install, backup and even move to another location.
- By using JEE the application have access to the entire Java API which is big and contains almost any functionality wished for. This simplifies much of the implementation and ensures the possibility of meeting future requirements. Especially I expect it to be easier to implement more complex graphical representations of the results from a review.

I started up with J2EE version 1.4 but soon realized there is an enormous difference between that and the new JEE 5.0 platform - in favor of JEE 5.0. Unfortunately I wasted a lot of time on the older version experimenting with different tools which not were reusable in the new version. But the upgrade were worth the effort of learning the new version too as it is much more versatile and simple.

9.2 JBoss Seam

JBoss Seam is an excellent middle layer between the view/controller and the model of a JEE application. Through some simple annotations it exposes EJBs to JSF and vice versa - and additionally it brings a lot of extra functionality to the standard JSF specification. But though JBoss Seam has a lot of advantages but also some disadvantages.

The form validation functionality that moves the validation of input to the Model instead of the View as with JSF is by far one of the biggest advantages of using JBoss Seam as no duplicate code for validating the same thing in JSF is needed. But the implementation has two major drawbacks.

The `@NotNull` annotation indicating a property must be filled does not work and the form field therefore must have the `required="true"` attribute as mentioned in section 2.1.4.1. A note in the Seam reference documentation [9] states this is due to the architecture of JSF.

The second problem is the error messages from the validation. These messages is hardcoded into the entity bean and are printed as is on the web page if validation fails. As JBoss Seam really has been utilizing JSF (and thereby the internationalization possibilities) it is problematic that validation does not utilize the use of language files like JSF and JSF validation does. Though searching intensively no documentation have been found on this problem.

There is no doubt that JBoss Seam will be an important framework in the JBoss world - and thereby an important brick in the JEE puzzle, but it is still a young framework. The reference documentation has good intentions on helping new developers getting started with the framework but it is not complete. During the implementation I experienced some problems and after working on the problem for two days trying different solutions and searching for hours, I found a small note in the middle of a discussion forum stating that what I was trying is not possible due to a architectural temporary solution in the framework. Then I used most of a third day changing the implementation. Only two books have been published so far and the number of developers using JBoss Seam is still limited so searching for help is not always a simple task.

9.3 Extreme Programming

The project showed that the Planning game actually works as intended. The user stories is a simple way of communicating functionality from the customer to the development team. It is by no means exhaustive but this is where the *Whole Team* practice comes in as the customer is present in the development team. As the customer sorts the user stories according to value it is obvious which stories to start with though they might be risky from the development teams point of view.

The value sorting is obvious at the end of the project. Though only the first of the three planned iterations have been completed the system actually is usable for carrying out a review. The review will be divided on departments and it is possible to see the results in a simple graphic manner which quickly outlines problems that should be addressed.

Comparing this with carrying out a review with paper schemas where everything is done manually is a huge relief on the entire safety organization. A manual review include collecting the results of about 500 schemas with 125 questions (which is about 62.000 answers), divide them into departments and then calculating the results for each department manually would require countless working hours.

The practice of not planning ahead in the process is very difficult and as software developer this seems like the hardest thing to learn. The entire IT educational system instructs you in planning everything - and so does ordinary life. Letting go of this is very hard and you must actually think about not thinking.

The test-driven development approach where the test of some functionality is implemented prior to actually coding the functionality is a very useful approach - if the developer knows the programming language, the platform and possible frameworks. When I first started I knew Java and I knew a bit of JEE but nothing about JBoss Seam and how it interacts with JEE. This makes it very difficult to write the test as the result is unknown. After implementing several parts and writing tests for them the result became more apparent but still the test were written over several times.

The overall experience of using Extreme Programming is positive. It is a more fun way of programming software as the analyzing and planning phase is short as the focus is on producing code. In my opinion XP requires a development group of at least four people - primarily due to the *pair programming* practice so it is possible to switch. The methodology might not work for very big projects as communication and sharing information is an overall premise.

9.4 e-assessment system

The e-assessment system was not finished through this project. It was originally divided into three small releases and at the project end the first release have been finished. This release implements the core functionality of carrying out an APV process and as of now will simplify the process.

In the middle of the first iteration the project were presented to the entire safety organisation at Thermo Fischer Scientific. The response to the project was really positive and even caused two user stories being added. There were a general satisfaction with the application and the scope of the functionality as well as the web layout were positively received.

After this masters project is ended the development of the APV system at Thermo Fischer Scientific continues until the last two iterations is carried out and the final release (compared to the user stories in this thesis) is finished.

The system may be seen in function at `www.eapv.dk` using the username *dtu* and the password *summer2007*.

9.5 Personal experiences and lessons learned

Planning is everything. When I started the project I expected to implement the entire e-assessment system during the project period and planned the release iterations to last about four weeks. After having a lot of small problems with especially JBoss Seam I could not keep up with the plan. At some point the second and third releases were simply removed in the effort of finishing the core functionality of the system.

At several occasions I considered dropping JEE and implement the entire application in *php* which I have been using for several years. Using a standard framework I have developed over the years the implementation of the entire e-assessment system would probably take a month or so. This would have left at least three months for writing this report. But it would also have left me with very little or no experience. Therefore I from the beginning choose JEE as I think it is an interesting platform - and it is used widely in the professional software development community.

At the end of the project it has been a very learning experience working on a large application over several months. It gives a much broader perspective of the challenges and problems you run into in real software development compared to the small ”Mickey Mouse” problems used in the daily teaching at a university. And - as Extreme Programming said - *it was great fun*.

Appendix A

Annotations in Java 5.0

A.1 Annotation

Annotations are used to attach meta-data to some kind of target being a declaration of a constructor, field, method, package parameter, type or even the declaration of another annotation. An annotation can hold a simple type, an Object, a String, a class, an enum-type or an array of one of these types - but only one of each.

Annotations are set above the declaration like

```
@Table (name="User")
public class User {
        private int id;

        @Column (name="Id")
        @Id
        public int getId() {
                return id;
        }
        ...
}
```

Custom annotations may be implemented as an interface

```
@Target (METHOD)
@Retention (RUNTIME)
@interface Column {
        public String name;
}
```

The example shows an column annotation which only may be applied to Methods. The attribute values are kept for run time and it has only one property `name`.

Source [4].

Appendix B

Working Environment Act

The extract below is section 15a of the Consolidated Danish Working Environment Act No. 268 of March 18th 2005. It is taken from part 4 on the *General duties of the employer*.

15a.

1. The employer shall ensure the preparation of a written workplace assessment of the safety and health conditions at the workplace, taking due regard to the nature of the work, the work methods and work processes which are applied, as well as the size and organisation of the enterprise. The workplace assessment shall remain at the enterprise and be available to the management and employees at the enterprise, as well as the Danish Working Environment Authority. A workplace assessment shall be revised when there are changes in work, work methods, work processes, etc., and these changes are significant for safety and health at work. The workplace assessment shall be revised at least every three years.

2. A workplace assessment shall include an opinion on the working environment problems at the workplace, and how these are to be solved, in compliance with the principles of prevention stated in the occupational health and safety legislation. The assessment shall include the following elements:

- Identification and mapping of the working environment conditions at the enterprise.
- Description and assessment of the working environment problems at the enterprise.
- Priorities and an action plan to solve the working environment problems at the enterprise.
- Guidelines for following up the action plan.

3. The employer shall involve the Internal Safety Organisation or the employees in planning, organising, implementation and following up the workplace assessment, cf. subsections (1) and (2) above.
4. The Minister of Employment shall lay down further rules on the duties of the employer under subsections (1) to (3) above.

Appendix C

User Stories

C.1 User administration

The system should contain a user administration system. The system should contain three different user-types:

- **User** is allowed to see the results of a review for all departments.
- **Safety Group member** has the same rights as a user. Additionally he should be able to enter an evaluation and create an action-plan to the results of his owen department.
- **Administrator** has the same rights as a Safety Group member. Additionally he should be able to enter evaluations and action-plans for all departments. The administrator also administrates users, schemas and reviews. He should also be able to view statistics for all departments.

C.1.1 Create a user

Create a virtual user in the system. The user should have a username, a password, an optional email address and a optional note. Additionally a type must be set on the user and the user must be connected to a department. The email address cannot be used for username as a safety group may be created as a user.

Value: 10 • Risk: Low

C.1.2 Edit a user

Edit a users information.

Value: 11 • Risk: Low

C.1.3 Delete a user

Delete a virtual user from the system and revoke all privileges from the user. This should NOT delete the evaluations, results and action-plans the user have created in the system.

Value: 23 • Risk: Low

C.2 Department administration

The administrator should be able to administrate virtual departments in the system modeling the organizational sections of the company. Users should be connected to departments.

C.2.1 Create a department

Create a department with a name and the possibility of information on the physical department.

Value: 17 • Risk: Low

C.2.2 Edit a department

Edit the information of the department

Value: 18 • Risk: Low

C.2.3 Retire a department

Remove the department from then active part of the application. The evaluations, reviews and and action-plans should still be accessible to administrators.

Value: 28 • Risk: Low

C.3 Schema administration

A schema is a tool in the APV review. The schema contains a set of questions divided into a set of categories.

C.3.1 Create a schema

Create a schema for the APV evaluation. The schema should have a name and it should be registered who created the schema.

Value: 3 • Risk: Low

C.3.1.1 Create a category

Create a category to a schema. The category should have a name.

Value: 3 • Risk: Low

C.3.1.2 Edit a category

Edit the name of the category or the position of the category in the schema (sorting).

Value: 4 • Risk: Low

C.3.1.3 Retire a category

The category should be removed from the active part of the application. The questions, evaluations for questions and action-plans for questions should not be deleted and should be accessible to the administrator.

Value: 21 • Risk: Low

C.3.1.4 Create a question

Create a question to a category. The questions should hold the question text.

Value: 3 • Risk: Low

C.3.1.5 Edit a question

Edit the question text or the position of the question in the category.

Value: 4 • Risk: Low

C.3.1.6 Create a new revision of question

If the text of a question is radically changed a new version of the question should be added to the system. The new revision should be connected to the old version such that statistics may still be extracted on the question.

Value: 19 • Risk: Low

C.3.1.7 Retire a question

Remove a question from the active part of the application. The evaluations and action-plans connected with the question should not be deleted and should be accessible to the administrator.

Value: 20 • Risk: Low

C.3.2 Edit a schema

Editing a schema involves editing the categories and questions of the schema using the functionality described under *Create schema*.

Value: 4 • Risk: Medium

C.3.3 Print a schema

Since most of the production will fill out the APC schema on paper it should be possible to print the schema e.g. as a *pdf* file. The layout of the print should be as close to the web layout as possible.

Value: 12 • Risk: High

C.3.4 Retire a schema

Remove the schema from the active part of the application. The evaluations, reviews and action-plans based on the schema should be accessible to administrators. A schema may only be retired if it is not connected to any not-retired reviews.

Value: 22 • Risk: Low

C.4 Review administration

A review models the APV review. It connects a schema with a set of departments

C.4.1 Create a review

Create a review. A schema should be attached to the review and a list of departments participating in the review should be added to it. The review should have a name and an optional note - and it should be registered who created the review. When the review is created it should open for evaluation.

Value: 5 • Risk: Low

C.4.2 Close a review

When the review is done it should be closed so no more evaluations is added to it. It should still be possible to add action-plans and follow-ups on action-plans.

Value: 15 • Risk: Low

C.4.3 Retire a review

Remove the review from the active part of the application. Nothing should be deleted and the review should still be available to the administrator.

Value: 16 • Risk: Low

C.5 Evaluation

An evaluation is done by the individual employee and is a set of answers to the questions of the schema.

C.5.1 Enter an evaluation

The evaluation is attached to a review and should display all questions in a schema one category at a time. The user should respond to every question by choosing among a set of predefined values (e.g. *Not relevant*, *OK*, *Problem*). Additionally the user may add a remark to the each question. The system should register the user entering the evaluation and the department it is attached to. The system should respond with a unique ID which may be written on the paper version of the schema making it possible to find the paper evaluation again if doubt on the remarks.

Value: 2 • Risk: Medium

C.6 Results

The results of a review is a presentation of the percentage of each value in the total responses to a question.

C.6.1 View results of a review

The results should be presented for each question on a selected department. A graphical presentation is preferred to give a quick overview of the result. The number of evaluations the result is based upon should be displayed. The remarks made through the evaluation should also be displayed.

Value: 1 • Risk: High

C.6.2 View results of a review as Pareto Chart

The results should be presented as a *Pareto Chart*. These charts are often used in the company group.

Value: 27 • Risk: High

C.7 Action-plan administration

If the result of a review shows a problem on a specific question an action-plan for solving the problem must be created.

C.7.1 Create an action-plan

Create an action-plan and attach it to a specific question, the review and the department. The action-plan should contain a name of the responsible owner of the plan, a deadline for solving it and a description of the problem and solution. The system should register the user that created the plan.

Value: 6 • Risk: Low

C.7.2 Create a follow-up action-plan

If an action-plan cannot be solved within the deadline a follow-up on the plan must be made. This resembles the action-plan as it sets a new deadline, a responsible owner and a description of the reasons that the original action-plan was not solved within the deadline. The follow-up action-plan must be connected to the original plan.

Value: 8 • Risk: Medium

C.7.3 Close an action-plan

When the problem of the action-plan is solved, the action-plan may be closed. It is not removed from the system but stays available as long as the review is not retired.

Value: 7 • Risk: Low

C.7.4 List action-plans

It should be possible to get a list of all action-plans (and follow-up action-plans) and the status of them. The administrator should be able to chose either a specific department or all departments whereas the the other users may only list action-plans from their own department.

Value: 9 • Risk: Low

C.8 Statistics

Statistics is an important tool when examining the health and safety of the organization. Rather than implementing some static tools for calculating and displaying statistics, the extraction of comma-separated files which may be used in statistics systems or spreadsheets is required.

C.8.1 Extract results of review as comma-separated file

It should be possible to extract the result of a review as a list with all questions and the number and percentage of each value among the answers. Administrators should be able to extract either a specific department or a set of departments.

Value: 24 • Risk: Medium

C.8.2 Extract statistics of action-plans as comma-separated file

The administrator should be able to extract statistics on action-plans. The interesting statistics are how fast the safety group in a department solves the action-plan and how many times it is postponed (that is a follow-up plan is created).

Value: 25 • Risk: Medium

C.8.3 Extract statistics of schema as comma-separated file

It should be possible to extract the results of a schema from several reviews to follow the development over several years. The statistics should be based on either a one or a set of departments and only the administrator should have access to them.

Value: 26 • Risk: Medium

C.9 Accessibility

The accessibility to the system is based on a login and a password (see User administration)

C.9.1 Log in

A user should be authenticated and logged in. The user then have access to the functionality on the department/departments he is connected to. The access is based on his user type.

Value: 13 • Risk: Medium

C.9.2 Log out

When the user logs out the session should be ended and the user should not have access to anything.

Value: 14 • Risk: Low

Appendix D

Source Code

This appendix contains the entire source code for the APV system release 1. It is divided into the web pages forming the view, the entity beans modeling the domain data and the session beans modeling the domain business logic. The appendix ends with some of the more interesting configuration files.

D.1 Web pages

The web pages is written in *xhtml* format and follows the *W3C standard*. The pages uses standard JSP tags (`<f:` and `<h:`) and JBoss Seam tags (`<s:`).

D.1.1 template.xhtml

This is the template file for the web pages generated. It sets up the layout of the page through simple html tags an using the cascading style sheet defined for the project. The template includes three *ui* JSF elements on the page: the content of the page, the sidebar.xhtml and the content of the lower right side.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="da" lang="da"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:s="http://jboss.com/products/seam/taglib">
   <head>
      <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
      <meta http-equiv="content-script-type" content="text/javascript" />
      <meta http-equiv="content-style-type" content="text/css" />
      <link rel="stylesheet" href="style/layout.css" type="text/css" />
      <title>eAPV.dk</title>
   </head>
   <body>
      <div id="sitecontainer">
         <div class="divider"></div>
         <div id="topbarcontainer">
            <div id="topbarleft">
               <ui:insert name="pagename"/>
            </div>
            <div id="topbarright">
               <!-- The top bar showing logged in user and date -->
               <img src="images/key.png" alt="Logget ind" id="topbarkey"
               border="0"/> Martin | <h:outputText value=" #{dateShort} " />
               [ Log ud ]
            </div>
            <div id="topbarcenter">&nbsp;</div>
         </div>
         <div class="divider"></div>

         <div id="leftcontentcontainer">
            <!-- Insert the content on the page -->
            <ui:insert name="content"/>
            <br />
            <hr />
            <!-- Print error faces messages - e.g. validation errors -->
            <h:messages/>
         </div>

         <div id="rightcontentcontainer">
            <!-- Include sidebar xhtml which generates the menu -->
            <ui:include src="sidebar.xhtml" />
            <hr/>
            <br/><br/>
            <!-- Insert content for the lower right frame -->
            <ui:insert name="sidecontent"/>
         </div>

         <div id="bottombar">
            eAPV.dk | 2007 | Thermo Fischer Scientific , Roskilde
         </div>
      </div>
   </body>
</html>
```

D.1.2 home.xhtml

This is the frontpage of the application after user has logged in.

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<ui:composition xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:s="http://jboss.com/products/seam/taglib"
template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
  <h:outputText value="#{messages.PageHeadFrontpage}"/>
</ui:define>

<!-- Page content block -->
<ui:define name="content">
  <h:outputText value="#{messages.Welcome}"/>
</ui:define>

<!-- Page sidebar block -->
<ui:define name="sidebar"></ui:define>

</ui:composition>
```

D.1.3 sidebar.xhtml

This is the menu file for the right side of the template. The drop-down box for department selection is not implemented correctly as it is a static solution in release 1. It should be generated by the departments list factory - and the selected department should be saved as a session value for the user.

```
<!DOCTYPE div PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<div xmlns="http://www.w3.org/1999/xhtml"
    xmlns:c="http://java.sun.com/jstl/core"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:s="http://jboss.com/products/seam/taglib">

    <!-- Select the department currently worked on
    Temporary static solution for display only
    Replaced by a dynamic list generated by the
    departments list factory and the selected
    is saved as a session variable -->
    <select     style="margin-left: 3px; width: 140px;"
        onchange="document.getElementById('menu').style.display='block';">
        <option>Afdeling 10</option>
        <option>Afdeling 11</option>
```

```
      <option>Afdeling 12</option>
      <option>Afdeling 13</option>
      <option>Afdeling 14</option>
   </select>
   <br />
   <br />
   <hr />
   <!-- Generate menu depending on the selected department -->
   <span id="menu">
      <s:link id="MenuFrontpage" view="/home.xhtml"
         value="#{messages.PageHeadFrontpage}" styleClass="menuitem"
         propagation="none"/>
      <s:link id="MenuSchemaAdd" view="/schemaAdd.xhtml"
         value="#{messages.addSchema}" styleClass="menuitem"
         propagation="none"/>
      <s:link id="MenuSchemaEdit" view="/schemaList.xhtml"
         value="#{messages.editSchema}" styleClass="menuitem"
         propagation="none"/>
      <s:link id="MenuReviewAdd" view="/reviewAdd.xhtml"
         value="#{messages.addReview}" styleClass="menuitem"
         propagation="none"/>
         <s:link id="MenuStartEvaluation"
            action="#{evaluationConversation.startEvaluation}"
            value="#{messages.startEvaluation}" styleClass="menuitem"
            propagation="none"/>
   </span>
   <hr />
</div>
```

D.1.4 schemaAdd.xhtml

This page displays a form for creating a new schema in the system. It contains a temporary field for the editor id. In release 2 this will be replaced by the logged in user.

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:s="http://jboss.com/products/seam/taglib"
   template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
   <h:outputText value="#{messages.PageHeadSchemaAdd}"/>
</ui:define>

<!-- Page content block -->
<ui:define name="content">
   <h:form id="schemaAdd">
      <fieldset>
         <s:validateAll>
            <h:outputLabel for="name">
               <h:outputText value="#{messages.schemaName}" />
            </h:outputLabel><br/>
            <h:inputText id="name" value="#{schema.name}"  required="true"/>
            <br/><br/>

            <!-- A temporary field for adding the editor of the
                 schema. Replaced by logged in user -->
            <h:outputLabel for="editor">
               <h:outputText value="#{messages.editor}" />
            </h:outputLabel><br/>
            <h:inputText id="editor" value="#{schema.editor}" required="true"/>
            <br/><br/>

            <h:commandButton id="submitAddSchema" value="#{messages.create}"
               action="#{schemaEditor.schemaAdd}"/>
         </s:validateAll>
      </fieldset>
   </h:form>
</ui:define>

<!-- Page sidebar block -->
   <ui:define name="sidebar"></ui:define>
</ui:composition>
```

D.1.5 schemaList.xhtml

This page displays a list of active schemas in the application along with an icon for editing the schema.

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:s="http://jboss.com/products/seam/taglib"
    template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
   <h:outputText value="#{messages.PageHeadSchemaList}"/>
</ui:define>

<!-- Page content block -->
<ui:define name="content">
   <h:form>
      <h:dataTable var="scm" value="#{schemaList}"
          rendered="#{schemaList.rowCount>0}">
         <h:column>
            <h:outputText value="#{scm.name}"/>
         </h:column>

         <h:column>
         <s:link id="editSchema" value=""
            action="#{schemaEditor.schemaEdit}">
               <h:graphicImage value="/images/pencil.png"/>
            </s:link>
         </h:column>
      </h:dataTable>
   </h:form>
</ui:define>
```

```
<!-- Page sidebar block -->
<ui:define name="sidebar"></ui:define>

</ui:composition>
```

D.1.6 schemaEditor.xhtml

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:s="http://jboss.com/products/seam/taglib"
   template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
   <h:outputText value="#{messages.PageHeadSchemaEditor}"/>
</ui:define>

<!-- Page content block -->
<ui:define name="content">

<h2>
   <h:outputText value="#{selectedCategory.name}"
      rendered="#{selectedCategory != null}"/>
</h2>

<!-- Generate list of questions in selected category -->
<h:form>
<h:dataTable var="qst" value="#{questionList}"
rendered="#{questionList.rowCount>0}"  styleClass="evaluation">
   <h:column>
      <h:outputText value="#{qst.id}"/>
   </h:column>
   <h:column>
      <h:outputText value="#{qst.text}"/>
   </h:column>

<!-- Link for moving the question up in the sorting order -->
   <h:column>
      <h:commandLink value="" action="#{questions.questionMoveUp}" >
         <h:graphicImage value="/images/arrow_up.png"/>
      </h:commandLink>
   </h:column>

<!-- Link for editing the question -->
   <h:column>
      <s:link id="editQuestion" value=""
         action="#{schemaEditor.questionEdit}">
         <h:graphicImage value="/images/pencil.png"/>
      </s:link>
   </h:column>
</h:dataTable>
</h:form>
<s:link id="addQuestion" action="#{schemaEditor.questionAdd}"
      value="#{messages.AddQuestion}" rendered="#{selectedCategory != null}"/>
<br/><br/>
<s:link id="schemaSave" action="#{schemaEditor.schemaSave}"
```

```
    value="#{messages.SaveSchema}"/>
</ui:define>


<!-- Page sidebar block -->
<ui:define name="sidebar">
<!-- Generate list for categories for right sidebar -->
<h:form>
<h:dataTable var="cat" value="#{categoryList}"
    rendered="#{categoryList.rowCount>0}">
    <h:column>
        <h:commandLink value="#{cat.name}" action="#{questions.findQuestions}"/>
    </h:column>

<!-- Link for moving the category up in the sorting order -->
    <h:column>
        <h:commandLink value="" action="#{categories.categoryMoveUp}">
            <h:graphicImage value="/images/arrow_up.png"/>
        </h:commandLink>
    </h:column>

<!-- Link for editing the category -->
    <h:column>
    <s:link id="editCategory" value=""
        action="#{schemaEditor.categoryEdit}">
            <h:graphicImage value="/images/pencil.png"/>
        </s:link>
    </h:column>
</h:dataTable>
</h:form>
<br/>
<s:link id="addCategory" action="#{schemaEditor.categoryAdd}"
    value="#{messages.AddCategory}"/>
</ui:define>

</ui:composition>
```

D.1.7 categoryAdd.xhtml

This page displays a form with one field for creating a new category in a schema.

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:s="http://jboss.com/products/seam/taglib"
    template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
    <h:outputText value="#{messages.PageHeadAddCategory}"/>
</ui:define>

<!-- Page content block -->
<ui:define name="content">
```

```
   <h:form id="addCategory">
      <fieldset>
         <h:outputLabel for="name">
            <h:outputText value="#{messages.schemaName}" />
         </h:outputLabel><br/>
         <h:inputText id="name" value="#{category.name}" required="true"/>
         <br/><br/>

         <h:commandButton id="submitAddCategory" value="#{messages.create}"
         action="#{schemaEditor.categorySave}"/>
      </fieldset>
   </h:form>
</ui:define>

<!-- Page sidebar block -->
<ui:define name="sidebar"></ui:define>

</ui:composition>
```

D.1.8 categoryEdit.xhtml

This page displays a form with one field with the name of the selected category. The name of the category can be changed and then saved.

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:s="http://jboss.com/products/seam/taglib"
   template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
   <h:outputText value="#{messages.PageHeadCategoryEdit}"/>
</ui:define>

<!-- Page content block -->
<ui:define name="content">

   <h:form id="editCategory">
      <fieldset>
         <s:validateAll>
            <h:outputLabel for="name">
               <h:outputText value="#{messages.categoryName}" />
            </h:outputLabel>
            <br/>

            <h:inputText id="name" value="#{category.name}" required="true"/>
            <br/><br/>
            <h:commandButton id="submitEditCategory" value="#{messages.save}"
               action="#{schemaEditor.categorySave}"/>
         </s:validateAll>
      </fieldset>
   </h:form>
</ui:define>
```

```
<!-- Page sidebar block -->
<ui:define name="sidebar"></ui:define>

</ui:composition>
```

D.1.9 questionAdd.xhtml

This page displays a form with one field for adding a question.

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:s="http://jboss.com/products/seam/taglib"
  template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
  <h:outputText value="#{messages.PageHeadCateoryEdit}"/>
</ui:define>

<!-- Page content block -->
<ui:define name="content">
  <h:form id="editQuestion">
    <fieldset>
      <s:validateAll>
        <h:outputLabel for="name">
          <h:outputText value="#{messages.questionText}" />
        </h:outputLabel>
        <br/>

        <h:inputText id="name" value="#{question.text}" required="true"/>
        <br/><br/>

        <h:commandButton id="submitEditQuestion" value="#{messages.save}"
          action="#{schemaEditor.questionSave}"/>
      </s:validateAll>
    </fieldset>
  </h:form>
</ui:define>

<!-- Page sidebar block -->
<ui:define name="sidebar"></ui:define>

</ui:composition>
```

D.1.10 questionEdit.xhtml

This page displays a form with one field with the selected question. The question text can be changed and then saved.

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:s="http://jboss.com/products/seam/taglib"
    template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
   <h:outputText value="#{messages.PageHeadCateoryEdit}"/>
</ui:define>

<!-- Page content block -->
<ui:define name="content">
   <h:form id="editQuestion">
      <fieldset>
         <s:validateAll>
            <h:outputLabel for="name">
               <h:outputText value="#{messages.questionText}" />
            </h:outputLabel>
            <br/>

            <h:inputText id="name" value="#{question.text}" required="true"/>
            <br/><br/>

            <h:commandButton id="submitEditQuestion" value="#{messages.save}"
               action="#{schemaEditor.questionSave}"/>
         </s:validateAll>
      </fieldset>
   </h:form>
</ui:define>

<!-- Page sidebar block -->
<ui:define name="sidebar"></ui:define>

</ui:composition>
```

D.1.11 reviewAdd.xhtml

This page displays a form for creating a view. Besides a name and a temporary editor field it contains two drop-down boxes where one schema and one or more departments can be chosen (*selectOneMenu* and*selectManyListBox*).

The two drop-down boxes uses a taglibrary named *entityconverter* to convert from `Schema` objects and `Department` objects respectively to a `String` which is used for the html `select` element as the id. When the form is submitted the *entityconverter* uses the string id to return the correct object. It uses the object `equals()` method for this.

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:s="http://jboss.com/products/seam/taglib"
   xmlns:ec="http://jboss.com/products/seam/entityconverter/taglib"
   template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
   <h:outputText value="#{messages.PageHeadReviewAdd}"/>
</ui:define>

<!-- Page content block -->
<ui:define name="content">
<h:form id="reviewAdd">
<fieldset>
   <s:validateAll>
      <h:outputLabel for="name">
         <h:outputText value="#{messages.reviewName}" />
      </h:outputLabel>
      <br/>
      <h:inputText id="name" value="#{review.name}"/>
      <br/><br/>

      <!-- Temporary field for editor id. Replaced by
      logged in user in release 2 -->
      <h:outputLabel for="editor">
         <h:outputText value="#{messages.editor}" />
      </h:outputLabel><br/>
      <h:inputText id="editor" value="#{review.editor}"/>
      <br/><br/>

      <h:outputLabel for="schema">
         <h:outputText value="#{messages.schemaName}" />
      </h:outputLabel>
      <br/>
      <h:selectOneMenu id="schema" value="#{review.schema}">
         <s:selectItems value="#{schemaModel}" var="scm"
            label="#{scm.name}"/>

         <!-- Entity converter used to convert Schema to text for
         the select elemtens value tag and vice versa -->
         <ec:convertEntity entityClass="dk.eapv.ejb.domain.Schema"/>
      </h:selectOneMenu>
      <br/><br/>

      <h:outputLabel for="department">
         <h:outputText value="#{messages.department}" />
      </h:outputLabel>
      <br/>
      <h:selectManyListbox id="department" value="#{review.departments}">
         <s:selectItems value="#{departmentList}" var="dpt"
            label="#{dpt.name}"/>

         <!-- Entity converter used to convert Department to text for
                the select elemtens value tag and vice versa -->
         <ec:convertEntity entityClass="dk.eapv.ejb.domain.Department"/>
      </h:selectManyListbox>
      <br/><br/>

      <h:commandButton id="submitReviewAdd" value="#{messages.create}"
```

```
         action="#{reviewAdd.reviewAdd}" />
   </s:validateAll>
</fieldset>
</h:form>
</ui:define>

<!-- Page sidebar block -->
<ui:define name="sidebar"></ui:define>

</ui:composition>
```

D.1.12 evaluationConversation.xhtml

This page is the actual questionnaire with questions answered by the employees. The possible answers for a question is determined by the number of `Value` objects in the database.

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:s="http://jboss.com/products/seam/taglib"
    xmlns:ec="http://jboss.com/products/seam/entityconverter/taglib"
    template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
    <h:outputText value="#{messages.PageHeadEvaluation}"/>
</ui:define>

<!-- Page content block -->
<ui:define name="content">
    <!-- The ID of the finished evaluaion. Is only rendered if the
           evaluationId is set -->
    <h:outputText value="#{messages.evaluationEnded}.
        #{messages.evaliationIdIs} #{evaluationId}"
        rendered="#{evaluationId != null}"/>

    <!-- Current category name -->
    <h2>
        <h:outputText value="#{evaluationCategory.name}"
        rendered="#{evaluationId == null}"/>
    </h2>

    <!-- Evaluation form. Only rendered if evaluationId is not set -->
    <h:form id="evaluationForm" rendered="#{evaluationId == null}">
        <fieldset>
            <s:validateAll>
                <h:dataTable value="#{evaluationAnswerList}" var="ans"
                    styleClass="evaluation"
                    columnClasses="question,valuecontainer,note">

                    <h:column>
                        <h:outputText value="#{ans.question.text}" />
                    </h:column>
                <!-- Radio button for ranking value of the question
                      The number of choices is determined by the
                      number of values in the database -->
                <h:column>
                    <h:selectOneRadio value="#{ans.value}"
                        required="true" layout="linedirection"
                        styleClass="value">

                        <s:selectItems value="#{valueList}" var="val" label=""/>
                        <!-- Use entityconverter to convert from Value to String
                              and vice versa on submit -->
                        <ec:convertEntity entityClass="dk.eapv.ejb.domain.Value"/>
                    </h:selectOneRadio>
                </h:column>

                <h:column>
                    <h:inputText value="#{ans.remark}" styleClass="remark"/>
                </h:column>
            </h:dataTable>

            <h:commandButton id="submitEvaluationPrevious"
                value="#{messages.previous}"
                action="#{evaluationConversation.previousCategory}">
                <!-- Propagate the conversation when clicked -->
                <s:conversationPropagation type="join"/>
            </h:commandButton>

            <h:commandButton id="submitEvaluationNext"
                value="#{messages.next}"
                action="#{evaluationConversation.nextCategory}">
                <!-- Propagate the conversation when clicked -->
                <s:conversationPropagation type="join"/>
            </h:commandButton>
        </s:validateAll>
    </fieldset>
  </h:form>
</ui:define>

<!-- Page sidebar block -->
<ui:define name="sidebar"></ui:define>

</ui:composition>
```

D.1.13 evaluationResult.xhtml

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
         xmlns:ui="http://java.sun.com/jsf/facelets"
         xmlns:h="http://java.sun.com/jsf/html"
         xmlns:f="http://java.sun.com/jsf/core"
         xmlns:s="http://jboss.com/products/seam/taglib"
         template="template.xhtml">

<!-- Page header block -->
<ui:define name="pagename">
   <h:outputText value="#{messages.PageHeadEvaluationResult}"/>
</ui:define>

<!-- Page content block -->
```

```
<ui:define name="content">
   <h:dataTable value="#{resultQuestionList}" var="qst"  styleclass="evaluation">
      <h:column>
         <h:outputText value="#{qst.text}" />
      </h:column>
   </h:dataTable>
</ui:define>

<!-- Page sidebar block -->
<ui:define name="sidebar"></ui:define>

</ui:composition>
```

D.2 Entity Beans

The entity beans are standard Java beans with additional annotations for the persistence mechanism.

D.2.1 Answer.java

The Answer class hold a contrete answer to a specific question in an evaluation of a review.

```
package dk.eapv.ejb.domain;

import java.util.Date;

import javax.persistence.*;

import org.hibernate.validator.NotNull;
import org.jboss.seam.annotations.Name;

@Entity
@Name ("answer")
@Table (name="Answer")
public class Answer {
   // Id of the answer
   private int id;

   // The rank value of the answer
   private Value value;

   // Remarks to the answer
   private String remark;

   // The question the answer is related to
   private Question question;

   // Date of creation
   private Date created;
```

```
// Date of last modification
private Date modified;

@Id @GeneratedValue
@Column (name="AnswerID")
public int getId() {
   return id;
}

public void setId(int id) {
   this.id = id;
   this.updated();
}

@ManyToOne
@JoinColumn (name="AnswerValue")
public Value getValue() {
   return value;
}

public void setValue(Value value) {
   this.value = value;
   this.updated();
}

@Column (name="AnswerRemark")
public String getRemark() {
   return remark;
}

public void setRemark(String remark) {
   this.remark = remark;
   this.updated();
}

@ManyToOne
@JoinColumn (name="AnswerQuestion")
public Question getQuestion() {
   return question;
}

public void setQuestion(Question question) {
   this.question = question;
   this.updated();
}

@Column (name="AnswerCreated")
@Temporal (TemporalType.TIMESTAMP)
public Date getCreated() {
   return created;
}

public void setCreated(Date created) {
   this.created = created;
}

@Column (name="AnswerModified")
@Temporal (TemporalType.TIMESTAMP)
public Date getModified() {
   return modified;
}

public void setModified(Date modified) {
   this.modified = modified;
}
```

```
   // Updates the modification date on an object
   private void updated() {
      if (this.id == 0)
         this.created = new Date();
      this.modified = new Date();
   }

   // Overriding equals method check on id only
   public boolean equals(Object o) {
      if (o == null)
         return false;
      else {
         Answer otherAnswer = (Answer) o;
         return this.getId() == otherAnswer.getId();
      }
   }
}
```

D.2.2 Category.java

The questionnaire schema contains a set of categories each with a name and a list of questions in the category.

```
package dk.eapv.ejb.domain;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.List;

import javax.persistence.*;

@Entity
@Name ("category")
@Table (name="Category")
public class Category {
   // If of the category
   private int id;

   // The name of the category
   private String name;

   // Placement in the sorting rank
   private int sorting;

   // List of questions in the category
   private List<Question> questions = new ArrayList<Question>();

   // Date of creation
   private Date created;

   // Date of last modification
   private Date modified;

   @Id @GeneratedValue
   @Column (name="CategoryId")
   public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @Column (name="CategoryName")
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
        this.updated();
    }

    @Column (name="CategorySorting")
        public int getSorting() {
            return sorting;
        }

    public void setSorting(int sorting) {
        this.sorting = sorting;
        this.updated();
    }

    @OneToMany
    @OrderBy ("sorting ASC")
    public List<Question> getQuestions() {
        return questions;
    }

    public void setQuestions(List<Question> questions) {
        this.questions = questions;
        this.updated();
    }

    @Column (name="CategoryCreated")
    @Temporal (TemporalType.TIMESTAMP)
    public Date getCreated() {
        return created;
    }

    public void setCreated(Date created) {
        this.created = created;
    }

    @Column (name="CategoryModified")
    @Temporal (TemporalType.TIMESTAMP)
    public Date getModified() {
        return modified;
    }

    public void setModified(Date modified) {
        this.modified = modified;
    }

    // Updates the modification date on an object
    private void updated() {
        if (this.id == 0)
            this.created = new Date();
        this.modified = new Date();
    }
    // Overriding equals method check on id only
    public boolean equals(Object o) {
        if (o == null)
            return false;
        else {
            Category otherCategory = (Category) o;
            return this.getId() == otherCategory.getId();
        }
    }

    // Simple function for adding a question to the
    // the end of the list of questions with the
    // correct sorting rank value
    public void addQuestion(Question question) {
        question.setSorting(questions.size()+1);
        this.questions.add(question);
    }
}
```

D.2.3 Department.java

Companies are often divided into organizational units which may be modeled by departments. This gives the possibility of dividing results of a review on departments for better overview.

```
package dk.eapv.ejb.domain;

import javax.persistence.*;

import java.util.*;
import static org.jboss.seam.ScopeType.EVENT;

import org.hibernate.validator.Length;
import org.hibernate.validator.NotNull;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.ui.tag.MessageTag;

@Entity
@Name ("department")
@Table (name="Department")

public class Department {
    // Id of the department
    private int id;

    // The name of the department
    private String name;

    // List of users in the department
    private List<User> users;

    // Date of creation
    private Date created;

    // Date of last modification
    private Date modified;
```

```
@Id @GeneratedValue
@Column (name="DepartmentId")
public int getId() {
   return id;
}

public void setId(int id) {
   this.id = id;
}

@Column (name="DepartmentName")
@NotNull (message="Navn skal udfyldes")
public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
   this.updated();
}

@ManyToMany
public List<User> getUsers() {
   return users;
}

public void setUsers(List<User> users) {
   this.users = users;
   this.updated();
}

@Column (name="DepartmentCreated")
@Temporal (TemporalType.TIMESTAMP)
public Date getCreated() {
   return created;
}

public void setCreated(Date created) {
   this.created = created;
}

@Column (name="DepartmentModified")
@Temporal (TemporalType.TIMESTAMP)
public Date getModified() {
   return modified;
}

public void setModified(Date modified) {
   this.modified = modified;
}

// Updates the modification date on an object
private void updated() {
   if (this.id == 0)
      this.created = new Date();
   this.modified = new Date();
}

// Overriding equals method check on id only
public boolean equals(Object o) {
   if (o == null)
      return false;
   else {
      Department otherDepartment = (Department) o;
```

```
            return this.getId() == otherDepartment.getId();
        }
    }
}
```

D.2.4 Evaluation.java

An evaluation is a set of answers given by one employee on the questions in a schema. The evaluation is related to a department and a review.

```
package dk.eapv.ejb.domain;

import java.util.Date;
import java.util.List;

import javax.persistence.*;

import org.jboss.seam.annotations.Name;

@Entity
@Name ("evaluation")
@Table (name="Evaluation")
public class Evaluation {
    // Id of the evaluation
    private int id;

    // The review the evaluation is related to
    private Review review;

    // The department the evaluation is related to
    private Department department;

    // List of answers to questions from the user
    private List<Answer> answers;

    // Date of creation
    private Date created;

    // Date of last modification
    private Date modified;

    @Id @GeneratedValue
    @Column (name="EvaluationId")
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @ManyToOne
    public Review getReview() {
        return review;
    }

    public void setReview(Review review) {
        this.review = review;
        this.updated();
    }

    @ManyToOne
    public Department getDepartment() {
        return department;
    }

    public void setDepartment(Department department) {
        this.department = department;
        this.updated();
    }

    @OneToMany (cascade={CascadeType.ALL})
    public List<Answer> getAnswers() {
        return answers;
    }

    public void setAnswers(List<Answer> answers) {
        this.answers = answers;
        this.updated();
    }

    @Column (name="EvaluationCreated")
    @Temporal (TemporalType.TIMESTAMP)
    public Date getCreated() {
        return created;
    }

    public void setCreated(Date created) {
        this.created = created;
    }

    @Column (name="EvaluationModified")
    @Temporal (TemporalType.TIMESTAMP)
    public Date getModified() {
        return modified;
    }

    public void setModified(Date modified) {
        this.modified = modified;
    }

    // Updates the modification date on an object
    private void updated() {
        if (this.id == 0)
            this.created = new Date();
        this.modified = new Date();
    }

    // Overriding equals method check on id only
    public boolean equals(Object o) {
        if (o == null)
            return false;
        else {
            Evaluation otherEvaluation = (Evaluation) o;
            return this.getId() == otherEvaluation.getId();
        }
    }
}
```

D.2.5 Question.java

A question is a simple question in the questionnaire.

```
package dk.eapv.ejb.domain;

import java.util.Date;

import javax.persistence.*;

import org.hibernate.validator.Length;
import org.hibernate.validator.NotNull;

@Entity
@Name ("question")
@Table (name="Question")
public class Question {
   // Id of the question
   private int id;

   // The question text
   private String text;

   // The version of the question.
   // New version can be created if the text
   // changes and the history is essential
   private int version;

   // The sorting rank of the question
   // within the category
   private int sorting;

   // Date of creation
   private Date created;

   // Date of last modification
   private Date modified;

   @Id  @GeneratedValue
   @Column (name="QuestionId")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   @Column (name="QuestionText")
   @NotNull @Length(min=5, max=255)
   public String getText() {
      return text;
   }

   public void setText(String text) {
      this.text = text;
      this.updated();
   }

   @Column (name="QuestionSorting")
   @NotNull
   public int getSorting() {
        return sorting;
    }

    public void setSorting(int sorting) {
        this.sorting = sorting;
        this.updated();
    }

    @Column (name="QuestionVersion")
    @NotNull
    public int getVersion() {
        return version;
    }

    public void setVersion(int version) {
        this.version = version;
        this.updated();
    }

    @Column (name="QuestionCreated")
    @Temporal (TemporalType.TIMESTAMP)
    public Date getCreated() {
        return created;
    }

    public void setCreated(Date created) {
        this.created = created;
    }

    @Column (name="QuestionModified")
    @Temporal (TemporalType.TIMESTAMP)
    public Date getModified() {
        return modified;
    }

    public void setModified(Date modified) {
        this.modified = modified;
    }

    // Updates the modification date on an object
    private void updated() {
        if (this.id == 0)
            this.created = new Date();
        this.modified = new Date();
    }

    // Overriding equals method check on id only
    public boolean equals(Object o) {
        if (o == null)
            return false;
        else {
            Question otherQuestion = (Question) o;
            return this.getId() == otherQuestion.getId();
        }
    }
}
```

D.2.6 Review.java

A revision of the APV is modeled as a review. The review contains a specific schema with questions used in the review.

```
package dk.eapv.ejb.domain;

import javax.persistence.*;

import java.util.*;

import static org.jboss.seam.ScopeType.EVENT;

import org.hibernate.validator.Length;
import org.hibernate.validator.NotNull;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.ui.tag.MessageTag;

@Entity
@Name ("review")
@Table (name="Review")

public class Review {
   // Id of the review
   private int id;

   // Name of the review
   private String name;

   // Id of the user who created the review
   private int editor;

   // A note used for description or guidelines
   private String note;

   // The schema used in the review
   private Schema schema;

   // A list of departments carring out the review
   private List<Department> departments;

   // Date of creation
   private Date created;

   // Date of last modification
   private Date modified;

   @Id @GeneratedValue
   @Column (name="ReviewId")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   @Column (name="ReviewName")
   public String getName() {
      return name;
}

public void setName(String name) {
   this.name = name;
   this.updated();
}

@Column (name="ReviewEditor")
@NotNull (message="Editor skal udfyldes")
public int getEditor() {
   return editor;
}

public void setEditor(int editor) {
   this.editor = editor;
   this.updated();
}

@Column (name="ReviewNote")
public String getNote() {
   return note;
}

public void setNote(String note) {
   this.note = note;
   this.updated();
}

@ManyToOne
@JoinColumn (name="ReviewSchema")
public Schema getSchema() {
   return schema;
}

public void setSchema(Schema schema) {
   this.schema = schema;
   this.updated();
}

@ManyToMany
public List<Department> getDepartments() {
   return departments;
}

public void setDepartments(List<Department> departments) {
   this.departments = departments;
   this.updated();
}

@Column (name="ReviewCreated")
@Temporal (TemporalType.TIMESTAMP)
public Date getCreated() {
   return created;
}

public void setCreated(Date created) {
   this.created = created;
}

@Column (name="ReviewModified")
@Temporal (TemporalType.TIMESTAMP)
public Date getModified() {
   return modified;
}
```

```
   public void setModified(Date modified) {
      this.modified = modified;
   }

   // Updates the modification date on an object
   private void updated() {
      if (this.id == 0)
         this.created = new Date();
      this.modified = new Date();
   }

   // Overriding equals method check on id only
   public boolean equals(Object o) {
      if (o == null)
         return false;
      else {
         Review otherReview = (Review) o;
         return this.getId() == otherReview.getId();
      }
   }
}
```

D.2.7 Schema.java

The schema models the questionnaire schema with questions divided into categories.

```
package dk.eapv.ejb.domain;

import javax.persistence.*;
import java.util.*;
import static org.jboss.seam.ScopeType.EVENT;

import org.hibernate.validator.Length;
import org.hibernate.validator.NotNull;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.ui.tag.MessageTag;

@Entity
@Name ("schema")

// As schema is a keyword in database terminology
// the table is named Scheme
@Table (name="Scheme")

public class Schema {
   // Id of the schema
   private int id;

   // The schema name
   private String name;

   // The id of the editor
   private String editor;

   // Used for a description or a guideline
   private String description;
```

```
// List of categories in the schema
private List<Category> categories = new ArrayList<Category>();

// Date of creation
private Date created;

// Date of last modification
private Date modified;

@Id @GeneratedValue
@Column (name="SchemeId")
public int getId() {
   return this.id;
}

public void setId(int id) {
   this.id = id;
}

@Column (name="SchemeName")
@NotNull (message="Skemanavn skal udfyldes")
@Length(min=3, max=255, message="Skemanavn skal være 3 - 255 tegn")
public String getName() {
   return this.name;
}

public void setName(String name) {
   this.name = name;
   this.updated();
}

@Column (name="SchemeEditor")
@NotNull (message="Brugernavnet skal udfyldes")
public String getEditor() {
   return editor;
}

public void setEditor(String editor) {
   this.editor = editor;
   this.updated();
}

@Column (name="SchemeDescription")
public String getDescription() {
   return description;
}

public void setDescription(String description) {
   this.description = description;
   this.updated();
}

@OneToMany (cascade={CascadeType.ALL})
@OrderBy ("sorting ASC")
public List<Category> getCategories() {
   return categories;
}

public void setCategories(List<Category> categories) {
   this.categories = categories;
   this.updated();
}

@Column (name="SchemeCreated")
```

```
    @Temporal (TemporalType.TIMESTAMP)
    public Date getCreated() {
        return created;
    }

    public void setCreated(Date created) {
        this.created = created;
    }

    @Column (name="SchemeModified")
    @Temporal (TemporalType.TIMESTAMP)
    public Date getModified() {
        return modified;
    }

    public void setModified(Date modified) {
        this.modified = modified;
    }

    // Updates the modification date on an object
    private void updated() {
        if (this.id == 0)
            this.created = new Date();
        this.modified = new Date();
    }

    // Overriding equals method check on id only
    public boolean equals(Object o) {
        if (o == null)
            return false;
        else {
            Schema otherSchema = (Schema) o;
            return this.getId() == otherSchema.getId();
        }
    }

    // Simple method for adding a category to the
    // end of the category list. The category will
    // get the correct sorting rank
    public void addCategory(Category category) {
        category.setSorting(categories.size()+1);
        this.categories.add(category);
    }
}
```

D.2.8 Value.java

A question may be answered with a rank value and a remark. The rank value is modeled by a value class to simplify statistics on the results.

```
package dk.eapv.ejb.domain;

import javax.persistence.*;

import org.jboss.seam.annotations.Name;
import java.util.Date;

@Entity
@Name("value")
```

```
@Table (name="Value")

public class Value {
   // Id of the value
   private int id;

   // Value text
   private String text;

   // Color of the result bar indicating
   // the percent of answers with this value
   private String color;

   // Sorting rank in evaluation and result
   private int sorting;

   // Date of creation
   private Date created;

   // Date of modification
   private Date modified;

   @Id @GeneratedValue
   @Column (name="ValueId")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   @Column (name="ValueText")
   public String getText() {
      return text;
   }

   public void setText(String text) {
      this.text = text;
      this.updated();
   }

   @Column (name="ValueColor")
   public String getColor() {
      return color;
   }

   public void setColor(String color) {
      this.color = color;
      this.updated();
   }

   @Column (name="ValueSorting")
   public int getSorting() {
      return sorting;
   }

   public void setSorting(int sorting) {
      this.sorting = sorting;
      this.updated();
   }

   @Column (name="ValueCreated")
   @Temporal (TemporalType.TIMESTAMP)
   public Date getCreated() {
        return created;
    }

    public void setCreated(Date created) {
        this.created = created;
    }

    @Column (name="ValueModified")
    @Temporal (TemporalType.TIMESTAMP)
    public Date getModified() {
        return modified;
    }

    public void setModified(Date modified) {
        this.modified = modified;
    }

    // Updates the modification date on an object
    private void updated() {
        if (this.id == 0)
            this.created = new Date();
        this.modified = new Date();
    }

    // Overriding equals method check on id only
    public boolean equals(Object o) {
        if (o == null)
            return false;
        else {
            Value otherValue = (Value) o;
            return this.getId() == otherValue.getId();
        }
    }
}
```

D.3 Session Beans

The session beans models the business logic of the domain. Each session bean contains methods modeling the functionality and the methods in a session bean is often related.

Session beans implements inferfaces either *local* or *remote*. These interfaces is not included here. All methods in the session beans is specified in the interface as to be available to the EJB container.

D.3.1 ReviewAddBean.java

Adds a new review to the system

```
package dk.eapv.ejb.business;

import javax.ejb.Remove;
import static javax.persistence.PersistenceContextType.EXTENDED;
import javax.ejb.Stateful;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.jboss.annotation.ejb.cache.simple.CacheConfig;
import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Destroy;
import org.jboss.seam.annotations.Factory;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Logger;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.log.Log;

import dk.eapv.ejb.domain.Review;
import dk.eapv.ejb.service.ReviewAdd;

@Stateless
@Name ("reviewAdd")
@Scope (ScopeType.PAGE)
public class ReviewAddBean implements ReviewAdd {

    private Review review;

    @PersistenceContext
        private EntityManager entityManager;

    @Logger
        private Log log;

    // Persists the review
    public void reviewAdd() {
        if (review != null) {
            entityManager.persist(review);
            log.debug("Review Persisted");
        }
    }

    public void destroy() {}

    // Setter method for review
    @In (required = false)
    public void setReview(Review review) {
        this.review = review;
    }
}
```

D.3.2 EvaluationConservationBean.java

This bean handles the entering of an evaluation. The bean starts a conversation that runs through the evaluation and ends when the evaluation is saved.

```
package dk.eapv.ejb.business;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Begin;
import org.jboss.seam.annotations.Conversational;
import org.jboss.seam.annotations.Destroy;
import org.jboss.seam.annotations.End;
import org.jboss.seam.annotations.Factory;
import org.jboss.seam.annotations.Logger;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.annotations.datamodel.DataModel;
import org.jboss.seam.log.Log;

import dk.eapv.ejb.domain.Answer;
import dk.eapv.ejb.domain.Category;
import dk.eapv.ejb.domain.Evaluation;
import dk.eapv.ejb.domain.Question;
import dk.eapv.ejb.domain.Review;
import dk.eapv.ejb.domain.Schema;
import dk.eapv.ejb.domain.Value;
import dk.eapv.ejb.service.EvaluationConversation;
import static javax.persistence.PersistenceContextType.EXTENDED;

@Stateful
@Name ("evaluationConversation")
@Scope (ScopeType.CONVERSATION)
@Conversational
public class EvaluationConversationBean implements EvaluationConversation {

   @DataModel
   private List<Answer> evaluationAnswerList;

   @PersistenceContext (type=EXTENDED)
   private EntityManager entityManager;

   @Logger
   private Log log;

   private Category evaluationCategory;
   private List<Value> valueList;
   private String evaluationId;
   private int categoryCount = 0;
   private Review review;
   private Schema schema;
   private List<Category> categoryList;
   private Map<Category, List<Answer>> totalAnswerMap;


   // Start the evaluation by getting the review
   // and begins a conversation
   // For release 1 a dummy value is used to select review id 1.
   // This should be selected from a list
```

```
@Begin
public void startEvaluation() {
   try {
      // TODO: Should be selected from a list instead
      review = (Review)entityManager.
         createQuery("FROM Review rew WHERE rew.id = 1").getSingleResult();
      schema = review.getSchema();
      categoryList = schema.getCategories();
      totalAnswerMap = new HashMap<Category, List<Answer>>();

      // Creates a list of values
      // Should be refactored to list factory
      this.findValueList();

         // Creates an empty answer list
         for (Category category : categoryList) {
            ArrayList<Answer> answerList = new ArrayList<Answer>();

            // Creates an answer in the answer list
            // for each question in the current category
            for (Question question : category.getQuestions()) {
               Answer currentAnswer = new Answer();
               currentAnswer.setQuestion(question);
               answerList.add(currentAnswer);
            }
            totalAnswerMap.put(category, answerList);
         }
      }
      catch (Exception e) {
         log.debug("EvaluationBean: Find review failed");
      }
   }

// Creates a list of answers for the current category
@Factory ("evaluationCategory")
public void findAnswersListForCategory() {
   evaluationCategory = categoryList.get(categoryCount);
   evaluationAnswerList = totalAnswerMap.get(evaluationCategory);
}

// Creates a list of values. Should be refactored to a list factory
@Factory ("valueList")
public void findValueList() {
   valueList = entityManager.
      createQuery("FROM Value val ORDER BY val.sorting").getResultList();
}

// Moves the pointer "evaluationCategory" to the next category
// and calls method for creating answers
public void nextCategory() {
   if (categoryCount < categoryList.size()-1) {
      // Move to next category
      categoryCount++;
      this.findAnswersListForCategory();
   }
   else {
      // End the evaluation
      this.saveEvaluation();
   }
}

// Moves the pointer "evaluationCategory" to the previous
// category. If current is the first nothing happens
public void previousCategory() {
   if (categoryCount > 0)
```

```
         categoryCount--;
      this.findAnswersListForCategory();
   }

   // Saves the evaluation and sets the evaluationId
   // The method ends the conversation.
   @End
   public void saveEvaluation() {
      Evaluation evaluation = new Evaluation();
      evaluation.setReview(review);
      List<Answer> saveList = new ArrayList<Answer>();

      for (List<Answer> list : totalAnswerMap.values()) {
         saveList.addAll(list);
      }
      evaluation.setAnswers(saveList);
      entityManager.persist(evaluation);
      evaluationId = review.getId() + "-" + evaluation.getId();
   }

   // Destroy method for stateful bean
   @Remove
   @Destroy
   public void destroy() {}

   // Getter method for evaluationCategory
   @Out (required = false)
   public Category getEvaluationCategory() {
      return evaluationCategory;
   }

   // Getter method for valueList
   @Out (required = false)
   public List<Value> getValueList() {
      return valueList;
   }

   // Getter method for evaluationId
   // (set when evaluation is ended and saved)
   @Out (required = false)
   public String getEvaluationId() {
      return evaluationId;
   }
}
```

D.3.3 EvaluationResultBean.java

This bean calculates the result of a review. The results are calculated from the selected category.

```
package dk.eapv.ejb.business;

import static javax.persistence.PersistenceContextType.EXTENDED;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```java
import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.jboss.seam.annotations.Destroy;
import org.jboss.seam.annotations.Factory;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Logger;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.log.Log;

import dk.eapv.ejb.domain.Answer;
import dk.eapv.ejb.domain.Category;
import dk.eapv.ejb.domain.Evaluation;
import dk.eapv.ejb.domain.Question;
import dk.eapv.ejb.domain.Schema;
import dk.eapv.ejb.domain.Value;
import dk.eapv.ejb.service.Categories;
import dk.eapv.ejb.service.Departments;
import dk.eapv.ejb.service.EvaluationResult;

@Stateful
@Name("evaluatioResult")
public class EvaluationResultBean implements EvaluationResult {

    Departments departments;
    Categories categories;
    private List<Question> resultQuestionList;
    private Map<Question, Map<Value, Double>> resultValueMap;

    @PersistenceContext (type=EXTENDED)
        private EntityManager entityManager;

    @Logger
        private Log log;

    // Finds the questions for the current review
    // Not implemented correctly in release 1
    // Uses a dummy value to find schema
    @Factory ("resultQuestionList")
    public void findResultQuestionList() {
        if (categories != null)
            resultQuestionList = categories.getSelectedCategory().
                                    getQuestions();
        else {
            // TODO: Change from dummy value to
            // a value based on a review selected
            // from a review list
            try {
                Schema defaultSchema = (Schema)entityManager.
                    createQuery("FROM Schema sch WHERE sch.id = 1").
                    getSingleResult();
                Category defaultCategory = defaultSchema.getCategories().get(0);
                resultQuestionList = defaultCategory.getQuestions();
            }
            catch (Exception e) {
                log.debug("EvaluationResultBean: Find schema failed");
            }
        }
    }

    // Calculates the fraction of each value
    // for the total answers of questions
// The fraction is mapped to the value and
// the value-fraction map is mapped to the
// correct question.
// Is not implemented correctly in release 1
// Uses a dummy value
@Factory ("resultValueMap")
public void findResultValueMap() {
   // Find possible Values
   List<Value> valueList = entityManager.
      createQuery("FROM Value val ORDER BY val.id").getResultList();

   // Find evaluations for selected department and review
   // TODO: Should be changed to use a review selected
   // from a review list
   List<Evaluation> evaluationList = entityManager.
      createQuery("FROM Evaluation eval WHERE eval.department=1 AND
            eval.review=1")
      .getResultList();

   // Initiate ResultMap with question value integer pairs
   resultValueMap = new HashMap<Question, Map<Value, Double>>();

      // Array for questions values and count
      int[][] noOfAnswersPerQuestionArray =
         new int[resultQuestionList.size()][valueList.size()];

      // Count number of values for each question
      for (Evaluation ev : evaluationList) {
         for (Answer ans : ev.getAnswers()) {
            noOfAnswersPerQuestionArray[resultQuestionList.
               indexOf(ans.getQuestion())]
               [valueList.indexOf(ans.getValue())] += 1;
         }
      }

      // Count a values fraction of the total answers to a question
      int indexOfQuestionInList = 0;
      for (int[] questionCounts : noOfAnswersPerQuestionArray) {
         int totalAnswersForQuestion = 0;
         for (int valueCounts : questionCounts) {
            totalAnswersForQuestion += valueCounts;
         }
         Map<Value, Double> valueFractions = new HashMap<Value, Double>();
         int indexOfValueInList = 0;
         for (int valueCounts : questionCounts) {
            valueFractions.put(valueList.get(indexOfValueInList),
                  (double)(valueCounts/totalAnswersForQuestion));
         }
         resultValueMap.put(resultQuestionList.get(indexOfQuestionInList),
               valueFractions);
      }
   }

// Destroy method for stateful bean
@Destroy @Remove
public void destroy() {}

// Setter method for departmens
@In (required = false)
public void setDepartments(Departments departments) {
   this.departments = departments;
}

// Setter method for categories
@In (required = false)
```

```
    public void setCategories(Categories categories) {
        this.categories = categories;
    }

    // Getter method for resultQuestionList
    @Out (required = false)
    public List<Question> getResultQuestionList() {
        return resultQuestionList;
    }

    // Getter method for resultValueMap
    @Out (required = false)
    public Map<Question, Map<Value,Double>> getResultValueMap() {
        return resultValueMap;
    }
}
```

D.3.4 SchemasBean.java

Factory for creating a list of schemas

```
package dk.eapv.ejb.business;

import java.util.List;

import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import static javax.persistence.PersistenceContextType.EXTENDED;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Conversational;
import org.jboss.seam.annotations.Destroy;
import org.jboss.seam.annotations.Factory;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Logger;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.annotations.datamodel.DataModel;
import org.jboss.seam.annotations.datamodel.DataModelSelection;
import org.jboss.seam.log.Log;

import dk.eapv.ejb.domain.Category;
import dk.eapv.ejb.domain.Schema;
import dk.eapv.ejb.service.Categories;
import dk.eapv.ejb.service.Schemas;

@Stateful
@Name ("schemas")
public class SchemasBean implements Schemas {

    @PersistenceContext(type=EXTENDED)
        private EntityManager entityManager;

    @DataModel
        List<Schema> schemaList;
    @DataModelSelection
       Schema schema;

    @Logger
       Log log;

    // Returns the schema selected from the datamodel
    public Schema getSelectedSchema() {
       return schema;
    }

    // Creates a list of schemas ordered by the id
    // descending thereby putting the newest schemas
    // at the head of the list
    @Factory("schemaList")
       public void findSchemas() {
          schemaList = entityManager.
             createQuery("FROM Schema scm ORDER BY scm.id DESC").getResultList();
       }

    // Destroy method for stateful bean
    @Destroy
       @Remove
       public void destroy() {}
}
```

D.3.5 CategoriesBean.java

Factory for creating a list of categories. The list is based on the selected schema

```
package dk.eapv.ejb.business;

import java.util.List;

import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;


import org.jboss.seam.annotations.Destroy;
import org.jboss.seam.annotations.Factory;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Logger;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.datamodel.DataModel;
import org.jboss.seam.annotations.datamodel.DataModelSelection;
import org.jboss.seam.log.Log;


import dk.eapv.ejb.domain.Category;
import dk.eapv.ejb.domain.Schema;
import dk.eapv.ejb.service.Categories;
import dk.eapv.ejb.service.SchemaEditor;
import static javax.persistence.PersistenceContextType.EXTENDED;

@Stateful
```

```
@Name ("categories")
public class CategoriesBean implements Categories {
   private Schema schema;
   private SchemaEditor schemaEditor;

   @PersistenceContext(type=EXTENDED)
      private EntityManager entityManager;

   @DataModel
      private List<Category> categoryList;

   @DataModelSelection
      private Category category;

   @Logger
      private Log log;



   // Creates a list of categories based on the
   // active schema of the schemaEditor
   @Factory("categoryList")
      public void findCategories() {
         schema = schemaEditor.getSelectedSchema();
         if (schema == null) {
            log.debug("CategoriesBean: schema is null");
            categoryList = null;
         }
         else
            categoryList = schema.getCategories();
      }

   // Moves the category selected from the datamodel
   // one step up in the sorting rank
   public void categoryMoveUp() {
      if (category.getSorting() > 1) {
         int indexOfCurrentCategory = categoryList.indexOf(category);
         if (indexOfCurrentCategory != 0) {
            // Current category
            categoryList.remove(indexOfCurrentCategory);
            category.setSorting(category.getSorting()-1);
            categoryList.add(indexOfCurrentCategory -1, category);

            // The switched category
            Category nextCategory = categoryList.remove(indexOfCurrentCategory);
            nextCategory.setSorting(nextCategory.getSorting()+1);
            categoryList.add(indexOfCurrentCategory , nextCategory);
         }
      }
   }

   // Destroy method of stateful bean
   @Destroy
      @Remove
      public void destroy() {}

   // Setter method for schema
   @In (required = false)
      public void setSchema(Schema schema) {
         this.schema = schema;
      }

   // Setter method for schemaEditor
   @In (required = false)
      public void setSchemaEditor(SchemaEditor schemaEditor) {
        this.schemaEditor = schemaEditor;
      }

   // Getter method for the category selected from the
   // datamodel
   public Category getSelectedCategory() {
      return category;
   }
}
```

D.3.6 QuestionsBean.java

Factory for creating a list of questions. The list is based on the selected category.

```
package dk.eapv.ejb.business;

import java.util.List;

import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import static javax.persistence.PersistenceContextType.EXTENDED;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Conversational;
import org.jboss.seam.annotations.Destroy;
import org.jboss.seam.annotations.Factory;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Logger;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.annotations.datamodel.DataModel;
import org.jboss.seam.annotations.datamodel.DataModelSelection;
import org.jboss.seam.log.Log;


import dk.eapv.ejb.domain.Category;
import dk.eapv.ejb.domain.Question;
import dk.eapv.ejb.domain.Schema;
import dk.eapv.ejb.service.Categories;
import dk.eapv.ejb.service.Questions;
import dk.eapv.ejb.service.SchemaEditor;

@Stateful
@Name("questions")
public class QuestionsBean implements Questions {

   private Categories categories;
   private Category selectedCategory;

   @PersistenceContext(type=EXTENDED)
      private EntityManager entityManager;

   @DataModel
      List<Question> questionList;
```

```
    @DataModelSelection
        Question question;

    @Logger
        Log log;

    // Returns the question selected from the
    // datamodel
    public Question getSelectedQuestion() {
        return question;
    }

    // Produces a list of questions based on the
    // selected category
    public void findQuestions() {
        selectedCategory = categories.getSelectedCategory();
        if (selectedCategory == null) {
            log.debug("QuestionsBean: selectedCategory is null");
        }
        else
            questionList = selectedCategory.getQuestions();
    }

    // Moves the selected question form the datamodel
    // one step up in the sorting rank
    public void questionMoveUp() {
        if (question.getSorting() > 1) {
            int indexOfCurrentQuestion = questionList.indexOf(question);
            if (indexOfCurrentQuestion != 0) {
                // Current question
                questionList.remove(indexOfCurrentQuestion);
                question.setSorting(question.getSorting()-1);
                questionList.add(indexOfCurrentQuestion-1, question);

                // The switched question
                Question nextQuestion = questionList.remove(indexOfCurrentQuestion);
                nextQuestion.setSorting(nextQuestion.getSorting()+1);
                questionList.add(indexOfCurrentQuestion, nextQuestion);
            }
        }
    }

    // Destroy method for stateful bean
    @Destroy
        @Remove
        public void destroy() {}

    // Setter method for Categories
    @In (required = false)
        public void setCategories(Categories categories) {
            this.categories = categories;
        }

    // Getter method for Category
    @Out (required = false)
        public Category getCategory() {
            return selectedCategory;
        }
}
```

D.3.7 ReviewsBean,java

Factory for creating a list of reviews. The list is based on the selected department. Not fully implemented in release 1

```
package dk.eapv.ejb.business;

import java.util.List;

import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.jboss.annotation.ejb.cache.simple.CacheConfig;
import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Conversational;
import org.jboss.seam.annotations.Destroy;
import org.jboss.seam.annotations.Factory;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Logger;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.annotations.datamodel.DataModel;
import org.jboss.seam.annotations.datamodel.DataModelSelection;
import org.jboss.seam.log.Log;

import dk.eapv.ejb.domain.Review;
import dk.eapv.ejb.domain.Schema;
import dk.eapv.ejb.service.Reviews;
import dk.eapv.ejb.service.SchemaEditor;
import static javax.persistence.PersistenceContextType.EXTENDED;

@Stateful
@Name ("reviews")
  public class ReviewsBean implements Reviews {
    @PersistenceContext(type=EXTENDED)
      private EntityManager entityManager;

    @DataModel
      List<Review> reviewList;

    @DataModelSelection
      Review review;

    @Logger
      Log log;

    // Returns the review selected from the datamodel
    public Review getSelectedReview() {
      return review;
    }

    // Created a list of reviews based on the selected
    // department.
    // Not fully implemented - uses dummy value
    @Factory("reviewList")
      public void findReviews() {
        reviewList = entityManager.
          createQuery("FROM Review rew WHERE rew.id=1").getResultList();
```

```
            }

        // Destroy method for stateful bean
        @Destroy
            @Remove
            public void destroy() {}
}
```

D.3.8 DepartmentsBean.java

Factory for creating a list of departments

```
package dk.eapv.ejb.business;

import java.util.List;

import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import static javax.persistence.PersistenceContextType.EXTENDED;

import org.jboss.seam.annotations.Destroy;
import org.jboss.seam.annotations.Factory;
import org.jboss.seam.annotations.Logger;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.annotations.datamodel.DataModel;
import org.jboss.seam.annotations.datamodel.DataModelSelection;
import org.jboss.seam.log.Log;

import dk.eapv.ejb.domain.Department;
import dk.eapv.ejb.service.Departments;

@Stateful
@Name ("departments")
public class DepartmentsBean implements Departments {

    private Department selectedDepartment;

    @PersistenceContext(type=EXTENDED)
        private EntityManager entityManager;

    @DataModel
        List<Department> departmentList;

    @DataModelSelection
        Department department;

    @Logger
        Log log;

    // Creates a list of departments ordered by
    // the name of the department
    @Factory("departmentList")
        public void findDepartments() {
            departmentList = entityManager.
                createQuery("FROM Department dpt ORDER BY dpt.name").getResultList();
            log.debug("Lookup Departments");
        }

    // Destroy method of stateful bean
    @Destroy
        @Remove
        public void destroy() {}

    // Getter method for department
    @Out (required = false)
        public Department getSelectedDepartment() {
            return selectedDepartment;
        }
}
```

D.3.9 TimeBean.java

Simple time bean providing a date for display

```
package dk.eapv.ejb.business;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Locale;

import javax.ejb.Stateless;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Factory;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.annotations.Scope;

import dk.eapv.ejb.service.Time;

@Stateless
@Name ("timeBean")
@Scope (ScopeType.PAGE)

public class TimeBean implements Time {

    private String dateShort;

    // Method for creating a short date time string
    // Uses Danish Locale
    @Factory ("dateShort")
    public void publishDateShort() {
        Calendar cal = Calendar.getInstance(new Locale("da", "DK"));
        SimpleDateFormat sdf =
            new SimpleDateFormat("d. MMMMMMMMMM yyyy", new Locale("da", "DK"));
        dateShort = sdf.format(cal.getTime());
    }

    // Getter method for short time string
    @Out
    public String getDateShort() {
        return dateShort;
    }
}
```

D.4 Configuration files

The JEE application is configured using *xml* files. Most of the files are described in the first part of the report.

D.4.1 ejb-jar.xml

The *ejb-jar.xml* file normally configures the beans of the application thereby making them manages beans. In JBoss Seam this is done through interceptors. These are configures here to make them available to the EJB container.

```
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
            http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
         version="3.0">

   <interceptors>
     <interceptor>
        <interceptor-class>
           org.jboss.seam.ejb.SeamInterceptor
        </interceptor-class>
     </interceptor>
   </interceptors>

   <assembly-descriptor>
      <interceptor-binding>
         <ejb-name>*</ejb-name>
         <interceptor-class>
            org.jboss.seam.ejb.SeamInterceptor
         </interceptor-class>
      </interceptor-binding>
   </assembly-descriptor>

</ejb-jar>
```

D.4.2 persistence.xml

The persistence unit is configured in this file. It is tightly connedted with the *eapv-ds.xml* file which configures the data source - that is the connection to the database.

```
<persistence>
   <persistence-unit name="eapv">
      <jta-data-source>java:/eapvDS</jta-data-source>
      <properties>
         <property name="hibernate.dialect"
```

```
            value="org.hibernate.dialect.MySQLInnoDBDialect"/>
         <property name="jboss.entity.manager.factory.jndi.name"
            value="java:/EntityManagerFactories/entityManager"/>
      </properties>
   </persistence-unit>
</persistence>
```

D.4.3 faces-config.xml

In standard JSF applications the navigation rules of the application is specified in this file. In the APV application this is handled by JBoss Seam - and this is configured here. The language bundle is also configured in this file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config
PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
"http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
<faces-config>

   <!-- A phase listener is needed by all Seam applications -->

   <lifecycle>
      <phase-listener>
         org.jboss.seam.jsf.SeamPhaseListener
      </phase-listener>
   </lifecycle>

   <application>
      <view-handler>
         org.jboss.seam.ui.facelet.SeamFaceLetViewHandler
      </view-handler>

      <message-bundle>eapv</message-bundle>
   </application>

</faces-config>
```

D.4.4 pages.xml

This file contains the JBoss Seam navigation rules for the navigation in the application.

```
<!DOCTYPE pages PUBLIC
"-//JBoss/Seam Pages Configuration DTD 1.1//EN"
"http://jboss.com/products/seam/pages-1.1.dtd">

<pages no-conversation-view-id="/home.xhtml">

   <page view-id="/home.xhtml">
      <navigation from-action="#{evaluationConversation.startEvaluation}">
```

```
        <redirect view-id="evaluationConversation.xhtml">
        </navigation>
</page>


<page view-id="/schemaEditor.xhtml">
    <navigation from-action="#{schemaEditor.categoryEdit}">
        <redirect view-id="/categoryEdit.xhtml"/>
    </navigation>

    <navigation from-action="#{schemaEditor.categoryAdd}">
        <redirect view-id="/categoryAdd.xhtml"/>
    </navigation>

    <navigation from-action="#{schemaEditor.questionEdit}">
        <redirect view-id="/questionEdit.xhtml"/>
    </navigation>

    <navigation from-action="#{schemaEditor.questionAdd}">
        <redirect view-id="/questionAdd.xhtml"/>
    </navigation>

    <navigation from-action="#{schemaEditor.schemaSave}">
        <redirect view-id="/home.xhtml"/>
    </navigation>

    <navigation from-action="#{evaluationConversation.startEvaluation}">
        <redirect view-id="evaluationConversation.xhtml">
        </navigation>
</page>

<page view-id="/schemaAdd.xhtml">
    <navigation from-action="#{schemaEditor.schemaAdd}">
        <redirect view-id="/schemaEditor.xhtml"/>
    </navigation>

    <navigation from-action="#{evaluationConversation.startEvaluation}">
        <redirect view-id="evaluationConversation.xhtml">
        </navigation>
</page>

<page view-id="/schemaList.xhtml">
    <navigation from-action="#{schemaEditor.schemaEdit}">
        <redirect view-id="/schemaEditor.xhtml"/>
    </navigation>

    <navigation from-action="#{evaluationConversation.startEvaluation}">
        <redirect view-id="evaluationConversation.xhtml">
        </navigation>
</page>

<page view-id="/categoryAdd.xhtml">
    <navigation from-action="#{schemaEditor.categorySave}">
        <redirect view-id="/schemaEditor.xhtml"/>
    </navigation>

    <navigation from-action="#{evaluationConversation.startEvaluation}">
        <redirect view-id="evaluationConversation.xhtml">
        </navigation>
</page>

<page view-id="/categoryEdit.xhtml">
    <navigation from-action="#{schemaEditor.categorySave}">
        <redirect view-id="/schemaEditor.xhtml"/>
    </navigation>
```

```
      <navigation from-action="#{evaluationConversation.startEvaluation}">
         <redirect view-id="evaluationConversation.xhtml">
         </navigation>
   </page>

   <page view-id="/questionAdd.xhtml">
      <navigation from-action="#{schemaEditor.questionSave}">
         <redirect view-id="/schemaEditor.xhtml"/>
      </navigation>

      <navigation from-action="#{evaluationConversation.startEvaluation}">
         <redirect view-id="evaluationConversation.xhtml">
         </navigation>
   </page>

   <page view-id="/questionEdit.xhtml">
      <navigation from-action="#{schemaEditor.questionSave}">
         <redirect view-id="/schemaEditor.xhtml"/>
      </navigation>

      <navigation from-action="#{evaluationConversation.startEvaluation}">
         <redirect view-id="evaluationConversation.xhtml">
         </navigation>
   </page>

   <page view-id="/reviewAdd.xhtml">
      <navigation from-action="#{reviewAdd.reviewAdd}">
         <redirect view-id="/home.xhtml"/>
      </navigation>

      <navigation from-action="#{evaluationConversation.startEvaluation}">
         <redirect view-id="evaluationConversation.xhtml">
         </navigation>
   </page>

   <page view-id="/evaluationConversation.xhtml">
      <navigation from-action="#{evaluationConversation.startEvaluation}">
         <redirect view-id="evaluationConversation.xhtml">
         </navigation>
   </page>
</pages>
```

D.4.5 web.xml

This file contains configuration for using JBoss Seam and MyFaces in the web application (view). Most of it is standard configuration for JBoss Seam applications.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5"
   xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
```

```
<!-- JBoss Seam -->
<listener>
   <listener-class>org.jboss.seam.servlet.SeamListener</listener-class>
</listener>

<!-- Propagate conversations across redirects -->
<filter>
   <filter-name>Seam Redirect Filter</filter-name>
   <filter-class>org.jboss.seam.servlet.SeamRedirectFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>Seam Redirect Filter</filter-name>
   <url-pattern>*.seam</url-pattern>
</filter-mapping>

<filter>
   <filter-name>Seam Exception Filter</filter-name>
   <filter-class>
      org.jboss.seam.servlet.SeamExceptionFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>Seam Exception Filter</filter-name>
   <url-pattern>*.seam</url-pattern>
</filter-mapping>

<!-- JSF -->

<context-param>
   <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
   <param-value>client</param-value>
</context-param>

<context-param>
   <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
   <param-value>.xhtml</param-value>
</context-param>

<context-param>
   <param-name>facelets.DEVELOPMENT</param-name>
   <param-value>true</param-value>
</context-param>

<servlet>
   <servlet-name>Faces Servlet</servlet-name>
   <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>

<!-- Faces Servlet Mapping -->
<servlet-mapping>
   <servlet-name>Faces Servlet</servlet-name>
   <url-pattern>*.seam</url-pattern>
</servlet-mapping>

<!-- MyFaces -->
<listener>
   <listener-class>
      org.apache.myfaces.webapp.StartupServletContextListener
   </listener-class>
</listener>

<session-config>
   <session-timeout>15</session-timeout>
```

```
    </session-config>

    <context-param>
        <param-name>facelets.LIBRARIES</param-name>
        <param-value>/WEB-INF/tomahawk.taglib.xml</param-value>
    </context-param>
</web-app>
```

D.4.6 eapv-ds.xml

This file configures the connection to the database server. The files is often placed outside the *.ear* package deployed to the JEE server.

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>
    <local-tx-datasource>
        <jndi-name>eapvDS</jndi-name>
        <connection-url>jdbc:mysql://localhost:3306/eapv</connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>java</user-name>
        <password>java</password>
        <dbname>eapv</dbname>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>20</max-pool-size>
        <idle-timeout-minutes>5</idle-timeout-minutes>
        <track-statements/>
        <exception-sorter-class-name>
            com.mysql.jdbc.integration.jboss.ExtendedMysqlExceptionSorter
        </exception-sorter-class-name>
        <valid-connection-checker-class-name>
            com.mysql.jdbc.integration.jboss.MysqlValidConnectionChecker
        </valid-connection-checker-class-name>
    </local-tx-datasource>
</datasources>
```

Bibliography

[1] Enterprise JavaBeans 3.0 • Bill Burke & Richard Monson-Haefel
O'Reilly, 2006, 5th edition • ISBN: 9780596009786

This book is called the definitive guide for EJB 3.0 and the Java Persistence API. The book covers the three kinds of EJBs and how the are used in Java EE 5.

[2] Beginning JBoss Seam • Joseph Faisal Nusairat
APress, 2007 • ISBN: 1590597923

This book is one of the first on the JBoss Seam framework. It gives a good introduction to the framework, but is not an in-depth advanced reference book.

[3] Pro EJB 3 - Java Persistence API • Michael Keith
APress, 2006 • ISBN: 1590596455

[4] Java Precisely • Peter Sestoft
The MIT Press, 2005, 2nd edition • ISBN: 0262693259

This book is a concise reference to Java version 5.0. It is not a starters book as it presents the entire language in less than 150 pages.

[5] JBoss at Work: A Practical Guide •Tom Marrs & Scott Davis
O'Reilly, 2006 • ISBN: 0596007345

This book is a good starters guide for setting up JBoss Application Server as your JEE platform. Unfortunately it only covers J2EE and therefore not any of the newer specifications of JEE 5.

[6] Introduktion til Extreme programming • Kent Beck
IDG, 2002 • ISBN: 8778435099

This book is called the ultimate book on Extreme Programming and is written by Kent Beck, who is one of the developers of Extreme Programming. This is the danish translation of the book.

[7] Database Systems: The Complete Book • Hector Garcia-Molina & Jeffrey D. Ullman & Jennifer Widom
Prentice-Hall, Int. Edition 2001 • ISBN: 0130980439

[8] Slides from Component Based Design and J2EE at ITU • Rasmus Lund & Jakob Bendsen
IT University, Copenhagen 2006

The slides are part of the course material on J2EE held by Peter Seetoft, Jakob Benden and Rasmus Lund during spring 2006. The slides are the base of most of the section on Servlets.

[9] http://docs.jboss.com/seam/1.1.6.GA/reference/en/html/index.html

This website contains the reference documentation for JBoss Seam version 1.1.6.GA. Unfortunately the documentations is not complete but should explain most of the concepts.

[10] http://www.wikipedia.org

Wikipedia is a web-based free content encyclopedia project. The content of the encyclopedia is written by volunteers and everybody may edit all subjects.

Index