Component Design

KSU Student Portal

Version 1.0

Submitted in partial fulfillment of the requirements of the degree of MSE

Javier Ramos Rodríguez
CIS 895 – MSE Project
Kansas State University
# Table of contents

1. **Introduction** ................................................................. 4

2. **Model Driven Architecture** ................................................... 5

3. **System Architecture** ........................................................... 7

4. **Technology** ........................................................................ 10

   4.1. **Data Layer** ................................................................. 10

   4.2. **Model Layer** ............................................................... 10

       4.2.1. EJB 3.0 ................................................................. 10

       4.2.2. Java 5 ................................................................. 16

       4.2.3. JNDI ................................................................. 16

   4.3. **View Layer** ............................................................... 17

       4.3.1. Overview ............................................................ 17

       4.3.2. Servlets .............................................................. 18

       4.3.3. JSP Pages ............................................................ 19

       4.3.4. Java Server Faces .................................................. 21

           4.3.4.1. MVC Pattern .................................................. 21

           4.3.4.2. Struts ........................................................... 22

           4.3.4.3. JSF ............................................................... 25

   4.3.5. ADF ........................................................................ 29

   4.3.6. JavaScript ................................................................ 29

   4.3.7. Cascading Style Sheets (CSS) ...................................... 29
5.  Component Design ............................................................ 30
  5.1. Overview ........................................................................ 30
    5.1.1. Package Overview .................................................. 30
    5.1.2. Process Overview .................................................. 31
  5.2. Use Case Model ............................................................ 36
    5.2.1. Administration Package ......................................... 36
    5.2.2. Unregistered User Package .................................... 37
    5.2.3. Registered User Package ....................................... 38
  5.3. Packages ........................................................................ 39
    5.3.1. Model Layer ............................................................ 39
      5.3.1.1. Model Façade .................................................. 40
      5.3.1.2. Entity Beans ................................................... 43
      5.3.1.3. Implementation Classes .................................. 44
    5.3.2. View Layer ............................................................ 48
    5.3.3. Dynamic Behavior .................................................. 52
      5.3.3.1. Register User .................................................. 52
      5.3.3.2. Add Article ...................................................... 55
  6.  Implementation Design ..................................................... 58
  6.1. Folder Structure ........................................................... 58
  6.2. Model Layer ............................................................... 59
    6.2.1. Design Patterns ..................................................... 59
      6.2.1.1. Architectural Patterns ................................... 59
      6.2.1.2. Design Patterns .............................................. 62
    6.2.2. Algorithms ............................................................ 63
      6.2.2.1. Search Algorithm .......................................... 63
      6.2.2.2. Filter Algorithm ............................................. 64
  6.3. View Layer .................................................................... 68
    6.3.1. Internationalization ............................................... 68
    6.3.2. Input Validation ..................................................... 70
    6.3.3. JavaScript ............................................................ 70
    6.3.4. CSS .................................................................. 70
  6.4. Tools ............................................................................ 71
1. Introduction

The purpose of this document is to provide a component design for the KSU Student Portal. The design will outline the internal design of each component.

This document describes the technical issues of this project. For details about the functionality of the system check the Vision and Architecture Design documents. These documents complement the other two by providing implementations details of the system.

We first describe the target architecture of the system and the technology used. Then, we described the components, packages and classes. Finally, we discuss some implementation issues.

As a reminder, this project is an enterprise Web application that uses rigorous design methods and new technologies. The system is divided in three layers: data, model and view. The model is implemented using EJB 3.0 and the view with JSF 1.2.
2. Model Driven Architecture

One of the goals of this project is to follow a model driven approach where we build models of the system with different levels of details depending on the information that we have at a given time. This way, we can make general models that can fit in any platform allowing this to reuse models. This can be very useful to test designs and to build prototypes. For example, we can build a platform independent model and then built prototypes to test several architectures. We can implement directly from this high level model and get accurate results. But when it comes to the final product we need to create a more refine model that match the implementation.

In phase one, we gathered the requirements and we started to work on the solution. Phase 2 was a critical phase where we created a platform independent model. This model was not only platform/language independent but also architecture independent. Note that it didn’t require that the system was a Web application.

In this last phase we created the implementation and the component design which includes all the details needed to understand the system. It is important to take a look again to the vision document and the architecture design document to be able to understand this document and see how we followed the same structure just adding more details.

Basically, we took that model and implemented it into the J2EE platform as a Web application. We added all the technology and architecture (client-server) details needed to understand the final product.

Some functionality was added in this last phase. This is discussed in section 5.2.
Next figure shows the model driven architecture approach:

In this last phase, we tool the platform independent model and create a J2EE Web Application model. We skipped one possible model that could include Java/J2EE details but exclude web technologies details. Since, from the beginning it was clear that this was going to be a web application we skipped this step.

In this document we describe the last model that includes all the details including technology and architecture details. To properly understand this model we begin by giving a little introduction of the architecture and technology used.
3. System Architecture

The system is a Web application built using the J2EE platform. It is OS and DBMS independent; although it was built with Oracle in mind it may support other DBMS.

The application supports a distributed environment using EJBs as distributed components. The EJB 3.0 describes the middleware needed to implement all the security and networking functionality that the EJBs components need. There are many application servers that implement the EJB 3.0 specification and provide the middleware necessary to run this application. The system was tested and implemented using Oracle OC4J application server.

The system business logic will be handled by the EJBs components that will communicate with the database and perform different operations. The load can be balanced and these components can be distributed in several servers. There will be one main server that runs the application and controls the other servers that execute the EJBs. There will be also a database server that executes the DBMS.

The clients can connect from the internet or within the network. The key is that the server farm is secured and reliable, using high speed connections of at least 100Mbps to send and receive data. The application server and the EJBs components need to query the database server very often requesting data that need to be transmitted as fast as possible to achieve an acceptable level of performance.

In the figure below the target architecture is shown, some aspects can be changed since this application was built trying to be as flexible and portable as possible. But some issues such as high speed connection between the EJBs and database are very important and must be kept.
As we can see in the picture, clients can connect from the Internet or from the same network. There will be a firewall to protect the server farm from being attacked. Since this is a Web application clients can run any OS.

Since the application was built with a distributed environment in mind, some of the logic that could be done by the database server has been moved to the business logic layer and is performed by the session beans. This decision was made to avoid overloading the database server and distribute the load between different CPUs. So, to take full advantage of the application we will require at least four processors to execute the EJBs, one to execute the user’s logic (registration, get users, edit profile…), another one to execute the news/events logic, another one for the articles and another one for the rest of EJBs.
The events, news and articles need a considerable amount of computational capabilities since they execute complex search and filtering algorithms.

In the next figure, another view of the system is shown dividing the components in logical layer to see the architecture and technology from another point of view.

The clients send request using a Web browser that supports JavaScript. The Web server will manage the request and query application server to perform some action. The Web server will request the application server to look-up to find the corresponding EJB that will perform the action since the EJBs can be in different computer. Then, the EJBs will query the database to fetch data. After that the information is formatted and sends back to the Web server that will incorporate JSF/ADF UI components and generate the view that will be shown in the client browser. Usually, the application and Web server will be in the same machine.
4. Technology

In this section we will describe the technology used to develop the application. The goal is not to explain the technology; this is just a small overview that helps to explain why this technology was chosen. I focus on the main two technologies Java Server Faces and Enterprise JavaBeans 3.0. For more information about the J2EE platform check [14].

4.1. Data Layer

The goal of this project is to build an application that is database independent. We will use the EJB 3.0 technology that implements an O-R mapping approach to achieve this independency. We just require that the DBMS supports sequences to generate IDs.

4.2. Model Layer

In this section we describe the technology used to implement the business logic focusing mainly in the new EJB 3.0 technology.

4.2.1. EJB 3.0

Enterprise Java Beans is an architecture that defines the way of building distributed components on the server side. This technology guarantees that the programmed components are scalable, effective and safe. EJB technology uses other Java APIs such as RMI to provide all the middleware needed to develop enterprise applications. It provides physical situation transparency and helps modulating programming by having the components distributes in different machines transparently to the programmer. This helps to improve performance and scalability.
There are three types of EJBs:

1. **Session Beans (SB):** The session EJB is in charge of solving the business logic of the application. We can say that each method contained in a SB solves a use case (function) of the application, for example: User Control, Prices Agent or Processes Control.

There are two types of SB:

   1) **Stateless Session Beans (SLSB):** The SLSB is a component without state, this means, they don't keep any relationship between different calls from a client. Even more, between two calls to the same type of SLSB is possible that the container addresses the client to different instances of the component. Another fact of how they behave this type of components it is their relationship with the number of users. For N users there are M instances being M < N in most of the cases. The maximum value and minimum of M is configurable in most of the servers. With these two values is defined the size of the object pool.

   2) **Statefull Session Beans (SFSB):** The SFSB is the opposite of the SLSB, they maintain the state, this means, they have a conversation one to one with each client that invokes them.

2. **Entity Beans (SB):** The entity EJB is directly related with the application data, are objects that maintain in memory the data that the application manages like News, Forums, Users.

The Entity Beans usually map the relational database tables, although it is also possible that they maintain the data persistence in files, for example a XML, or in LDAP. In any of the cases the objective of an Entity Bean is to search the data in memory from a persistent source and to maintain a total synchronization between the state of the data in memory and the source of the data.
For this reason it is said that the Entity Bean are the EJB that survive to the system falls, in the case of a system failure, the data in memory is kept in a persistent device, with this, when the server is restarted they recovered without any problem.

There are two forms of managing persistence: one lets the programmer to manually take care of the persistence, in the other the container takes care of everything.

1) *Bean Managed Persistence*: The BMP is the Entity Bean that support persistence thanks to the programmer's explicit instructions. More and more often this type of Entity Beans is used less, but there are still some operations that can only be carried out thanks to the programmer abilities.

2) *Container Managed Persistence*: The CMP supports the persistence in a declarative or implicit way thanks to the container.

3. *Message-Driven Beans*: They are very similar to the session beans but they receive messages without responding to the client, in other words, they are asynchronous.

**EJB 2.x**

EJB 2.x is the previous version of the EJB specification and the most used one. It is crucial to understand this version before moving to the new EJB 3.0 specification. An EJB 2.x has the next parts:

- **EJB class** that implements the business methods and life cycle methods; uses other helper classes and libraries to implement.
- **Client-view API**: consists of *EJB home interface* and *remote interface*.
  
  - **Home interface**: controls life cycle: create, remove, and find methods. There should be one Home interface for each local or remote interface.
  
  - **Remote interface**: to invoke the EJB object methods (if the EJB is declared as remote).
  
  - **Local Interface**: Idem for local EJB.

- **Deployment Descriptor**: XML document for bean assembler and deployed on the container.
  
  - A declaration about EJB environment needed for customizing the bean to the operating environment.

- **Container Runtime services** include: transactions, security, distribution, load balancing, multithreading, persistence, failure recovery, resource pooling, state management, clustering…
A graphical representation of a session bean and an entity bean is how below:
As we can see, the problem of the EJB 2.x specification was its complexity. It did take care of a big part of the middleware but it required lot of effort to understand and develop these components for the following reasons:

1. Too many interfaces and methods to override because of the client-view API.
2. Too many deployment descriptors (XML files) which are really difficult to maintain.

The new EJB 3.0 was introduced to overcome these problems and make the development more easily. It has the following features:

1. Eliminates the home interfaces. Now EJBs are POJO.
2. Abstracts the use of local and remote interfaces make it easier to use. We just create the Bean as a POJO.
3. It uses an “Entity Manager” API that implements and Object-Relational Mapping (ORM) approach like Hibernate.
4. It uses the new Java 5 annotation to eliminate the use of XML files encapsulating the object with its configuration helping to maintain the system and to hide the information. It also reduces the number of files.
5. It includes many other minor improvements.

Summarizing, the two big changes are: the elimination of the interfaces, making development easier and the introduction of an ORM approach. Object-Relational mapping is a programming technique that links databases to object-oriented language concepts, creating the effect of a "virtual object database". Basically, EJB 3.0 takes the idea from Hibernate or Top-Link and adds another level of abstraction defining an API that uses Java 5 annotations instead of XML files to configure the mapping between objects and tables. This API can be implemented using Hibernate or Top-Link. For more information about ORM visit [http://en.wikipedia.org/wiki/Object-relational_mapping](http://en.wikipedia.org/wiki/Object-relational_mapping)
For these reasons and because it is an open standard supported by many organizations such as IBM or Oracle I have chosen this technology. For more information check [14, 15].

### 4.2.2. Java 5

Java 5 is the name of new Java API. It includes new interesting features. In the application the more important ones are used, which are:

1. **Metadata (Java annotation):** This feature provides the ability to associate additional data alongside Java classes, interfaces, methods, and fields. This additional data, or annotation, can be read by the `javac` compiler or other tools, and depending on configuration can also be stored in the class file and can be discovered at runtime using the Java reflection API.

2. **Generic Types and Enhanced Loops to support generic object lists.**

It also includes other features, for more information visit [16].

### 4.2.3. JNDI

JNDI is a Java technology used in distributed environment base on Java applications.

JNDI is a Java implementation of Naming and directory services that play a vital role in intranets and the Internet by providing network-wide sharing of a variety of information about users, machines, networks, services, and applications. A naming service maintains a set of bindings. Bindings relate names to objects. All objects in a naming system are named in the same way. Clients use the naming service to locate objects by name.

This technology is used to locate the EJBs and achieve a physical location transparency.
4.3. View Layer

4.3.1. Overview

In this section we will describe the technology used in the ViewController module. The technologies are organized in a hierarchy where some technologies rely on others. The next figure shows the relation between the technologies used:

In the first level we have the native Java code and its corresponding portable bytecode. To generate the view, two traditional view generation techniques are used in the second level: JSP pages and Servlets. Servlets are special Java classes that can generate HTML code to generate the view. JSP pages live in a JSP container that can dynamically compile the JSP pages to Servlets to generate dynamic web content. Two new technologies, JSF and ADF, are used within the JSP enriching the view by adding UI components.

Check [14] for more information about level two technologies.
4.3.2. Servlets

A Servlet is a software application written in Java that is executed in the server. The Java Servlet API allows a software developer to add dynamic content to a web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML.

A Servlet is an object that receives requests and generates a response based on the request. The API defines HTTP subclasses of the generic Servlet requests and responses as well as an HTTP session object that tracks multiple requests and responses between the web server and a client. Servlets may be packaged as a Web application. The main packages are `javax.servlet` and `javax.servlet.http`.

```
<<interface>>>
javax.servlet.Servlet

+ init(config : ServletConfig) : void
+ destroy() : void
+ service(request : ServletRequest, response : ServletResponse) : void

javax.servlet.GenericServlet

+ init(config : ServletConfig) : void
+ destroy() : void
+ service(request : ServletRequest, response : ServletResponse) : void

javax.servlet.http.HttpServlet

+ service(request : ServletRequest, response : ServletResponse) : void
# service(request : HttpServletRequest, response : HttpServletResponse) : void
# doGet(request : HttpServletRequest, response : HttpServletResponse) : void
# doPost(request : HttpServletRequest, response : HttpServletResponse) : void
```

The programmer in order to create a Servlet only needs to write a class which extends from `HttpServlet` and implement the method `doGet` or `doPost`. 
Before the JSP pages appear, the main use of the Servlets was to generate the view of an application web:

1. Receive petition HTTP associated to a URL
2. Read parameters
3. Invoke operation on the business model
4. Generate HTML

Nowadays, the JSP pages are use to generate the view (HTML) because as we’ll see later on Servlets have many limitations. JSP pages provide better mechanisms to generate dynamically Web content. But still, the Servlets are really important and are usually used as application controllers.

4.3.3. JSP Pages

The main problem with Servlets is that the text generate for the view is mix with Java code and this has some disadvantages:

- It is not possible to use tools for generation of HTML, WML, etc. directly.
- The text generation needs to be written by a person with knowledge of Java, which is economically very expensive.
- Changes in the aspect of the graphic interface require recompilation, creation of a new .war file and re-start the server. In an application web, especially in Internet, changes in the graphic interface are very common.
We want to be in a situation where there is a complete job separation:

- People that work on the graphic aspect with knowledge of graphic design and tools for generation of HTML and WML.
- People that implement the controller and the model like computer engineering with design and implementation knowledge.

JSP pages were created to overcome this Servlets limitation on the generation of the view. Servlets are still used as application controllers but JSP pages have substituted them in the view generation.

JavaServer Pages (JSP) is a group of technologies that allow the dynamic generation of web pages using a mark-up language like HTML or XML, to generate the content of the web pages.

To be able to use this technology it is necessary a web server that supports HTML pages, and code that implements a JSP container where we can execute the JSP tags. It’s important to realize that the JSP specification doesn’t require that JSP are implemented using Servlets (Java language). The main packages are `javax.servlet.jsp` and `javax.servlet.jsp.tagext`.

```java
<<interface>>
javax.servlet.Servlet

<<interface>>
javax.servlet.jsp.JspPage

<<interface>>
javax.servlet.jsp.HttpJspPage

+ _jspService(request : HttpServletRequest, response : HttpServletResponse) : void
```
4.3.4. Java Server Faces

Java Server Faces (JSF) is the new standard from Sun Microsystems that pretends to facilities web development by adding UI components to the JSP pages plus a MVC framework. We will begin by taking a look to the MVC pattern and the JSF rival Struts. Then, we will discuss more deeply the JSF.

4.3.4.1. MVC Pattern

The MVC architecture pattern (Model-View-Controller) is a pattern that defines the independent organization of the Model (Objects of Business), the View (interface with the user or another system) and the Controller (controller of the workflow of the application). It is used regularly in many areas especially in web applications.
4.3.4.2. Struts

Struts is a tool for the development of applications Web under the MVC patron using the J2EE platform. It is a framework that implements the MVC architecture pattern in Java. It works on any application server that implements the APIs Servlets and JSP.

A framework is the extension of a language by means of one or more classes hierarchies that implement a functionality and (optionally) they can be extended. The framework can involve TagLibraries.

The browser generates a request that is handled by the Controller (a specialized Servlet). The Servlet is in charge of analyzing the request, follow the configuration that has been programmed in the XML and call the corresponding Action (A java class that is used to access to the model) passing the corresponding parameters. The Action will instance or use the business objects (EJB) and they will do the task. Depending on the result of Action, the Controller will redirect to one or more JSP pages, which will be able to access the objects of the Model in order to carry out its task.
The *Servlet Front Controller* pattern is the key of Struts implementation providing the basic structure of the MVC pattern, the structure is how below:

Let's take a look to each class of the Struts MVC implementation:

- **ActionServlet.**
  - It's the Servlet Front Controller.
  - In `web.xml` file is specified that all the URLs that imply process (GET or POST) are redirect to this servlet.
    - E.g.: URLs that finishes in `.do`

- **ActionForm Classes.**
  - If the programmer wants it, can access to the request parameters through a *JavaBean* that extends from *ActionForm*.
    - Specially useful in forms.
**Action** Classes (method execute).

- It access to the request parameters, directly or via the corresponding *ActionForm*. It carries out the operation invoking a method from the model. Usually the *Session Facade* of the model. The Session Facade is a design pattern that will discuss later on.
- It leaves the result returned by the method in the request or in the session.
- It returns an *ActionForward* object that represents the URL that is necessary to visualize next (*sendRedirect* or *forward*).

As we can see, using Struts is quite complex, we need to write lots of classes and manage too many XML files. JSF has many similarities but it was introduced to overcome the difficulty of using Struts and make Web development easier.
4.3.4.3. JSF

JavaServer Faces (JSF) is a Java-based Web application framework that simplifies the development of user interfaces for Java EE applications. It has some similarities with Struts such as Tag libraries or state management. But it improves some of the techniques used in Struts and includes new technologies. For example, it includes UI components to enrich the view and make the development easier by using a drag & drop component design like Java Swing in Web applications. Also, it replaces the action classes and action forms with Managed Beans that take care of the UI components. Summarizing, JSF includes:

- A set of APIs for representing user interface (UI) components and managing their state, handling events and input validation, defining page navigation, and supporting internationalization and accessibility
- A default set of UI components
- Two JavaServer Pages (JSP) custom tag libraries for expressing a JavaServer Faces interface within a JSP page.
- A server-side event model
- State management
- Managed Beans (JavaBeans created with dependency injection)

JSF implements the Servlet Front Controller pattern too. The main controller is the `javax.faces.webapp.FacesServlet` class that must be implemented by the application server. Every time the user does something, such as clicking a button or submitting a form, an event occurs. Event notification is then sent via HTTP to the server. On the server is a web container that employs this special Servlet.
We also have a main configuration XML file where we include:

- Page Navigation Specification
- Internationalization Options
- Managed Beans
- Lifecycle configurations
- Message Bundle Configuration

Java Server Faces Technology relies on other technologies as shows in the picture below:

```
<table>
<thead>
<tr>
<th>JSF App</th>
<th>JSF App</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSF Tags</td>
<td>JSF API</td>
</tr>
<tr>
<td>JSP (1.2)</td>
<td></td>
</tr>
<tr>
<td>Servlets (2.3)</td>
<td></td>
</tr>
</tbody>
</table>
```

Basically, the JSF components are “injected” in the JSP pages and managed by its corresponding Bean. IDEs like JDeveloper support component drag & drop and automatic generation of components in the Java Bean with their corresponding getters and setters methods.
The main features are:

- **Page navigation specification**: In the main descriptor we specify the navigation flow of the pages. When clicking on a link or pressing a button the action will call a method (action) of the managed bean that will perform the action which usually is call the model. Then, the method returns a string that forwards to another page, this string is specified in the main descriptor and the main controller will be in charge of finding this string in the descriptor and look-up for the next page.

Sometimes when pressing a link we don’t perform any action, in this case, the action just return the string that will be use to find the next page. This way, we don’t depend on JSP pages names been easier to maintain. IDEs such as JDeveloper can read the main descriptor and build a graphic page flow diagram that makes navigation specification very easy.

- **Standard user interface components**: JSF specification is a standard that specifies Java UI components such buttons, links and inputs that can be drag & drop in the JSP pages to enrich the view.

- **User input validation**: JSF incorporates a tag library that includes standard validation tags to check the type or length of the inputs and send standard messages. Developers can create their on validation tags and change the error messages.

- **Easy error handling**: JSF provides mechanisms as JSF messages that make really easy show error messages to the users.

- **Java bean management**: JSF provides Java Beans to manage the JSP pages and its JSF components. They perform different actions in the view such validation or input conversion and are also in charge to call the model to perform the business logic.
The managed beans include all the UI components of the JSP page that they manage with the getter and setters methods.

- **Event handling**: JSF provide a Event handling mechanism that will call the Faces Servlets every time the user perform an action. This main server is in charge of controlling the navigation. It is the main controller; the managed beans are “sub controllers” that only control their corresponding JSP page.

- **Internationalization support**: JSF faces were built to support pages with several languages. It uses the `java.util.locale` and the main configuration file is used to configure the available languages. A message bundle can be used with several `message.properties` where we write all the messages, each file will be written in one of the supported languages.

- **IDE Integration**: JSF were created to help developers to build web interfaces like building desktop application. IDEs are been build to support this feature.

- **No Logic in JSP pages**: JSF help to separate roles and let the Web designer perform his/her job without Java language knowledge. Further some, no logic is used in the JSF tags. This improves the Struts or JSTL libraries where, although they provide a good set of tags, they still use loops and conditions. JSF doesn’t use conditions or loops inside the JSP pages.
4.3.5. ADF

Oracle Application Development Framework, usually called Oracle ADF, is a commercial framework for quickly creating enterprise applications. In this project we will use the ADF UI components provided by Oracle. Those are just a small part of the whole ADF framework focus on complementing the JSF components adding more features such as better tables, file upload menus and so on. These components work like the JSF components.

4.3.6. JavaScript

Java Script is client side code that is executed in the Web browser. This technology is useful to perform basic functions on the client side avoiding overload the network and the server.

4.3.7. Cascading Style Sheets (CSS)

CSS is a standard language created to help designing Web applications. This language allows authors and users to attach style (e.g., fonts, spacing, and aural cues) to structured documents (e.g., HTML documents and XML applications). By separating the presentation style of documents from the content of documents, CSS simplifies Web authoring and site maintenance.

For more information check [18].
5. Component Design

5.1. Overview

5.1.1. Package Overview

In this section, we will describe the system design. First of all, we are going to view the last version of the use case model describing the new functionalities that were added at the end of the process.

Then, we will see the packages and their classes. The system has to main modules: the model and the ViewController. The model implements the business logic using EJBs. The ViewController implements the view and the controllers of the MVC pattern.

The Model module contents two packages; one is the Junit test cases that execute the critical functions of the system. The other package contents the EJBs, session beans and entity beans that perform all business logic. It also contents a class that implements the session façade pattern and is in charges of look-up the EJBs.
The ViewController module depends on the model since it uses the session façade to interact with it. It also has two parts: the web content that includes the JSP pages, the JavaScript files, images, configuration files and all the tag libraries. Basically it has everything that the web server needs. The view.backing package has all the managed beans that the JSF needs to take care of the UI components. For each JSP page, there will be a managed bean that contains all the UI components used in the page and all the logic needed to manage them. That class is also in charge of calling the model and return the string that will be used for page navigation.

5.1.2. Process Overview

We are going to describe the general process of the system that begins when the user sends a request by performing an action and ends when the user receives the result HTML page in the browser.
Recall the graph from the high level analysis model from the vision document:

Now we are going to refine this model adding implementation details. The control classes from the view will change to Java Classes and Servlets, users will be JSP pages. Control classes from model will be Session Beans and Entity classes Entity Beans. Also, more details will be added.
The process is started by the user by clicking a link or filling a form and pressing the submit button. As we mention before, every time that the user perform an action the request is handled by the Faces Servlets that uses the main descriptor where we write all the navigation information. In case of link, we just want to change from page to page and we don’t need to call the model, further some, sometimes we don’t even have to perform any action in the managed bean, in this case the link will just have a string in the action field that will be used by the Faces Servlet to find the next page by looking the page navigation specification.
Then, the main Servlet will call the JSP page Servlet to dynamically generate the HTML page that is send to the user. Remember that every JSP page has a Servlet that is automatically compiled by the JSP container and the developer doesn’t need to care about it.

There are certain occasions that we don’t need to call the model but we won’t to perform an operation in the view that is done by the managed bean by performing an action just like using Struts. The method in the managed bean will return a string that will be used by the Faces Servlet to forward to the corresponding Servlet that will generate the HTML code. Recall that the managed bean contents all the JSF/ADF UI components that will be rendered (\texttt{<<include>>}) when the Servlet generates the view.

Finally, in some cases, usually when submitting form info, the whole process takes place:

1) User enters data and submits the form by clicking the submit button.
2) The Command Action calls a method of the managed bean of that particular page.
3) The method instantiates the model façade and calls a method from this class.
4) The method in the model façade looks up for the session bean that will perform the functionality.
5) The model façade instantiates the session bean and calls the corresponding method.
6) The session bean performs the business logic. Usually if the user request data, the session bean will create a entity bean or a generic list of entity beans using the entity manager. The entity manager follows an Object-Relational Mapping approach to map tables to objects (entity beans).
7) The results are returned to model façade and them back to the managed bean.
8) The managed bean returns a string that will be used by the Faces Servlet to look-up and forward to the next page.

9) The corresponding page Servlet is executed and it will include all the UI components from this new JSP page.

10) The Servlets generates dynamically the HTML code and the page is sent to the user.
5.2. Use Case Model

In this section, we will show the use case model to explain the new functionalities added to the system compare to those ones described in the vision document.

5.2.1. Administration Package

The administration besides removing users will also have the possibility to add and remove languages and courses from the system.

This feature was added to allow the administrator to add/remove languages/courses without manually entering the queries in the database, so it will not be necessary to access the server to add/remove these items making the maintenance a lot easier. The administrator just needs to enter the login and password to access to these functionalities. States and countries will be entering in the deployment and they will have a fix value that can be only changed manually by querying the database. We don’t expect countries/states to be added/removed very often.
5.2.2. Unregistered User Package

Visibility was changed in the implementation. Articles are always private not available to guest users since they content files and other important data. The user also has its own visibility that will permit or deny guest users to see his/her personal information. Also the blogs will have a visibility flag to permit or deny users to see the blog entries.
5.2.3. Registered User Package

In this case, a new functionality was added and another one removed. The message system was only partly implemented in the model but for schedule problems was not finally implemented. A new feature that permits the user to change password was added to the system.
5.3. Packages

5.3.1. Model Layer

A simplified version of the model is shown in the next figure:

To simplified the diagram some details were omitted and will be shown later one. The methods and attributes are omitted; the relations between Entity Beans are also omitted and object List is only shown once.

If we take a look at the figure, we can see that we have a class which is the entry point to the model and implements two design patterns: Session Façade and Business Delegate.
The Session Façade will instantiate the session beans that may be in a different CPU. The session beans will perform the desire operation often times returning an entity bean or a list of entity beans as shown in the figure. The session beans use the entity manager API to perform the database access.

5.3.1.1. Model Façade

As we mention before, this is a crucial element of the system. It is composed of a public interface where we publish all the methods available for the client and its corresponding implementation. This class implements the session façade and business delegate patterns. The idea of the Session Façade pattern is to have one single entry point in the system helping to maintainability allowing changes on the model without affecting the client (view).

It also reduces the amount of remote calls since this class will be the one that is going to make the remote calls and look up for the EJBs. So, the Session Façade will be local to the clients and should reside in the Web server, this way the network traffic will decrease considerably. For more information about design patterns check section 5.

This class also implements the business delegate pattern. This pattern is used to hide the underlying technology used in the model and separate it from the view allowing changing the technology used in the model without affecting the view. For example, if we decide not to use EJBs and use other framework such as Spring the clients won’t be affected by this change making the system easier to maintain.
The methods available to the client from the Interface are shown below:

```java
boolean addCourse(String code, String cName);
boolean addLanguage(String lanCode, String language);
void changePassword(String login, String password);
public boolean checkLogin(String login, String password);
public boolean checkPK(String login);
public AppUser createAppUser(String login, String password, String fName,
                             String lName, String about, String city,
                             String email, String pPicture, String quotes,
                             String state, String street, String telf,
                             String visibility, String country,
                             String[] courseList, String hUniv,
                             String[] languageList, String mayor);
Article createArticle(AppUser aAuthor, String aHeader, String aText,
                     String aUrl, String country, String course,
                     String hUniv, String language, String mayor);
Blog createBlog(AppUser appUser, String visibility);
BlogEntry createBlogEntry(String bId, String picture, String text,
                          String header);
Event createEvent(AppUser eAuthor, String eHeader, String eText,
                  String visibility, String country, String course,
                  String hUniv, String language, String mayor);
Link createLink(AppUser lAuthor, String lLink);
public void removeAppUser(String uId);
void removeArticle(String article);
void removeBlog(String bId);
void removeBlogEntry(String beId);
void removeCourse(String course);
void removeEvent(String event);
void removeLanguage(String lanCode);
void removeLink(String link);
List<Article> searchArticles(String param);
List<Event> searchEvents(String param);
List<Event> searchPublicEvents(String param);
List<AppUser> searchPublicUsers(String param);
List<AppUser> searchUsers(String param);
void updateAppUser(String login, String fName, String lName, String about,
                   String city, String email, String pPicture,
                   String quotes, String state, String street,
                   String telf, String visibility, String blogVis,
                   String country, String[] courseList, String hUniv,
                   String[] languageList, String mayor);
void updateArticle(String aId, String aHeader, String aText, String aUrl,
                   String country, String course, String hUniv,
                   String language, String mayor);
```
void updateBlogEntry(Long bId, String picture, String text, String header);
void updateEvent(String eId, String eHeader, String eText,
    String visibility, String country, String course,
    String hUniv, String language, String mayor);

public AppUser getAppUser(String uId);
public List<AppUser> getAppUsers();
Article getArticle(String aId);
List<Article> getArticles();
List<Article> getArticles(Profile profile);
Blog getBlog(String bId);
BlogEntry getBlogEntry(String beId);
List<Blog> getBlogs();
List<CountryCode> getCountries();
List<Course> getcourses();
Event getEvent(String eId);
List<Event> getEvents();
List<Event> getEvents(Profile profile);
List<Language> getLanguages();
List<AppUser> getPublicAppUsers();
List<Blog> getPublicBlogs();
List<Event> getPublicEvents();
List<State> getStates();
Blog getUserBlog(String uId);
Profile getUserProfile(String login);
List<BlogEntry> getUserBlogEntries(String uId);
5.3.1.2. Entity Beans

The entity beans and their relations represent the data model of the system and will maintain the same relations than tables in the relational database. The entity manager will manage the mapping.

Next figure shows the main relations between the entity beans:

The User is the center of the application. He/She may manage links, articles, events or the blog. The user has one profile associated that may have one country and several languages/courses. The filter is associated to events/articles and may only have one course/language.
In this section we are going to take a closer look to the classes in the model package. The test.model package just has one class that contains the unit test cases to test the façade methods. First, we are going to take a closer look on the EJBs structure and then show the most critical implementation classes.

In the next figure we show the structure that all the EJBs follow. To simplify the diagram we only show one session bean, but note that the rest of them have the same relations.

In the figure we can see the example of the article session bean. The session façade will look up the session bean and call one of the methods. To get the reference the session façade can call the remote or local interface. The session bean will use the Entity Manager to fetch the data from the relational database and use the list to store the results that will be returned.
Now we are going to show the actual implementation classes. Since there are many of them we will divide and show the classes in groups omitting some details.

- **AppUser:**
User Blog:
5.3.2. View Layer

The ViewController Module consists in the Web content and the managed beans from the view.backing package. Each JSP page will have one managed bean that will perform operations on it UI components and will also be in charge of calling the model.

Summarizing, a JSP page will use the JSF Tag Library and will have several JSF UI components included in the page as part of the view. The corresponding managed bean will have these components with their getters and setters methods. Every time that the page is compiled the managed bean is instantiated and the constructor and the setter methods are called. The developer can implement logic in the constructor to perform operations such page control access. He/She can also add logic to the setter methods to select whether or not render a specific component or to set a default value (e.g. when editing an item). When submitting information, the command button will execute an action from the managed bean that can use the getters to extract the information from the form to call the model.

Finally, there is another class which is not a managed bean. This class implements the transfer object design pattern and is used to transform the generic list that the model returns to the specific list that the view uses. Note, that we want to achieve independence between the model and the view, so the model has to use generic types because the view may be changed. This class will adapt the generic lists to the lists used in the JSF components such menus. The menus in the JSF use lists of SelectItems objects, the ModelConvert class transform the generic lists to SelectItem lists. This is the implementation of the transfer object pattern.
There are some objects such data tables that support generic types so it will be no necessary to use the model convert. Looking at the figure we can see that each JSP page has a managed bean that is in charge of calling the model. When the model returns the results they may be converted to *SelectItem* list if necessary using the model convert.
Session Check

As we mentioned before, every JSP page has a backing bean that contents the UI components. Another function of this bean is to check the access to the page and control the session. All managed beans will extend from the SessionCheck class that implements a method called checkSession() that checks two things:

- The user session did not expire: If so, the user will be redirected to a page that will inform him/her about it and will allow the user to log in again.
- The user has access to the page: If the session is valid, this method will check the user session information to see if the user has access to the page.

This method will be called before submitting any information or clicking to a link to verify the session didn’t expired. This method is also called in the constructor to check the access before the page is rendered.

The class will also have two attributes and the corresponding getters and setters. It will store the session and also the resourceBundle variable that contents the current language file and is used to get the messages to show to the user.

There exists a hierarchy in the SessionCheck class. There are two classes that extend the SessionCheck class and redefine the checkSession() method to verify that the user has access to the JSP page.

- AdminSessionCheck redefines the checkSession() method to verify that the user is the administrator. Administration beans will extend this class.
- RegisteredUserSessionCheck redefines the method to verify that the user is a registered user. Registered user backing beans will extend this class.
The managed beans that are for guest user will extend the *SessionCheck* class. This hierarchy is shown in the figure below:

In the diagram we included just one backing bean to keep it small. But all the beans used in registered user pages will extend the *RegisteredUserSessionCheck* class. Managed beans used by the administrator will extend *AdminSessionCheck* and managed beans shared by guest and registered users will extend *SessionCheck*. Remember, that the registered user actor extends the user actor.
5.3.3. Dynamic Behavior

In this section we will show the dynamic behavior of the application by showing two sequence diagrams of two of the critical use cases. Again, this sequence diagrams refine the ones shown in phase two adding more details. We will show the uses case “Register” and “Add Article” as examples, the behavior of the other uses cases will be very similar and will be omitted. In section 5, we will describe the algorithms used in the application and their behavior; for that purpose activity/state diagrams will be used instead of sequence diagrams.

5.3.3.1. Register User

Recall the sequence diagram for this use case from the architecture design:

Next, we show the same functionality implemented in the J2EE platform as a Web application with all implementation details.
In the figure the whole process is explained. The user click the link in the index page, since no action is performed the backing bean is not use and the request is handled by the front controller that redirect the request to the registration page. The Register Servlet is called and it will be in charge of generating the view. To do this task, first instantiates the managed bean of the register JSP page and this one will instantiate the UI components (JSF/ADF).

Then the Servlet is ready to generate the view and create the dynamic web content. The stereotype <<build>> represent the action performed by the Servlet when generation the HTML code. The page will include a HTML form where the user will introduce the data and click “submit”.

The action method of the managed bean will be called and it will check that all the information is valid before calling the model. In case that the input is wrong it will create a message using the message system provided by JSF and tell the Faces Servlet that there was an error by returning the string “failure”. The Faces Context will reload the page and the error message will be shown.

If the input is correct, the managed bean will gather all the information from the UI components and call the model by instantiating the model façade and calling the method register(). The model façade will perform the look-up of the AppUserSessionBean and instantiate the Session Bean. Then, it will call the registerUser() method that will create the AppUser Entity Bean and use the Entity Manager to perform the persistence in the relational database.

The model façade will then create the user’s profile by using the same procedure. Take the data that was gathered from the UI components and passed to the façade, call the model and let the session bean perform the operation using the Entity Manager. Note that in the previous diagram (architecture design) this was planned to be performed just once and have just one entity AppUser. But in the database design the user’s profile was separated from the entity user to increase the performance when implemented the filtering algorithm.
Finally, the results are sent to the managed bean. The managed bean will create a message to inform the user about the operation and return a string that will be used to forward to the next page.

The request is handled by the Faces Servlet that will find the next page and call the corresponding Servlet. The Servlet will instantiate the managed bean and UI components and create the view. The message will be shown on the next page.

As we can see, the diagram got too big comparing to the platform independent one; but this is due to the complex technology used. If you take a closer look to the problem domain and forget about implementation details, the functionality is the same as in the architecture design sequence diagram.

In the next sequence diagram we will try not to repeat the same process and omit technology dependent details such page generation that are always performed the same way. This way the diagram will be smaller.

5.3.3.2. Add Article
In this case we have omitted some details such as the UI components generation or the functionality that takes place when the user input is invalid. This is exactly the same than the previous diagram.

The Registered User clicks on the link “add article”, the request is handled by the Faces Servlet that redirects the user to the add_article.jsp page. The page generation logic takes place, the managed bean and the UI components are created and rendered by the Servlet that generates the dynamic content.

The user will fill the form and submit the information. The input will be validated by the managed bean and in case of error a message will be shown to the user. Note that some validations such input type or length is automatically perform by the JSF using validation tags.

If the inputs are correct, the managed bean calls the method uploadFile() from the ModelConvert class. As we mentioned before, this class implemented the transfer object pattern but it is also used to execute the method that uploads the file to the web server. To perform this operation the ADF input file component is used.

Then, the managed bean calls the model and creates the article and its corresponding filter information. The result is sent back to the managed bean that will create a message to inform the user about the result of the operation and then return a string that will be used by the Faces Servlet to redirect the user to a new page.

The rest of the system functions follow the same pattern than this two sequence diagrams. The behavior is described in the architecture design document. This document excludes the tedious implementation details to make it easier to read. Understanding these last two sequences it will be easy to understand the rest of the code.
6. Implementation Design

In this section we will discuss some implementation issues that were not discussed in the previous sections such as design patterns or the algorithms used.

6.1. Folder Structure

The figure below shows how the system is organized in folders:

As we know by now, the system has two modules: Model and ViewController. In the model we have the source code in the “src” folder that contains the two packages used in this module with their corresponding java classes. The model sub folder content the diagrams used in this document. The JavaDoc is also included, where we describe all the methods used in the model. Finally we have the compiled source code under “classes”.

58
In the ViewController module we have the source code in the “src” folder where we have the managed beans of the JSP pages. Under “classes” we have all compiled java files including the managed beans and the Servlets from the JSP pages. In the “public_html” folder we have the entire web content, this includes:

- JSP pages
- Java Script Files
- CSS Files
- Tag Libraries
- Images
- Configuration Files
- Properties Files (Language Files)

6.2. Model Layer

6.2.1. Design Patterns

In this section we will describe the design patterns used in the project. The intention is not to give a completely description of the patterns, the idea is to show the patterns used and why they were implemented. For more information check [11, 12, 19].

6.2.1.1. Architectural Patterns

Two architectural patterns are used in the application. We already talked about them.
**Layers Pattern:** This pattern represents the main architecture of the system. Basically, it tells us that we need to divide the system in logical layers that must be independent one of the others. This design pattern is essential in order to build scalable and reusable application. Each layer will have a different function.

- **PRESENTATION**

- **LOGIC**

- **DATABASE**

  - The presentation layer is the part of the application where all the code written for the UI resides. This layer represents the client tier in the 3-tier architecture, if the client is standalone this layer will represent all the classes used to create the interface. If the client is Web based this layer represent the pages or classes use to create the web pages that the user sees in the Web browser. In the case of the J2EE, the JSP pages are commonly used to generate dynamic Web pages, and are the main component in the presentation layer.
• The logic layer represents the business model and is in charge of implementing the main functions of the system. It fetches information from the presentation layer and processes the information, if necessary, it will ask the data layer to get some information. Here we will have the main classes of our application. This layer corresponds to the business tier in the 3-layer architecture; the middle server will have the code corresponding to this layer.

• The data layer will have the code necessary to connect to the database and create the objects used in the second layer. It represents the database layer in the 3-tier architecture.

The main issue here is how the layers communicate with each other. The idea is that one layer only communicates with the lower layer sending request, no bidirectional communication takes place. The upper layer sends a request to the lower layer and gets a response, but the upper layer doesn’t know how this operation was done, it only knows the public interface. So, the communication takes place only from the upper layers to the lower layers, it is not allowed to send a request from one lower layer to an upper layer. This approach provides a high level of abstraction and encapsulation allowing the developers to build scalable and reusable applications because we can change the data layer and the logic layer won’t be affected as long as the public interfaces are kept the same. This approach allows us to build reusable software been able to change different modules without affecting the others. For example, we can change the database and the logic layer will not be affected.
1. **Model-View-Controller (MVC):** This pattern tells us how the three layers communicate with each other:

- The model handles application and business logic.
- The view handles presentation logic showing the data to the user.
- The controller accepts and interprets keyboard and mouse input.

This pattern is implemented by the Java Server Faces.

### 6.2.1.2. Design Patterns

Several design patterns were used during the development of this application; we already mentioned them and on this section we are going to summarize them. For more information visit [19].

The Model Façade class implements the session façade and business delegate patterns to help reusability and maintainability of the system by having one entry point to the model that hides the technology used. It also implements the *service locator* pattern that is used to find resources in the system; in this case the resources are the EJBs.

The Model Convert class implements the transfer object pattern to transform the list of entity beans that are returned from the model to lists of *SelectItem* used by the JSF in the view, adapting the object to the client.
Other pattern could have been used such the *page-by-page iterator* or the *fast-lane reader*. The page-by-page iterator was not used because we don’t expect to have so large amount of data. The fast lane reader is not necessary since EJB 3.0 solves the problems of EJB 2.x that make reasonable use this pattern.

### 6.2.2. Algorithms

#### 6.2.2.1. Search Algorithm

The search algorithm is quite simple; it just takes all the items from the database and compares their fields to the search string. If one of the field match the item is added to the result and the other fields are not checked.

The search could be performed in the database but this option was not chosen for two reasons:

- We don’t want to overload the database since is executed in one server whereas the EJBs are in different machines. So, we move the logic from the database to the model to increase performance and take advantage of the distribute system.

- When searching the database there are always problems with some symbols and also it doesn’t differentiate upper case from lower case.

Note that the search can be used to find information that is not available when browsing the items due to the filter algorithm. Usually, the user will not be interested in some themes but if for any reason the user is looking for a particular item that doesn’t match his/her profile, the search function can be used to find it.

The Access Control System will also check that a guest doesn’t receive private data using the search function.
6.2.2.2. Filter Algorithm

This is one of the key components of the system. One of the critical goals of the system besides support many languages or follow rigorous design principles was to offer personalized information. User will see what they are interested in seeing. Some attributes will be use to filter information and show the user only a subset of the items.

Some changes were made in the implementation that makes the filtering algorithm less restrictive. Basically, the user who publishes an item (event/article) will decide the target users.

The filter attributes are:

- Country
- Courses
- Languages
- Home University
- Mayor

As a reminder the user has associated a profile and the events/articles a filer. Profile and filter have the same attributes but the profile can have several languages and courses and the filter only one. This means, that as an user I may want to receive information in several languages but when I publish something it will be only in one language.

When a user registers in the application a profile is created and it may have:

- Zero or more languages
- Zero or more courses
- Zero or one country
- Zero or one mayor
- Zero or one Mayor
This is saved in the database and will be used when browsing events or articles to get personalized information.

When the user publish and item a filter is created that may have:

- Zero or one language
- Zero or one course
- Zero or one country
- Zero or one mayor
- Zero or one Mayor

Check the Data Model document for more information.

The filter is lead by the items which means that the profile is not restrictive. When the user publish an item it can leave the filtering information blank, in this case all the users will receive the information no matter what is in the profile.

If filter information is used in the item, that information will be compare to the user profile. For example, if we say that the article is in Spanish, when a user clicks on “browse articles” the system will compare that article to the user profile. If the user has Spanish in his/her profile the article will be shown, in any other case the article will not be shown. The same behavior is followed by the other attributes.

So, if the item has one attribute null we don’t even compare to the profile to increase performance. Because this will mean that the author doesn't care about that attribute and doesn’t want to filter using that particular attribute.

The filter information is only used for registered users. Guest can only see public items and no filtering is done. The user should not fill the filter information if he/she set public visibility making it available to everyone.
When the user click on browse items (events/articles) the algorithm is executed:

1. The system receives the method call with the user’s profile as parameter.
2. The system fetches the list of items from the database using the entity manager.
3. The System creates an empty result list.
4. For each Item:
   4.1. If it is an event and is public the item is added to the list and the loop changes to the next item and no more attributes are checked
   4.2. If the author is the current user the item is added to the list and the loop changes to the next item and no more attributes are checked. Note that, the user’s item will have two links: edit and remove.
   4.3. If not, a variable `check` is initialized to zero and for each filter attribute:
      4.3.1. Compare profile attribute to filter attribute
         4.3.1.1. If filter attribute null check++ (no comparison is performed)
         4.3.1.2. If matches (ignoring case) check++
   4.4. If check equals 5 (all attributes match to the profile or are null) then we add the item to the result.
Next figure shows this process:

1. Get User Profile
2. Get Item List
3. Create Empty List
4. Select Item
5. Is User the Author?
   - Yes: Add To Result List
   - No: Select Attribute
6. Check = 0
7. Select Attribute
   - Null?:
     - Yes: Check ++
     - No: More Attributes?
9. More Attributes?
   - Yes: More Items?
   - No: Select Attribute
10. Is Event?
   - Yes: Public?
      - Yes: Add To Result List
      - No: Select Attribute
   - No: Select Attribute
11. More Items?
   - Yes: More Attributes?
   - No: Select Attribute
12. Check == 5?
   - Yes: Add To Result List
   - No: Select Attribute
6.3. View Layer

This section discusses some issues related to the view, especially the JSF. Note, that this project focused more on the model and not so much on the view. The goal is to show how to build an enterprise web application using the J2EE platform and not to how to design web sites. The important aspect is the architecture and not so much the presentation. However, the application was developed separating the view and the model, so a web designer can take the JSP pages and change the layout without affecting the Java code.

But there are some aspects that are not so related to web design that are very important. The most important one is the internationalization and input validation mechanisms that the JSF provides.

6.3.1. Internationalization

The application was built to support different languages taking full advantage of the JSF. The languages are stored in the databases and the administrator can add them. To do this, the administrator first needs to add the language to the list of supported languages in the faces configuration file (faces-config.xml).

```
<locale-config>
  <default-locale>en</default-locale>
  <supported-locale>es</supported-locale>
</locale-config>
```

The `locate-config` command is used to select the supported languages. In this case we have English as default and Spanish as one of the supported languages.

Once, we added the language we have to log in the application as administrator and add the language to the database. And finally, we need to add the language file (`message_es.properties`) to the `/public_html/lib/classes` directory. The application behavior is described next:
1. The user enters the application URL in the browser.
2. The application checks the Web browser default language.
3. If supported the page is shown in that language.
4. If not supported the application looks the next supported language by the Web browser.
5. If none of the Web browser languages is supported by the application the default language (English) is shown.
In the main page the user can manually change the language. This can be useful when a user uses another computer that doesn’t have his/her native language.

6.3.2. Input Validation

JSF validation features are used in the application. JSF provides a tag library called JSF Core that provides tags to convert and validate inputs which are typical operations performed in web applications. It also lets the developer create new validation tags.

We just used the standard libraries to check the length of the fields or the type of input. For example, we use validation tags to check that the password is no less than 4 characters long or than the phone number contents numbers and no letters. We also check that all the required fields are filled before submitting the form.

Standard error messages are shown to the user in any of the supported languages.

Also, access to pages is controlled by implementing mechanisms to check if a particular user has permission to access a given page.

6.3.3. JavaScript

JavaScript is used to implement an encryption algorithm to send the password encrypted throughout the network. A public encryption library is used.

JSF validation tags are used instead of JavaScript to be able to personalize the error message supporting several languages.

6.3.4. CSS

Cascade Sheets were used to change the style of the web page.
6.4. Tools

The following tools were used to develop the application:

2. *Smart Draw 7*: Diagrams for documentation.
3. *JDeveloper 10g*: Main IDE. Used for development (coding, testing, deployment) and also to create class diagrams.