

```

1 <html:javascript formName="authorForm" />
2
3 <html:form action="/admin/author/save.do" enctype="multipart/form-data"
4   onsubmit="return validateAuthorForm(this);">
5
6   <logic:present name="author">
7     <html:hidden name="author" property="authorId" />
8   </logic:present>
9   <logic:notPresent name="author">
10    <html:hidden property="authorId" value="" />
11  </logic:notPresent>
12  <table width="100%" height="500px">
13    <tr height="5px">
14      <td colspan="3" align="right" width="100%><html:img
15        page="/img/return.gif"></html:img></td>
16    </tr>
17    <tr height="490px">
18      <td width="15%">></td>
19
20      <td>
21        <table width="70%" align="center" cellspacing="2" CELLPADDING="4">
22          <tr>
23            <td class="cellLabel" colspan="4">></td>
24
25            </tr>
26            <html:errors property="authorExist" />
27          <tr>
28            <td class="cellLabel" colspan="1" width="15%>
29              <bean:message key="author.username" /></td>
30            <td class="cellText" colspan="3" width="85%" align="left">
31              <logic:present name="author">
32                <html:text name="author" property="username" size="35"/>
33              </logic:present>
34              <logic:notPresent name="author">
35                <html:text property="username" size="35" maxlength="50" />
36              </logic:notPresent></td>
37            </tr>
38            <tr>
39              <td class="cellLabel" width="15%" align="left">
40                <bean:message key="author.firstName" /></td>
41              <td class="cellText" width="35%" align="left">
42                <logic:present name="author">
43                  <html:text name="author" property="firstName" size="35"/>
44                </logic:present>
45                <logic:notPresent name="author">
46                  <html:text property="firstName" size="35" maxlength="50" />
47                </logic:notPresent></td>
48              <td class="cellLabel" width="15%" align="left">
49                <bean:message key="author.lastName" /></td>
50              <td class="cellText" width="35%" align="left">
51                <logic:present name="author">
52                  <html:text name="author" property="lastName" size="35" />
53                </logic:present>
54                <logic:notPresent name="author">
55                  <html:text property="lastName" size="35" maxlength="50" />
56                </logic:notPresent></td>
57            </tr>
58            <tr>
59              <logic:present name="author">
60                <td class="cellLabel" width="15%" align="left">
61                  <bean:message key="author.isMember" /></td>

```

Figura 40: Archivo author.jsp(I)

```

62      <td class="cellText" width="35%" align="left">
63          <html:checkbox name="author" property="isMember" /></td>
64      <td class="cellLabel" width="15%" align="left">
65          <bean:message key="author.isPhd" /></td>
66      <td class="cellText" width="35%" align="left">
67          <html:checkbox name="author" property="isPhd" /></td>
68      </logic:present>
69      <logic:notPresent name="author">
70          <td class="cellLabel" width="15%" align="left">
71              <bean:message key="author.isMember" /></td>
72          <td class="cellText" width="35%" align="left">
73              <html:checkbox property="isMember" value="0" /></td>
74          <td class="cellLabel" width="15%" align="left">
75              <bean:message key="author.isPhd" /></td>
76          <td class="cellText" width="35%" align="left">
77              <html:checkbox property="isPhd" value="0" /></td>
78      </logic:notPresent>
79  </tr>
80
81  <tr>
82
83      <td class="cellText" colspan="4" align="right">
84          <table width="25%">
85              <tr>
86                  <td class="cellText" align="right"><html:submit
87                      styleClass="button">
88                          <bean:message key="button.save" />
89                  </html:submit></td>
90                  <td class="cellText" align="right"><html:cancel
91                      styleClass="button">
92                          <bean:message key="button.cancel" />
93                  </html:cancel></td>
94              </tr>
95          </table>
96      </td>
97  </tr>
98  <tr>
99      <td class="cellLabel" colspan="4"></td>
100     </tr>
101     </table>
102     </td>
103     <td width="15%"></td>
104 </tr>
105 <tr height="5px">
106     <td colspan="3" align="left" width="100%><html:link
107         styleClass="link" action="/admin/author/list.do?action=load">
108             <html:img border="0" pageKey="images.goBack"></html:img>
109         </html:link></td>
110     </tr>
111 </table>
112</html:form>

```

Figura 41: Archivo author.jsp(II)

En la línea 1, la etiqueta `<html:javascript formName=authorForm>` es un tag que pertenece a la colección de tags html del package `org.apache.struts.taglib`. Con la inclusión de esta etiqueta indicamos que el formulario asociado a esta

página jsp se llama *authorForm*. En uno de nuestros archivos de configuración existe la declaración de una clase que extiende a la clase **DynaValidationForm** a la que hemos identificado con el nombre de *authorForm*.

La etiqueta `<html:form>` incluye la llamada a la función javascript `validateAuthorForm()`. Esta llamada se realiza cuando el usuario hace un *submit* del formulario. Esta función comprueba que los datos introducidos por el usuario sean válidos. Si lo son, se invoca el método `execute()` de la instancia de la Action incluida en el atributo `action` del tag (tras pasar por el proceso de validación en el servidor que ilustramos en la figura 26). Para definir las reglas de integridad que queremos aplicar sobre los campos del formulario, tenemos unos archivos xml dónde incluimos un conjunto de funciones javascript que se invocan según especificamos. Estos archivos son **validation.xml** y **validation-rules.xml**. Hemos añadido y comentado un fragmento de ambos más adelante en este mismo apartado.

Siguiente con nuestra página jsp del formulario de alta de un autor, la etiqueta `<logic:present>` en la línea 6, checkea si el bean con nombre *author* está presente. Este tag, perteneciente a la colección de tags `logic` del package `org.apache.struts.taglib` admite el atributo `scope` donde indicamos si el bean ha de estar presente en el *request*, en la *session* ó en la aplicación. Si el atributo no está especificado, busca en todos los *scopes*.

El tag `<html:img>` incluye una imagen. El atributo `pageKey` contiene el nombre por el que se conoce esa imagen. En un fichero de propiedades guardamos los *paths* de nuestras imágenes asociándoles un nombre que las identifique. De esta manera, si en un futuro necesitamos cambiar las imágenes ó el *path* de las mismas, todos los cambios estarán centralizados en este archivo.

El tag `<html:errors>` se utiliza para recuperar los errores que en el lado del servidor se hayan podido encontrar. Desde el servidor, se genera una instancia de la clase `org.apache.struts.action.ActionErrors` y se mete en el objeto `Request` que identifica a esta petición. Esta etiqueta comprueba si existe una instancia de la clase `org.apache.struts.action.ActionErrors` y si existe, muestra un mensaje de error. El contenido de este mensaje está asociado al identificador *authorExist* que se carga desde un fichero de propiedades.

En la línea 26, el tag `<bean:message key=author.username >` pertenece a la colección de tags `bean` y lo utilizamos para recuperar del fichero de propiedades el mensaje que queremos enseñar al usuario para indicarle en qué campo ha de introducir el username del autor. Este mensaje variará según el idioma predefinido en el navegador, igual que en los mensajes del resto de la aplicación.

El atributo `name` del tag `<html:text name=author property=username>`

`size=35 maxlength=50>` en la línea 29 contiene el nombre del bean del que queremos recuperar alguna propiedad. La propiedad del bean *author* que queremos es *username*, especificado con el atributo *property*.

Con los tags de la colección `html` declaramos dos botones, uno para salvar el autor introducido y otro para cancelar la operación. El primero lo obtenemos con el tag `<html:submit styleClass=button>` en la línea 73. El segundo con `<html:cancel styleClass=button >`, una línea más abajo. Cuando el usuario pulsa el botón de cancelar, el submit del formulario no se produce.

En la línea 88, el tag `<html:link>` contienen el atributo *action* que contiene el nombre de la Action sobre la cual se invocará el método *execute()* cuando el usuario haga un click sobre ese link. Esta Action es especial, si observamos el nombre podemos ver como tras el nombre que la identifica (`/admin/author/list.do`) hay un parámetro llamado *action* que se inicializa con el valor de *load*. Existe la clase `org.apache.struts.actions.LookupDispatchAction` que extiende a la clase Action. La Action de este ejemplo extiende a esa clase. Tienen un comportamiento específico pero nosotros aquí no entraremos en detalle. Si el lector lo desea puede consultar alguno de los documentos que listamos en la bibliografía [4],[5].

Para acabar, mostramos el formulario resultante de éste código.

Usuario	<input type="text"/>		
Nombre	<input type="text"/>	Apellido	<input type="text"/>
DAC	<input type="checkbox"/>	Doctorando	<input checked="" type="checkbox"/>
<input type="button" value="Guardar"/> <input type="button" value="Cancelar"/>			

Figura 42: Formulario de alta ó modificación de un autor e investigador

Antes hemos explicado brevemente el tag `<html:errors>`. Hemos dicho que el mensaje de error que muestra este tag está guardado en un fichero de propiedades ó *messages resource*, que es como mayormente se le conoce. También los tags `<bean:message key=author.username>` ó `<html:img>` utilizan este fichero de propiedades para recuperar mensajes ó *paths* de las imágenes. El código mostrado a continuación es un fragmento de ese fichero de propiedades ó *messages resource* donde podemos ver qué sintaxis utiliza.

```

author.username=Usuari
author.firstName=Nom
author.lastName=Cognom
author.isPhd=Doctorand
author.isMember=DAC

button.new = Nou
button.del = Esborrar
button.save = Guardar

menu.maintenance.languaje=Idiomes
menu.maintenance.institution=Institucions

menu.project=Projectes
menu.trip=Viatges

error.login=<tr>
            <td colspan="2" align="center" class="error">Usuari ó contrasenya
incorrectes</td>
        </tr>
error.author.exist=<tr>
            <td colspan="4" align="center" class="error">
                Ja existeix un autor amb aquest username</td></tr>
error.user.exist=<tr><td colspan="6" align="center" class="error">
                Ja existeix un user amb aquest username</td></tr>

error.author.notunique=<tr><td align="center" class="error" colspan="6">
                No existeix un autor amb aquestes dades ó n'hi ha més d'un
autor vàlid
            </td></tr>
errors.required={0} és un camp obligatori

image.goBack=/img/return.gif

```

Figura 43: Archivo Application.resources\_CA

Para poder incluir el código en este documento y facilitar la lectura hemos introducido unos saltos del líne en los mensajes de errores, pero la sintaxis es la siguiente: cada líne contiene un mensaje. El formato es **identificador=mensaje**, donde identificador puede estar separado por puntos y es el utilizado desde las páginas jsp para recuperar el mensaje.

También hemos dicho que la función javascript *validateAuthorForm()* incluida en el tag <html:form> se llama cuando el usuario hace un submit del formulario y que ésta realiza unas validaciones para comprobar si los datos son correctos. Las reglas de validación de los campos se muestran a continuación.

El fichero **validator.xml** contiene qué campos de cada formulario hay

que comprobar y con qué reglas.

```

1 <!-- author -->
2   <formset>
3     <form name="authorForm">
4       <field property="username" depends="required">
5         <arg0 key="author.username" />
6       </field>
7       <field property="firstName" depends="required">
8         <arg0 key="author.firstName" />
9       </field>
10      <field property="lastName" depends="required">
11        <arg0 key="author.lastName" />
12      </field>
13    </form>
14  </formset>
15
16 <!-- conference paper -->
17 <formset>
18   <form name="conferencePaperForm">
19     <field property="title" depends="required">
20       <arg0 key="paper.title" />
21     </field>
22
23     <field property="yearPubAux" depends="required,integer,intRange">
24       <arg0 key="common.yearPub"/>
25       <arg1
26         name="intRange"
27         key="${var:min}"
28         resource="false"/>
29       <arg2
30         name="intRange"
31         key="${var:max}"
32         resource="false"/>
33       <var>
34         <var-name>min</var-name>
35         <var-value>1975</var-value>
36       </var>
37       <var>
38         <var-name>max</var-name>
39         <var-value>2050</var-value>
40       </var>
41     </field>
42
43     <field property="pagBeg" depends="required,integer">
44       <arg0 key="common.pagBeg" />
45     </field>
46
47     <field property="pagEnd" depends="required,integer">
48       <arg0 key="common.pagEnd" />
49     </field>
50
51   </form>
52 </formset>
```

Figura 44: Archivo validator.eps

Para cada formulario se crea un tag xml que contiene un atributo llamado `name` donde indicamos el nombre del formulario que queremos validar. Este

nombre se define en el fichero de configuración de formularios y acciones que explicamos en esta sección, en el apartado del **Controlador**(el fichero **struts-author.config.xml**).

Los subelementos del tag xml contienen los campos del formulario que queremos validar. El atributo **property** contiene el nombre del campo del formulario que queremos validar y el atributo **dependes** indica el nombre de la regla que utilizamos para la validación, puede haber más de una regla por campo.

El fichero **validator-rules.xml** contiene las reglas que se aplican para cada campo y las funciones javascript correspondientes.

Este archivo contiene tags xml llamados **<validator>**. Cada uno de estos tags representa una regla de validación. El atributo **name** contiene el nombre de la regla. Este nombre es el referenciado en el archivo **validator.xml** en el atributo **dependes** del tag **<form>**. Del resto de elementos nos interesa para este apartado la función javascript. Son éstas las funciones llamadas cuando el usuario hace un *submit* del formulario.

```

<validator name="required"
    classname="org.apache.struts.validator.FieldChecks"
    method="validateRequired"
    methodParams="java.lang.Object,
        org.apache.commons.validator.ValidatorAction,
        org.apache.commons.validator.Field,
        org.apache.struts.action.ActionErrors,
        javax.servlet.http.HttpServletRequest"
    msg="errors.required">
    <javascript><![CDATA[
        function validateRequired(form) {
            var isValid = true;
            var focusField = null;
            var i = 0;
            var fields = new Array();
            oRequired = new required();
            for (x in oRequired) {
                var field = form[oRequired[x][0]];

                if (field.type == 'text' ||
                    field.type == 'textarea' ||
                    field.type == 'file' ||
                    field.type == 'select-one' ||
                    field.type == 'radio' ||
                    field.type == 'password') {

                    var value = '';
                    // get field's value
                    if (field.type == "select-one") {
                        var si = field.selectedIndex;
                        if (si >= 0) {
                            value = field.options[si].value;
                        }
                    } else {
                        value = field.value;
                    }

                    if (trim(value).length == 0) {

                        if (i == 0) {
                            focusField = field;
                        }
                        fields[i++] = oRequired[x][1];
                        isValid = false;
                    }
                }
            }
            if (fields.length > 0) {
                focusField.focus();
                alert(fields.join('\n'));
            }
            return isValid;
        }

        // Trim whitespace from left and right sides of s.
        function trim(s) {
            return s.replace( /\s*/ , "" ).replace( /\s*$/ , "" );
        }
    ]]>
</javascript>

```

Figura 45: Archivo validator-rules.eps(I)

```
</validator>

<validator name="intRange"
    classname="org.apache.struts.validator.FieldChecks"
    method="validateIntRange"
    methodParams="java.lang.Object,
        org.apache.commons.validator.ValidatorAction,
        org.apache.commons.validator.Field,
        org.apache.struts.action.ActionErrors,
        javax.servlet.http.HttpServletRequest"
    depends="integer"
    msg="errors.range">

    <javascript><![CDATA[
function validateIntRange(form) {
    var isValid = true;
    var focusField = null;
    var i = 0;
    var fields = new Array();
    oRange = new intRange();
    for (x in oRange) {
        var field = form[oRange[x][0]];

        if ((field.type == 'text' ||
            field.type == 'textarea') &&
            (field.value.length > 0)) {

            var iMin = parseInt(oRange[x][2]("min"));
            var iMax = parseInt(oRange[x][2]("max"));
            var iValue = parseInt(field.value);
            if (! (iValue >= iMin && iValue <= iMax)) {
                if (i == 0) {
                    focusField = field;
                }
                fields[i++] = oRange[x][1];
                isValid = false;
            }
        }
    }
    if (fields.length > 0) {
        focusField.focus();
        alert(fields.join('\n'));
    }
    return isValid;
}
]]>
    </javascript>
</validator>
```

Figura 46: Archivo validator-rules.eps(II)

Para acabar, como ya hemos comentado, utilizamos el componente *Tiles* para crear nuestra propia plantilla jsp y no tener que repetir código en nuestras páginas web.

El archivo layout.jsp contiene una tabla html. Esta tabla tiene cuatro filas y en cada una de las filas hay un tag xml del tipo <tiles:insert>. Con éste tag indicamos qué contenido jsp ha de incluirse. El atributo **attribute** contiene un identificador del contenido jsp. El atributo **ignore** es para indicar si este contenido puede estar vacío.

Como ya hemos dicho, el archivo **layout.jsp** contiene la plantilla que vamos a utilizar para casi todas nuestras páginas jsp. Por ejemplo, el identificador **menu** estará siempre asociado a una página jps que contiene el menú de la aplicación que siempre está visible. Imaginémonos por un momento que no utilizamos esta plantilla. Tendríamos que replicar el código del menú en cada pantalla jsp. Si cambiáramos un link del menú, tendríamos que modificar todas las pantallas.

```
1 <%@page pageEncoding="UTF-8" %>
2 <%@taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
3 <%@taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
4 <%@taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
5 <%@taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
6 <%@taglib uri="/WEB-INF/struts-layout.tld" prefix="layout" %>
7
8 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
9 <html:html locale="true">
10 <head>
11 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
12 <script type="text/javascript" src="/hpc/js/javascript.js"></script>
13 <script type="text/javascript" src="/hpc/js/array.js"></script>
14 <link href="
```

Figura 47: Archivo layout.jsp

El mapeo de los identificadores con las pantallas jsp se hace en otro fichero de configuración. El framework de struts nos permite crear tantos ficheros de configuración como necesitemos. Poder hacerlo es necesario. Si pensamos en la cantidad de páginas jsp que contiene nuestra aplicación, es evidente que poder agruparlas nos facilitará el mantenimiento y el orden en gene-

ral. A continuación, mostramos un fragmento del fichero de configuración **tiles-author-defs.xml** que es dónde mapeamos los identificadores con las pantallas jsp.

```
<tiles-definitions>

    <definition name="authorList" extends="main">
        <put name="body" value="/WEB-INF/jsp/author/authorList.jsp" />
    </definition>

    <definition name="author" extends="main">
        <put name="body" value="/WEB-INF/jsp/author/author.jsp" />
    </definition>

</tiles-definitions>
```

Figura 48: Archivo tiles-author-defs.xml

Cada tag xml de tipo **definition** representa un conjunto de archivos jsp que se incluyen en una página ó pantalla. Si recordamos el archivo **layout.jsp**, incluía una tabla html y en cada fila un tag xml **<tiles:insert>** que se identificaba por el atributo **attribute**. Ese identificador es lo que se utiliza en los ficheros de configuración para indicar qué archivo jsp ocupará su lugar en el archivo **layout.jsp**. También podemos extender ó heredar de una **tile**. En este caso, éstas tiles heredan de la tile **main**, incluida en el fichero **tiles-defs.xml** que mostramos a continuación. La herencia se indica en el atributo **extends**.

```
<tiles-definitions>

<!-- ===== -->
<!-- Master definition -->
<!-- ===== -->

<!-- Doc index page description -->

<definition name="main" path="/WEB-INF/jsp/layout/layout.jsp">
    <put name="title" value="HPC" />
    <put name="head" value="/WEB-INF/jsp/header.jsp" />
    <put name="menu" value="/WEB-INF/jsp/menu.jsp" />
    <put name="footer" value="/WEB-INF/jsp/footer.jsp" />
</definition>

<definition name="tableList" extends="main">
```

Figura 49: Archivo tiles-defs.xml

## 5.2. El Controlador

Como explicamos en la sección de diseño, nuestro controlador está formado principalmente por un conjunto de servlets que extienden de la clase `org.apache.struts.action.Action`. Cuando un usuario hace un submit del formulario en el que se encuentra, uno de estos Servlets se invoca y gestiona la petición.

Siguiendo con el ejemplo del formulario de alta ó modificación de un autor ó investigador, adjuntamos el código del Servlet que gestiona la petición. Lo comentamos después.

## Archivo AuthorSaveAction.java

```

1 package hpc.action.author;
2
3 import hpc.database.AuthorBean;
4 import hpc.database.AuthorManager;
5 import hpc.util.HpcException;
6
7
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10
11 import org.apache.struts.action.Action;
12 import org.apache.struts.action.ActionError;
13 import org.apache.struts.action.ActionErrors;
14 import org.apache.struts.action.ActionForm;
15 import org.apache.struts.action.DynaActionForm;
16 import org.apache.struts.action.ActionForward;
17 import org.apache.struts.action.ActionMapping;
18 import org.apache.struts.action.DynaActionFormClass;
19 import org.apache.struts.config.FormBeanConfig;
20 import org.apache.commons.beanutils.*;
21
22
23
24 public class AuthorSaveAction extends Action{
25
26 /**
27 * After save Author, this function is called. It puts needed filters with author
28 data.
29 * Author's List will only show him
30 * @param author is the AuthorBean instance with author's data
31 * @param request The HTTP request we are processing
32 */
33 private void setFilters(HttpServletRequest request,AuthorBean author) throws
Exception{
34     FormBeanConfig fbc = new FormBeanConfig();
35     fbc.setName("authorListForm");
36     DynaActionFormClass dafc = DynaActionFormClass.createDynaActionFormClass(fbc);
37     DynaActionForm form = (DynaActionForm) dafc.newInstance();
38
39     form.set("username",author.getUsername());
40
41     request.setAttribute("authorListForm", form);
42 }
43
44 /**
45 * Creates an author
46 * Process the specified HTTP request, and create the corresponding HTTP
47 * response (or forward to another web component that will create it).
48 * Return an <code>ActionForward</code> instance describing where and how
49 * control should be forwarded, or <code>null</code> if the response has
50 * already been completed.
51 *
52 * @param mapping The ActionMapping used to select this instance
53 * @param form The optional ActionForm bean for this request (if any)
54 * @param request The HTTP request we are processing
55 * @param response The HTTP response we are creating
56 *
57 * @return Action to forward to /author/list.do

```

Figura 50: Archivo AuthorSaveAction.java

```

58  * @exception Exception if an input/output error or servlet exception occurs
59  */
60 public ActionForward execute(
61     ActionMapping mapping,
62     ActionForm form,
63     HttpServletRequest request,
64     HttpServletResponse response)
65 throws Exception {
66     if(!isCancelled(request)){
67         // Manages checkbox properties
68         String isPhd = (String)((DynaActionForm)form).get("isPhd");
69         if(!isPhd.equalsIgnoreCase("0"))
70             ((DynaActionForm)form).set("isPhd", "1");
71         String isMember = (String)((DynaActionForm)form).get("isMember");
72         if(!isMember.equalsIgnoreCase("0"))
73             ((DynaActionForm)form).set("isMember", "1");
74
75         //Copy properties
76         AuthorBean author = new AuthorBean();
77         BeanUtils.copyProperties(author,form);
78
79     try{
80
81         //Saves the author's data
82         AuthorManager.getInstance().save(author);
83
84         // Filters the author list with the new author
85         setFilters(request,author);
86
87     }
88     catch (HpcException e) {
89         getServlet().log("Connection.close", e);
90         if(e.getErrorCode() == HpcException.ER_DUP_ENTRY){
91             ActionErrors errors = new ActionErrors();
92             errors.add("authorExist",new ActionError("error.author.exist"));
93             saveErrors(request, errors);
94         }
95         else{//unknow exception
96             throw e;
97         }
98     }
99 }
100 }
101 return mapping.findForward("success");
102 }
103}
104

```

Figura 51: Archivo AuthorSaveAction.java(II)

El método `execute()` es el método invocado. Desde el formulario de alta ó modificación de un autor, el usuario puede querer grabar los datos que ha introducido en el formulario ó cancelar y volver al listado de autores(que es la pantalla previa a ésta en la aplicación). En la línea 66 se comprueba si el usuario a clicado el botón de *Guardar* ó el de *Cancelar* con el método

`isCancelled()` de la clase Action. Si el usuario ha cancelado la operación no hemos de hacer nada. Si el usuario quiere guardar la información que ha introducido, nos disponemos a guardar la información de un autor.

En la línea 60 tenemos la declaración del método `execute()`. En esta declaración podemos ver que uno de los parámetros que recibe es un objeto `org.apache.struts.ActionForm`. Recordemos que tal y como explicamos en la sección de Diseño, estos objetos contienen la información introducida por el usuario en el formulario. Recordemos también que nosotros utilizamos objetos `org.apache.struts.DynaActionForm`, que extienden de la clase `org.apache.struts.ActionForm`. En la línea 38 recuperamos el valor del checkbox que utiliza el usuario para indicar si un autor es doctor ó no.

En la línea 77 inicializamos un objeto `hpc.database.AuthorBean` con los datos del formulario utilizando el método `copyProperties()` de la clase `org.apache.commons.beanutils.BeanUtils`.

En la línea 83 se guarda en la base de datos al autor introducido. En este punto de la aplicación desconocemos si se ha dado de alta un nuevo autor ó ha sido modificado uno existente. Es la clase `hpc.database.AuthorManager` quien se ocupa de esto.

La línea 86 prepara el regreso al formulario que contiene la lista de los autores.

El método `setFilters()` crea una instancia de la clase `org.apache.struts.DynaActionForm` y la inicializa con el valor del campo `username`. Este valor es único. La instancia de la clase `org.apache.struts.DynaActionForm` se introduce en el objeto `request` que se recuperará desde el método `execute()` del próximo servlet que gestione el listado de autores. Éste servlet mira si existe una instancia de `DynaActionForm` con identificador `authorListForm` en el objeto `Request`. Si existe, utiliza el valor del campo `username` para filtrar la lista de autores por aquellos que contengan ese `username`. De esta manera, en el listado de autores aparecerá el autor modificado ó dado de alta.

Un autor ha de tener un `username` único. Esta restricción está controlada por el SGBD y en el momento de la insercción, el SGBD puede lanzar una excepción si ya existe un autor con el `username` introducido por el usuario. En la línea 89 recogemos una excepción de tipo `hpc.database.HpcException`. Esta clase es nuestra y extiende de la clase `java.lang.Exception`. Cuando desde el método `save()` de la clase `hpc.database.AuthorManager` se captura una `java.sql.SQLException`, una instancia de `HpcException` es creada y lanzada hacia arriba. Desde el `catch` de la línea 89 comprobamos si la excepción es debida a la restricción del `username` y si lo es, introducimos en el objeto `Request` una instancia de la clase `org.apache.struts.action.ActionErrors` que será recogida desde la página JSP.

## Autentificación

En la sección de diseño explicábamos cómo habíamos decidido gestionar el proceso de autentificación en nuestra aplicación. A continuación exponemos el código de la clase `hpc.security.AuthenticationManager`, una de las principales clases que gestiona este proceso y lo comentamos.

```

1 package hpc.security;
2
3 import java.io.FileInputStream;
4 import java.util.Properties;
5 import hpc.security.Jcrypt;
6
7 public class AuthenticationManager {
8
9     private static AuthenticationManager manager_instance = new AuthenticationManager();
10    //Contains role mappings.
11    private Properties pwdMappings;
12
13    /**Load mappings from a properties file on the file system.*/
14    public AuthenticationManager() {
15        //Read in properties file containing role mappings...
16        this.pwdMappings = new Properties();
17        try {
18            this.pwdMappings.load(new FileInputStream(System.getProperty("file.separator") + "home" +
19                System.getProperty("file.separator") + "zeta" +
20                System.getProperty("file.separator") + "PFC" +
21                System.getProperty("file.separator") + "hpc" +
22                System.getProperty("file.separator") + "shadow"));
23        }
24        catch (Exception e) {
25            throw new RuntimeException(e);
26        }
27    }
28
29    /**
30     * Returns the AuthenticationManager singleton instance.
31     */
32    public static AuthenticationManager getInstance()
33    {
34        return manager_instance;
35    }
36
37    /**Returns boolean indicating whether user has been correctly
38     * authentified
39     */
40    public boolean isUserAuthentified(String username, String password) {
41
42        boolean authentified = false;
43
44        String pwd = (String)pwdMappings.get(username);
45        if(pwd != null){
46            if(pwd.equals(Jcrypt.crypt(pwd.substring(0,2), password)))
47                authentified = true;
48        }
49        return authentified;
50    }
51 }
```

Figura 52: Archivo Authentication.java

Esta clase es una clase *singleton*. En la línea 14, el constructor de la clase,

lee el fichero de passwords y lo carga en memoria.

La clase hpc.security.Jcrypt, importada en la línea 5, es la clase que contiene la implementación del algoritmo de encriptación con el que los passwords han sido encriptados.

El método `isUserAuthentified()` en la línea 40, es llamado con el username y el password del usuario que está intentando logarse en la aplicación. Con el username recuperamos el password original del atributo `pwdMappings`. En la línea 46 encriptamos el password que el usuario ha introducido y lo comparamos con el original. Un booleano es devuelto para indicar si la autenticación ha finalizado con éxito. Este método se llama desde la Action a la que se le envía la petición de usuario desde el formulario de login. Veamos el código de esta Action, `hpc.action.AuthenticationAction`.

```
1 package hpc.action;
2
3 import hpc.database.UserBean;
4 import hpc.database.UserManager;
5 import hpc.database.AuthorBean;
6 import hpc.database.AuthorManager;
7 import hpc.util.HpcException;
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10
11 import org.apache.commons.beanutils.BeanUtils;
12 import org.apache.struts.action.Action;
13 import org.apache.struts.action.ActionForm;
14 import org.apache.struts.action.ActionForward;
15 import org.apache.struts.action.ActionMapping;
16 import org.apache.struts.action.ActionErrors;
17 import org.apache.struts.action.ActionError;
18
19 import hpc.security.AuthenticationManager;
20
21
22 public class AuthenticateAction extends Action{
23 /**
24  * Process the specified HTTP request, and create the corresponding HTTP
25  * response (or forward to another web component that will create it).
26  * Return an <code>ActionForward</code> instance describing where and how
27  * control should be forwarded, or <code>null</code> if the response has
28  * already been completed.
29  *
30  * @param mapping The ActionMapping used to select this instance
31  * @param form The optional ActionForm bean for this request (if any)
32  * @param request The HTTP request we are processing
33  * @param response The HTTP response we are creating
34  *
35  * @return Action to forward to
36  * @exception Exception if an input/output error or servlet exception occurs
37 */
38 public ActionForward execute(
39         ActionMapping mapping,
40         ActionForm form,
41         HttpServletRequest request,
42         HttpServletResponse response)
43 throws Exception {
44
45     AuthorBean author;
46     UserBean user = new UserBean();
47     BeanUtils.copyProperties(user,form);
48
49     if(!AuthenticationManager.getInstance().isUserAuthentified(user.getUsername(),
50                                     user.getPassword
50())){
51         ActionErrors errors = new ActionErrors();
52         errors.add("loginID",new ActionError("error.login"));
53         saveErrors(request, errors);
54         return mapping.getInputForward();
55     }
56     try{
57         UserBean[] v = ((UserManager.getInstance()).loadByWhere("where username = '"+
```

Figura 53: Archivo AuthenticationAction.java(I)

```

59         if(v.length > 0){
60             user.setRole((String)v[0].getRole());
61             request.getSession().setAttribute("loggedUser", user);
62         }
63     AuthorBean[] v1 = ((AuthorManager.getInstance()).loadByWhere("where username
= '"++
64 user.getUsername()+"'"));
65     if(v1.length > 0){
66         author = v1[0];
67         request.getSession().setAttribute("loggedAuthor", author);
68     }
69 }
70 catch (HpcException hpce) {
71     getServlet().log("Connection.close", hpce);
72     throw hpce;
73 }
74 return mapping.findForward("success");
75 }
76 }
```

Figura 54: Archivo AuthenticationAction.java(II)

El método ejecutado es el método `execute()`. En la línea 50, se comprueba si el usuario y password son correctos. Si no son correctos, introducimos en el objeto Request una instancia de la clase `org.apache.struts.action.ActionErrors` que será recogida desde la página JSP para indicarle al usuario que se ha equivocado.

En las líneas 57 y 62 recuperamos del la base de datos información del usuario y del autor respectivamente. Un usuario de la aplicación puede no ser autor pero si lo es, necesitamos información del autor para decidir en cada pantalla qué mostrar. En las líneas 60 y 65 introducimos en sesión dos instancias de objetos `hpc.database.UserBean` y `hpc.database.AuthorBean`. Ambos permanecerán en sesión hasta que el usuario cierre el navegador. La información de usuario, guardada en la instancia de la clase `hpc.database.UserBean` se utiliza en el proceso de autorización, que explicamos a continuación.

## Autorización

Como adelantábamos en el apartado de diseño, en el archivo `web.xml` especificamos qué clase gestionará la autorización. Adjuntamos a continuación ese fragmento de código.

```
<!--Servlet Filter that handles site authorization.-->
<filter>
    <filter-name>AuthorizationFilter</filter-name>
    <filter-class>hpc.security.AuthorizationFilter</filter-class>
    <description>
        This Filter authorizes user access to application
        URI.
    </description>
    <init-param>
        <param-name>error_page</param-name>
        <param-value>/home.jsp</param-value>
    </init-param>
</filter>
```

Figura 55: Archivo web.xml

El elemento **Filter** se utiliza para definir una clase Filter y sus propiedades. Una clase Filter puede ser utilizada para otros intereses, tales como por ejemplo contar el número de peticiones de usuario en una página web concreta. En nuestro caso, la utilizamos para gestionar la autorización. El atributo **filter-class** sirve para especificar la clase que implementa. El atributo **init-param** es donde se incluyen los atributos de clase. En nuestro caso, nos interesa tener como atributo el nombre de la página JSP a dónde el usuario será redireccionado en caso de no estar autorizado para acceder a la página que introdujo en la URL del navegador.

El método **doFilter()** de la clase **AuthorizationFilter** es invocado cada vez que se intenta acceder a una URL. Vamos a comentar aspectos relevantes del código de este método.

```

1 package hpc.security;
2 import java.io.IOException;
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import org.apache.struts.action.ActionError;
6 import org.apache.struts.action.ActionErrors;
7 import hpc.database.UserBean;
8
9 /**
10 * Filters all requests for application resources and uses an AuthorizationManager
11 * to authorize access.
12 * @author Michael Klaene
13 */
14 public class AuthorizationFilter implements Filter {
15
16     private String errorPage;
17
18     /**Filter should be configured with an system error page.*/
19     public void init (FilterConfig filterConfig) throws ServletException {
20         if (filterConfig != null) {
21             errorPage = filterConfig.getInitParameter("error_page");
22         }
23     }
24
25     /**Obtain user from current session and invoke a singleton AuthorizationManager to
determine if
26     * user is authorized for the requested resource.
27     * If not, forward them to a standard error page.
28     */
29     public void doFilter(ServletRequest request, ServletResponse response,FilterChain
chain)
30     throws ServletException, IOException {
31         HttpSession session = ((HttpServletRequest)request).getSession(false);
32         if(session == null){
33             returnError(request,response);
34         }
35         else{
36             UserBean currentUser = (UserBean)session.getAttribute("loggedUser");
37
38             if (currentUser == null) {
39                 ActionErrors errors = new ActionErrors();
40                 errors.add("loginID",new ActionError
("error.login"));
41                 returnError(request,response);
42             }
43             else {
44                 //Get relevant URI.
45                 String URI = ((HttpServletRequest)request).getRequestURI();
46
47                 //Invoke AuthorizationManager method to see if user can access resource.
48                 boolean authorized = AuthorizationManager.getInstance().
isUserAuthorized
49                 (currentUser,URI);
50                 if (authorized) {
51                     if(chain != null)
52                         chain.doFilter(request,response);
53                 }
54             else {
55                 if(!authorized)

```

Figura 56: Archivo AuthorizationFilter.java(I)

```

56             returnError(request,response);
57         }
58     }
59 }
60 }
61 public void destroy(){}
62 /**Accepts error string, forwards to error page with error.*/
63 private void returnError(ServletRequest request, ServletResponse response)
64 throws ServletException, IOException {
65     ActionErrors errors = new ActionErrors();
66     errors.add("loginID",new ActionError("error.login"));
67     request.setAttribute("Globals.ERROR_KEY",errors);
68     request.getRequestDispatcher(errorPage).forward(request,response);
69 }
70}

```

Figura 57: Archivo AuthorizationFilter.java(II)

En la línea 41, tras comprobar en la línea 37 la existencia de una sesión, se comprueba también que el usuario se haya autenticado. Como ya adelantamos en el apartado de **Autentificacion**, intentamos recuperar de sesión una instancia de la clase `hpc.database.UserBean`. Si no existe esta instancia, significará que el usuario no se ha autenticado y se devolverá un error, redirigiendo al usuario a la pantalla de autenticación.

En la línea 54,

el método `isUserAuthorized()` de la clase `hpc.security.AuthorizationManager` comprueba si el usuario está autorizado a visitar la URL en cuestión. Nuevamente aquí, la instancia de la clase `hpc.database.UserBean` es utilizada para recuperar el atributo `role` y comprobar si el rol de este usuario es válido para la URL. Recordemos que un usuario puede ser *admin* ó *user*.

Para acabar con la Autorización,

adjuntamos el código de la clase `hpc.security.AuthorizationManager`. Esta clase implementa el método `isUserAuthorized()`.

Esta clase es una clase *singleton*. En el constructor se inicializa el atributo `roleMappings` con los roles de los usuarios de la aplicación que son leídos del fichero de propiedades **/WEB-INF/mapping.properties**.

En el método `isUserAuthorized()`, se recorre el atributo `roleMappings` hasta encontrar para el rol del usuario la URL a la que intenta acceder ó acabar de recorrer `roleMappings`. Finalmente un booleano es devuelto para indicar si el usuario está autorizado ó no a acceder a la URL.

```
1 package hpc.security;
2 import java.util.*;
3 import java.io.FileInputStream;
4 import hpc.database.UserBean;
5
6 /**
7  * Default Implementation of AuthorizationManager. Loads mappings from a
8  * Properties file and compares URI with user's roles.
9  * @author Michael Klaene
10 */
11 public class AuthorizationManager {
12
13     // Contains role mappings.
14     private Properties roleMappings;
15     private static AuthorizationManager singleton = new AuthorizationManager();
16
17     /** Load mappings from a properties file on the file system. */
18     private AuthorizationManager() {
19         // Read in properties file containing role mappings...
20         this.roleMappings = new Properties();
21         try {
22             this.roleMappings.load(
23                 new FileInputStream("/home/zeta/PFC/hpc/WebContent/WEB-INF/
mapping.properties"));
24         } catch (Exception e) { throw new RuntimeException(e);}
25     }
26
27     /**
28      * Get the AuthorizationManager singleton.
29      * @return AuthorizationManager
30      */
31     synchronized public static AuthorizationManager getInstance() {
32         return singleton;
33     }
34     /**
35      * Returns boolean indicating whether user has the appropriate role for the
36      * specified URI.
37      */
38     public boolean isUserAuthorized(UserBean user, String uri) {
39         boolean matchFound = false;
40         boolean authorized = false;
41         Iterator i = roleMappings.entrySet().iterator();
42
43         // Loop through user roles, exit once match is found.
44         while ((!authorized) && (i.hasNext())) {
45             Map.Entry me = (Map.Entry) i.next();
46             // Pattern match. '*' should be interpreted as a wildcard for any
47             // ASCII character.
48             String mapPattern = ((String) me.getValue())
49                 .replaceAll("\\*", "*");
50             matchFound = Pattern.matches(mapPattern, uri);
51             if (matchFound && user.getRole().equals(me.getKey())) authorized = true;
52         }
53     }
54     return authorized;
55 }
56}
```

Figura 58: Archivo AuthorizationManager.java(II)

### 5.3. El modelo

Como ya adelantamos en el apartado de diseño, nuestro modelo está compuesto de un conjunto de clases Java. Por cada tabla de nuestra base de datos tenemos un objeto *Bean* que lo representa. Tenemos además una clase que contiene todas las operaciones sobre esa tabla que hemos considerado necesarias.

Siguiendo con el ejemplo del apartado del **Controlador**, comentamos la clase `AuthorBean.java` y la clase `AuthorManager.java`.

La clase `AuthorBean` consta principalmente de tantos atributos como columnas tiene la tabla `author` (que es la implementación física del diseño lógico de la tabla `investigador`). Por cada una de las columnas existe un atributo booleano que indica si la columna asociada ha sido modificada ó no.

La clase `AuthorManager` contiene las operaciones de insertar, actualizar y borrar sobre la tabla. Tiene también un conjunto de operaciones de consulta. Las operaciones de insertar y actualizar reciben un objeto `AuthorBean` como parámetro y lo utilizan para recuperar el valor de sus atributos e insertar ó actualizar un registro en la tabla `author`. Las funciones de consulta devuelven objetos `AuthorBean` (un array ó un único objeto) que contienen los registros que satisfacen determinadas consultas. Todas las operaciones capturan las excepciones de tipo `SQLException` y devuelven excepciones de tipo `HpcException`.

Hemos valorado la posibilidad de adjuntar el código entero de estas clases en esta documentación pero ocupa demasiado y no queremos sobrecargar al lector. Hemos optado por enseñar a continuación la operación `loadByWhere(...)` de la clase `AuthorManager` y comentarla.

```

    public AuthorBean[] loadByWhere(String where, int[] fieldList, int startRow, int
numRows)
throws
HpcException
{
    String sql = null;
    if(fieldList == null)
        sql = "select " + ALL_FIELDS + " from author " + where + " order by "
               + FIELD_NAMES
[ID_USERNAME];
    else{
        StringBuffer buff = new StringBuffer(128);
        buff.append("select ");
        for(int i = 0; i < fieldList.length; i++){
            if(i != 0)
                buff.append(",");
            buff.append(FIELD_NAMES[fieldList[i]]));
        }
        buff.append(" from author ");
        buff.append(where);
        sql = buff.toString();
        buff = null;
    }
    Connection c = null;
    Statement pStatement = null;
    ResultSet rs = null;
    java.util.ArrayList v = null;
    try{
        c = getConnection();
        pStatement = c.createStatement();
        rs = pStatement.executeQuery(sql);
        v = new java.util.ArrayList();
        int count = 0;
        while(rs.next() && rs.getRow()!=startRow)numTotal++;
        if (rs.getRow()==startRow & numRows!=0) {
            do
            {
                if(fieldList == null)
                    v.add(decodeRow(rs));
                else
                    v.add(decodeRow(rs, fieldList));
                count++;
                numTotal++;
            } while ( (count<numRows||numRows<0) && rs.next() );
            while(rs.next())numTotal++;
        }
        return (AuthorBean[])v.toArray(new AuthorBean[0]);
    }
    catch(SQLException sqle){
        throw new HpcException(
                "Error in op LOAD for {"+where+"} sql where clause in author
table",sqle);
    }
    finally{
        if (v != null) { v.clear(); }
        getManager().close(pStatement, rs);
        freeConnection(c);
    }
}
}

```

Figura 59: Operación loadByWhere de la clase AuthorManager.java

La función `loadByWhere` de la figura 60 recibe cuatro parámetros.

- El parámetro `where` contiene la cláusula `WHERE` de la sentencia SQL.
- El parámetro `fieldList` contiene los nombres de las columnas que queremos que devuelva la consulta.
- `numRows` y `startRow` son dos enteros que se utilizan para indicar el número de registros que hemos de devolver(del total que satisfacen la consulta) y a partir de qué registro comenzamos a devolver respectivamente. Ambos parámetros son utilizados para paginar las listas en las pantallas JSP. Por ejemplo, un investigador que lleva muchos años publicando, puede tener un número muy elevado de artículos publicados en congresos. Para el usuario de la aplicación no es cómodo tener que moverse por la pantalla subiendo y bajando el cursor para poder buscar un artículo entre muchos. Hemos paginados las listas de forma que un máximo de diez items es enseñado al usuario en una pantalla, si quiere ver los diez siguientes ha de clicar sobre uno de los botones de la parte inferior. Esto queda reflejado en la figura ??.

Lo primero que hace la función `loadByWhere` es comprobar si el parámetro `fieldList` contiene campos, si no los contiene, devuelve todas las columnas. Después recupera una conexión del `pool` de conexiones(aunque para esta clase sucede de forma transparente) y ejecuta la query. Si ocurre cualquier excepción, se captura y se devuclve una excepción de tipo `hpc.util.HpcException` tal y como explicábamos en el apartado de diseño. Si todo va bien, carga en un vector de objetos `AuthorBean` los registros que han satisfecho a la consulta.

#### 5.4. Conexión con la base de datos

Como explicamos en la sección de diseño, la gestión de las conexiones a la base de datos la realizamos através de JNDI y el pool de conexiones que implementa Tomcat. La explicación puede consultarse en este apartado ó consultar alguna de nuestras fuentes bibliográficas si se quiere aprender con más detalle [9].

A continuación mostramos el método `getConnection()` de nuestra clase `hpc.database.Manager` donde puede apreciarse como recuperamos el objeto `DataSource` através del servicio de nombre (JNDI). Todo lo concerniente a implementación de la gestión de conexiones es transparente para nosotros.

```
public synchronized Connection getConnection() throws HpcException{
    Connection tc = (Connection)trans_conn.get();
    if (tc != null) {
        return tc;
    }
    if (driver != null && url != null && username != null && password != null)
    {
        try{
            return DriverManager.getConnection(url, username, password);
        }
        catch(SQLException sqle){
            throw new HpcException("Error getting a connection from driver
manager",sqle);
        }
    }
    else{
        if (ds!=null) {
            try{
                return ds.getConnection();
            }
            catch(SQLException sqle){
                throw new HpcException("Error getting a connection from the pool",sqle);
            }
        }
        else{
            try{
//                Obtain our environment naming context
                Context initCtx = new InitialContext();
                Context envCtx = (Context) initCtx.lookup("java:comp/env");
//                Look up our data source
                ds = (DataSource)envCtx.lookup("jdbc/hpc");
//                Allocate and use a connection from the pool
                return ds.getConnection();
            }
            catch (NamingException ne) {
                throw new HpcException("Error looking up java:comp/env for jdbc/
hpc",ne);
            }
            catch(SQLException sqle){
                throw new HpcException("Error getting a connection from the pool",sqle);
            }
        }
    }
}
```

Figura 60: Operación getConnection() de la clase Manager.java

En el archivo de configuración **web.xml** es necesario especificar que habrá un recurso. Lo hacemos de la siguiente manera:

```
<resource-ref>
    <description>Datasource for connection</description>
    <res-ref-name>jdbc/hpc</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Y en el fichero **context.xml** especificamos los datos de la conexión:

```
<Context path="" docBase="/hpc" debug="0" reloadable="true" crossContext="true">
    <Resource name="jdbc/hpc" auth="Container" type="javax.sql.DataSource"
        maxActive="50" maxIdle="5" maxWait="20000"
        username="root" password=""
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://localhost:3306/hpc?autoReconnect=true"/>
</Context>
```

## 5.5. Modelo físico de la base de datos

En este apartado incluimos el modelo físico de la base de datos, es decir, las sentencias SQL que ejecutamos para generar nuestra base de datos en MySQL.

```
-- 
-- Table structure for table 'user'
-- 

CREATE TABLE user (
    user_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL,
    role ENUM('USER', 'ADMIN') DEFAULT 'USER',
    last_update TIMESTAMP NOT NULL,

    UNIQUE(username),
    PRIMARY KEY (user_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
```

```
-- Table structure for table 'author'
--
CREATE TABLE author (
    author_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    is_member CHAR(1) NOT NULL DEFAULT '1',
    is_phd CHAR(1) DEFAULT '0',
    last_update TIMESTAMP NOT NULL,
    UNIQUE (username),
    PRIMARY KEY (author_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Table structure for table 'journal'
--
CREATE TABLE journal (
    journal_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    journal_name TINYTEXT NOT NULL,
    issn VARCHAR(30) NOT NULL,
    last_update TIMESTAMP NOT NULL,
    UNIQUE (issn),
    PRIMARY KEY (journal_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Table structure for table 'country'
--
CREATE TABLE country (
    country_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    country_name VARCHAR(50) NOT NULL,
    last_update TIMESTAMP NOT NULL,
    UNIQUE (country_name),
    PRIMARY KEY (country_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
--  
-- Table structure for table 'publishing'  
  
CREATE TABLE publishing (  
    publishing_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
    publishing_name TINYTEXT NOT NULL,  
    last_update TIMESTAMP NOT NULL,  
    UNIQUE (publishing_name(255)),  
    PRIMARY KEY (publishing_id)  
)ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
--  
-- Table structure for table 'conference'  
  
CREATE TABLE conference(  
    conference_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
    -- congress C, day's journal J, simposi S,others O  
    ceremony_type CHAR(1) NOT NULL DEFAULT 'C',  
    date_beg DATETIME NOT NULL,  
    date_end DATETIME NOT NULL,  
    conference_name TINYTEXT NOT NULL,  
    acronym VARCHAR(30) NOT NULL,  
    region VARCHAR(30) NOT NULL,  
    country_id SMALLINT UNSIGNED NOT NULL,  
    -- Autonomous A, statal conference S,international I  
    scope CHAR(1) NOT NULL DEFAULT 'I',  
  
    -- research R, teaching T, others O  
    interest_area CHAR(1) NOT NULL DEFAULT 'R',  
    month_pub ENUM('jan','feb','mar','apr','may','jun',  
                  'jul','aug','sep','oct','nov','dec') DEFAULT 'jan',  
    year_pub YEAR NOT NULL,  
    proceedings_title TINYTEXT,  
    issn VARCHAR(30),  
    -- paper P, cdrom C, web pages W, others O  
    support_pub CHAR(1) NOT NULL DEFAULT 'P',  
    publishing_id SMALLINT UNSIGNED NOT NULL,  
    isTop CHAR(1) DEFAULT '0',  
    last_update TIMESTAMP NOT NULL,  
  
    PRIMARY KEY (conference_id),  
    FOREIGN KEY (publishing_id) REFERENCES publishing(publishing_id)
```

```
        ON DELETE RESTRICT ON UPDATE CASCADE,
FOREIGN KEY (country_id) REFERENCES country(country_id) ON DELETE
                    RESTRICT ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--  
-- Table structure for table 'language'  
--  
CREATE TABLE language (
    language_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    language_name VARCHAR(50) NOT NULL,
    last_update TIMESTAMP NOT NULL,  

    PRIMARY KEY (language_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--  
-- Table structure for table 'university'  
--  
CREATE TABLE university (
    university_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    university_name TINYTEXT NOT NULL,
    last_update TIMESTAMP NOT NULL,  

    PRIMARY KEY (university_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--  
-- Table structure for table 'institution'  
--  
CREATE TABLE institution(
    institution_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    institution_name TINYTEXT NOT NULL,
    last_update TIMESTAMP NOT NULL,  

    PRIMARY KEY (institution_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--  
-- Table structure for table 'conference_paper'
```

```
--  
CREATE TABLE journal_paper(  
    document_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
    journal_id SMALLINT UNSIGNED NOT NULL,  
    month_pub ENUM('jan','feb','mar','apr','may','jun','jul',  
                  'aug','sep','oct','nov','dec') NOT NULL DEFAULT 'jan',  
    year_pub YEAR NOT NULL,  
    volum VARCHAR(10),-- I have seen volums like this E87D  
    number_ VARCHAR(10),  
    title TINYTEXT NOT NULL,  
    -- paper P, cdrom C, web pages W, others O  
    support_pub CHAR(1) NOT NULL DEFAULT 'P',  
    pag_beg SMALLINT UNSIGNED NOT NULL,  
    pag_end SMALLINT UNSIGNED NOT NULL,  
    notes TEXT,  
    file_link VARCHAR(30),  
    last_update TIMESTAMP NOT NULL,  
  
    PRIMARY KEY (document_id),  
    FOREIGN KEY (journal_id) REFERENCES journal(journal_id)  
                  ON DELETE RESTRICT ON UPDATE CASCADE  
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
--  
-- Table structure for table 'conference_paper'  
--  
CREATE TABLE conference_paper(  
    document_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
    conference_id SMALLINT UNSIGNED NOT NULL,  
    month_pub ENUM('jan','feb','mar','apr','may','jun','jul',  
                  'aug','sep','oct','nov','dec') NOT NULL DEFAULT 'jan',  
    year_pub YEAR NOT NULL,  
    title TINYTEXT NOT NULL,  
    -- invited, paper, poster  
    reason_pub ENUM('I','PA','PO') NOT NULL DEFAULT 'PO',  
    -- full text F, abstract A  
    type_pub CHAR(1) NOT NULL DEFAULT 'F',  
    pag_beg SMALLINT UNSIGNED NOT NULL,  
    pag_end SMALLINT UNSIGNED NOT NULL,  
    notes TEXT,  
    file_link VARCHAR(50),
```

```
last_update TIMESTAMP NOT NULL,  
  
PRIMARY KEY (document_id),  
FOREIGN KEY (conference_id) REFERENCES conference(conference_id)  
    ON DELETE RESTRICT ON UPDATE CASCADE  
)ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
--  
-- Table structure for table 'chapter'  
--  
CREATE TABLE chapter(  
    document_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
    isbn VARCHAR(30) NOT NULL,  
    -- research,teaching phg, teaching btech, others  
    book_type ENUM('R','TP','TB','O') NOT NULL DEFAULT 'R',  
    language_id SMALLINT UNSIGNED NOT NULL,  
    book_title TINYTEXT NOT NULL,  
    publishing_id SMALLINT UNSIGNED NOT NULL,  
    country_id SMALLINT UNSIGNED NOT NULL,  
    region VARCHAR(50),  
    publishing_ref VARCHAR(30),  
    -- paper P, cdrom C, web pages W, others O  
    support_pub CHAR(1) NOT NULL DEFAULT 'P',  
    -- Autonomous A, statal conference S,international I  
    scope CHAR(1) NOT NULL DEFAULT 'I',  
    month_pub ENUM('jan','feb','mar','apr','may','jun','jul',  
        'aug','sep','oct','nov','dec') NOT NULL DEFAULT 'jan',  
    year_pub YEAR NOT NULL,  
    num_pag SMALLINT UNSIGNED,  
    num_edition TINYINT UNSIGNED,  
    num_authors TINYINT UNSIGNED,  
  
    title TINYTEXT NOT NULL,  
    pag_beg SMALLINT UNSIGNED NOT NULL,  
    pag_end SMALLINT UNSIGNED NOT NULL,  
    notes TEXT,  
    file_link VARCHAR(50),  
    last_update TIMESTAMP NOT NULL,  
  
PRIMARY KEY (document_id),  
FOREIGN KEY (publishing_id) REFERENCES publishing(publishing_id)
```

```
        ON DELETE RESTRICT ON UPDATE CASCADE,
FOREIGN KEY (country_id) REFERENCES country(country_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
FOREIGN KEY (language_id) REFERENCES language(language_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--  
-- Table structure for table 'thesis'  
--  
CREATE TABLE thesis(
    document_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    author_id SMALLINT UNSIGNED,
    month_pub ENUM('jan','feb','mar','apr','may','jun','jul',
                   'aug','sep','oct','nov','dec') NOT NULL DEFAULT 'jan',
    year_pub YEAR NOT NULL,
    -- cum laude A, excellent B, good C, pass D
    qualification CHAR(1) NOT NULL DEFAULT 'A',
    university_id SMALLINT UNSIGNED,
    title TINYTEXT NOT NULL,
    notes TEXT,
    file_link VARCHAR(50),
    last_update TIMESTAMP NOT NULL,
    PRIMARY KEY (document_id),
    FOREIGN KEY (university_id) REFERENCES university(university_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY( author_id) REFERENCES author(author_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--  
-- Table structure for table 'report'  
--  
CREATE TABLE report(
    document_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    -- research R, work W
    report_type ENUM('R','W') NOT NULL DEFAULT 'R',
    month_pub ENUM('jan','feb','mar','apr','may','jun','jul',
                   'aug','sep','oct','nov','dec') NOT NULL DEFAULT 'jan',
```

```
year_pub YEAR NOT NULL,  
project_id SMALLINT UNSIGNED,  
title TINYTEXT NOT NULL,  
num_pag SMALLINT UNSIGNED,  
institution_id SMALLINT UNSIGNED NOT NULL,  
num_inst1 VARCHAR(30) NOT NULL,  
num_inst2 VARCHAR(30),  
notes TEXT,  
file_link VARCHAR(30),  
last_update TIMESTAMP NOT NULL,  
  
PRIMARY KEY(document_id),  
FOREIGN KEY(institution_id) REFERENCES institution(institution_id)  
          ON DELETE RESTRICT ON UPDATE CASCADE,  
FOREIGN KEY(project_id) REFERENCES project(project_id) ON DELETE  
          RESTRICT ON UPDATE CASCADE  
)ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
--  
-- Table structure for table 'J_author'  
--  
CREATE TABLE J_author(  
author_id SMALLINT UNSIGNED NOT NULL,  
document_id SMALLINT UNSIGNED NOT NULL,  
author_order SMALLINT UNSIGNED NOT NULL,  
last_update TIMESTAMP NOT NULL,  
  
PRIMARY KEY(author_id,document_id),  
FOREIGN KEY(author_id) REFERENCES author(author_id) ON DELETE  
          RESTRICT ON UPDATE CASCADE,  
FOREIGN KEY(document_id) REFERENCES journal_paper(document_id)  
          ON DELETE CASCADE ON UPDATE CASCADE  
)ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
--  
-- Table structure for table 'Phd_author'  
--  
CREATE TABLE Phd_advisor(  
advisor_id SMALLINT UNSIGNED NOT NULL,  
document_id SMALLINT UNSIGNED NOT NULL,
```

```
author_order SMALLINT UNSIGNED NOT NULL,  
last_update TIMESTAMP NOT NULL,  
  
PRIMARY KEY(advisor_id,document_id),  
FOREIGN KEY(advisor_id) REFERENCES author(author_id) ON  
DELETE RESTRICT ON UPDATE CASCADE,  
FOREIGN KEY(document_id) REFERENCES thesis(document_id) ON DELETE  
CASCADE ON UPDATE CASCADE  
)ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
--  
-- Table structure for table 'C_author'  
--  
CREATE TABLE C_author(  
author_id SMALLINT UNSIGNED NOT NULL,  
document_id SMALLINT UNSIGNED NOT NULL,  
author_order SMALLINT UNSIGNED NOT NULL,  
last_update TIMESTAMP NOT NULL,  
  
PRIMARY KEY(author_id,document_id),  
FOREIGN KEY(author_id) REFERENCES author(author_id) ON DELETE  
RESTRICT ON UPDATE CASCADE,  
FOREIGN KEY(document_id) REFERENCES conference_paper(document_id)  
ON DELETE CASCADE ON UPDATE CASCADE  
)ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
--  
-- Table structure for table 'TR_author'  
--  
CREATE TABLE TR_author(  
author_id SMALLINT UNSIGNED NOT NULL,  
document_id SMALLINT UNSIGNED NOT NULL,  
author_order SMALLINT UNSIGNED NOT NULL,  
last_update TIMESTAMP NOT NULL,  
  
PRIMARY KEY(author_id,document_id),  
FOREIGN KEY(author_id) REFERENCES author(author_id) ON DELETE  
RESTRICT ON UPDATE CASCADE,  
FOREIGN KEY(document_id) REFERENCES report(document_id) ON DELETE  
CASCADE ON UPDATE CASCADE
```

```
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--  
-- Table structure for table 'IB_author'  
--  
CREATE TABLE IB_author(  
    author_id SMALLINT UNSIGNED NOT NULL,  
    document_id SMALLINT UNSIGNED NOT NULL,  
    author_order SMALLINT UNSIGNED NOT NULL,  
    last_update TIMESTAMP NOT NULL,  
  
    PRIMARY KEY(author_id,document_id),  
    FOREIGN KEY(author_id) REFERENCES author(author_id) ON DELETE  
        RESTRICT ON UPDATE CASCADE,  
    FOREIGN KEY(document_id) REFERENCES chapter(document_id) ON DELETE  
        CASCADE ON UPDATE CASCADE  
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

## 6. Pruebas

Las pruebas son una actividad dentro del proceso de Validación y Verificación. En ellas pretendemos revisar las especificaciones, el diseño y la codificación del sistema para comprobar que las funcionalidades implementadas se ajustan a las especificaciones recogidas y cumplen todos los requerimientos.

Las pruebas tienen como objetivo principal la detección de errores de implementación y/o funcionalidad.

Podemos distinguir:

- Pruebas unitarias: en ellas se comprueba el funcionamiento de un componente aislado
- Pruebas de integración: en ellas se comprueba cómo interaccionan éstos componentes
- Pruebas de validación: en ellas se comprueba si el sistema satisface los requerimientos desde el punto de vista del usuario
- Pruebas de sistema: en ellas se comprueba que todos los elementos del sistema se han integrado correctamente

Para realizar estas pruebas, pueden tomarse dos enfoques:

- Enfoque de caja negra: es un método que no contempla la estructura interna de sistema. Se centra en descubrir errores del tipo:
  1. Funciones incorrectas ó ausentes
  2. Errores de interfaces
  3. Errores de acceso a información
  4. Errores de rendimiento
  5. Errores de inicio ó finalización
- Enfoque de caja blanca: es un método que se centra en el análisis de la estructura interna del sistema:
  1. Ejecución de todos los caminos de manera independiente
  2. Ejecución de todas las ramas de cada condicional
  3. Ejecución de todos los bucles
  4. Uso de todas las estructuras de datos

Para la realización de las pruebas hemos utilizado una herramienta de testeo, TestLink [13]. Con ella hemos construido un conjunto de test cases. Para el proceso de ejecución de los mismos, nos hemos ayudado también de otra herramienta para reportar los errores, Bugzilla [12]. Gracias a ésta última, hemos podido clasificar los errores encontrados según prioridad y componentes. Ambas nos han servido para tener una intuición de en qué punto está nuestra aplicación y estimar el tiempo que hemos dedicado y tendremos que dedicar a mejorarla.

Hemos decidido no darle a este proceso un enfoque de caja blanca. Creemos que la complejidad algorítmica del sistema en general no lo requiere y por otra parte, no disponemos hasta el momento, del tiempo necesario para realizar este tipo de pruebas que implicarían el aprendizaje de otra herramienta que nos ayudara a construirlas.

Con la herramienta Testlink hemos diseñado un total de 138 *testcases*, agrupados por componentes. Por ejemplo, para el componente que engloba la gestión de un artículo publicado en una revista, tendríamos un conjunto de *testcases* para probar el alta, la actualización, la baja, una búsqueda, etc.

La batería de pruebas ó *testcases* han sido probados inicialmente por mí, y ahora, tras la corrección de una serie de errores ó *bugs* que he considerado críticos, los vuelve a probar mi director de proyecto. En el proceso de ejecución de los *testcases*, los *testcases* verifican que la aplicación responda bien en cuanto a requerimientos funcionales (que haga lo que se le ha pedido que ha de hacer).

En cuanto a requerimientos no funcionales, debemos comprobar que la aplicación es fácilmente navegable e intuitiva. Con la ejecución de la batería de pruebas, mi director, tiene la oportunidad de corregir la apariencia, mensajes al usuario e incluso, si fuera necesario, la navegación entre pantallas. Por último deberíamos contemplar la concurrencia. Para ello probamos la ejecución de los *testcases* en paralelo (diferentes máquinas con diferentes personas) sobre los mismos datos y/o sobre las mismas funcionalidades.

Hemos contemplado la posibilidad de adjuntar los *testcases* en la memoria como un apéndice pero, el volumen de papel que ello supondría nos ha hecho descartar la alternativa dejando al lector la posibilidad de enviarnos un mail para solicitarlos.<sup>5</sup>

---

<sup>5</sup>zhidalgo@ac.upc.edu

## 7. Posibles ampliaciones

Como ya adelantábamos en la introducción. Una de las características que ha de tener nuestra herramienta es la de ser accesible para todos los miembros de la línea. Desde cualquier máquina interna al departamento se ha de poder acceder a la aplicación. El grupo CAP ha de generar listados de forma periódica y se encuentra frecuentemente con el problema de no tener toda la información centralizada. Con la herramienta pretendemos incentivar a los investigadores a que ellos mismos realicen la tarea de introducir la información de sus publicaciones.

Otra forma de incentivar a los investigadores es através de la gestión de los viajes. Cuando el artículo de un investigador es aceptado en un congreso, este investigador ha de asistir a la conferencia y presentar el artículo. Para que la línea se haga cargo de los gastos del viaje, el investigador ha de presentar un presupuesto y la persona encargada de aceptar presupuestos dentro de la línea ha de aprobarlo. Una ampliación de la aplicación será la implementación de un módulo que gestione este concepto. Si el investigador ha de solicitar la financiación através de la aplicación, uno de los requisitos puede ser dar de alta en la aplicación el artículo que presentará en el congreso.

La aplicación es accesible desde cualquier máquina interna a la red del departamento. Esto quiere decir que los listados de las publicaciones no pueden consultarse en la red. Está previsto desarrollar una web para listar todas las publicaciones. Esta web se generará dinámicamente con la información mantenida desde nuestra aplicación y sí será accesible desde fuera del departamento.

Por último, está también previsto desarrollar un módulo para la gestión de las patentes en las que participa la línea CAP.

## 8. Conclusión

Creemos que hemos conseguido los objetivos que explicábamos en la primera sección.

Por un lado, la migración se ha completado con éxito. Hemos recuperado la información almacenada en los ficheros de texto plano. Contrastando un conjunto de listados generados por nuestra aplicación y por la aplicación original comprobamos que los resultados son los mismos. Podemos concluir entonces en que nuestros datos son consistentes con los datos almacenados en la aplicación anterior.

Por otro lado, todos los casos de uso especificados han sido implementados. También los *test cases* están siendo ejecutados y tras corregir errores que se encontraron en la primera ronda de test, estamos subiendo el porcentaje de *test cases* superados con éxito. En definitiva, teniendo en cuenta que los *test cases* cubren todos los casos de uso del sistema y en condiciones muy diversas, parece justo pensar que la aplicación podrá ser utilizada con normalidad.

Creemos también que, toda la estructura de la aplicación y su documentación está pensada para que en un futuro, a cualquier persona, con los conocimientos necesarios por supuesto, le sea muy sencillo ampliarla.

También la tecnología utilizada está en continuo desarrollo. Si fuera el caso que una mejora importante se añadiera a las nuevas versiones, podríamos plantear una migración de nuestra versión a las nuevas sin que ello supusiera una gran inversión de tiempo.

Finalmente, el éxito de cualquier aplicación depende en gran parte de la aceptación de sus usuarios. En nuestro caso, los usuarios de la aplicación hasta el momento han sido mis compañeros de despacho y mi director de proyecto. Durante el diseño de pantallas he tenido la opinión y el *asesoramiento* de todos ellos. Podemos decir entonces que, si mis compañeros de despacho son un grupo suficientemente representativo del conjunto de usuarios de la aplicación, ésta será entonces del agrado de todos.

## 9. Valoración personal

Cuandó trabajé por primera vez como programadora en una empresa de informática me horrorizaba al ver cómo de mal se hacían las cosas. No entendía como mi analista, aún siendo ella también *fiber*, podía ordenar indiscriminadamente la construcción de clases para aquí y clases para allá... A dónde habían ido a parar el diagrama de clases, los casos de uso, las restricciones... ¿dónde estaban las operaciones OCL que había yo de implementar? Sentía como aquella figura que ordenaba en mí y a la que no podía oponerme había desciertado... Sí, había abandonado nuestra doctrina. Todos aquellos postulados dc cómo se hacen bien las cosas habían perdido sentido para ella y ... sólo el tiempo y los recursos cobraban protagonismo.

Desde aquello han pasado ya casi tres años. He trabajado en una empresa de informática más, con un equipo bastante más grande de compañeros y bastante más metódicos también. Ya no me horroriza saber que he de acabar un trabajo en un tiempo determinado y que la improvisación y el regateo van a ser mis mayores aliados. Ahora entiendo mejor porqué en las empresas no siempre pueden ser todo lo metódicos que quisieran. Pero no es aquí donde voy a explicar porqué creo que lo entiendo, no es aquí donde continúe con el debate estrella que tantas veces he disfrutado con mis compañeros de facultad.

Durante todo el desarrollo del proyecto he tenido presente mis dos experiencias: la empresa y la FIB. Las he comparado ante cada situación que lo ha merecido y, en muchas ocasiones he optado por obrar como lo hubiera hecho en la empresa y otras muchas obrar como la FIB me ha enseñado.

Cuando Agustín me ofreció este proyecto pensé que me venía *como anillo al dedo*. Necesitaba acabar mi licenciatura en el menor tiempo posible y necesitaba también unos ingresos. Además ya estaba familiarizada con alguna tecnología web y no sería muy complicado volver a retomarlo. Pensé que recuperaría los libros de diseño del software de algún armario, aprendería UML y me *machacaría* por las noches en casa diseñando como en la facultad nos habían enseñado a hacer. Cuando llegó el momento de ponerse *manos a la obra* y empezar a tomar decisiones, descubrí que Agustín confiaba muchísimo en mis elecciones y yo tenía la responsabilidad de escoger las tecnologías con un criterio forjado y no en base a la experiencia que ya tenía. Olvidé por completo la planificación inicial, la especificación y el diseño, y comencé a documentarme de todas las alternativas tecnológicas que teníamos. Instalé el software de cada una de ellas, hice mis ejemplos, le di unas cuantas vueltas y elegí. Cuando ya estaba segura de mi elección ya tenía todo un esqueleto montado de lo que podía ser mi aplicación: una base de datos bastante coherente a los requerimientos que Agustín me había ido adelantando, unas clases

java que representaban a la perfección el modelo... incluso unos formularios que mostraban datos. Finalmente, mi diagrama de clases, casos de uso, operaciones... quedaron delegados al momento de empezar a documentar esta memoria.

Las consecuencias de esta metodología *expontánea e improvisadora* han sido buenas y malas.

En ocasiones, he descubierto *tarde* que no había tenido en cuenta restricciones y/o funcionalidades y esto ha repercutido en el tiempo de desarrollo. He tenido que desechar gran parte del trabajo que tenía hasta el momento y sentarme con tranquilidad a abstraer los conceptos para traducirlos a algo implementable.

Si tuviera que volver a hacer un proyecto desde *cero*, pararía muchísimo más tiempo a recoger los requerimientos y a *marearlos* hasta creer que los he entendido bien. También haría un diagrama de clases e intentaría normalizarlo para obtener el diseño lógico de la base de datos. Creo que volvería a no hacer los diagramas de secuencia ó los casos de uso. Cuando he cometido un error me he ayudado del diagrama de clases y ello a repercutido a veces en el diseño de la base de datos, pero nunca he desarrollado un diagrama de secuencia. Supongo que indirectamente, al pensar en la navegabilidad entre mis clases lo he estado haciendo, pero nunca en UML. Con todo esta disertación quiero decir que parte del trabajo realizado en esta documentación, concretamente los diagramas de secuencia ó los contratos de las operaciones, no han sido, desde mi punto de vista, de gran utilidad a la hora de construir mi aplicación. Y esto no quiere decir que no sean útiles, quiero decir que yo no he sabido sacarles provecho en esta experiencia y en principio, no recurriría a ellos en alguna otra.

Por otro lado, he desarrollado más mi habilidad de resolver situaciones puntuales, buscando soluciones parciales. Puede parecer esta última frase un tanto difusa: podemos querer solucionar una cuestión y pueden existir un conjunto de soluciones válidas. Algunas de estas soluciones son muy buenas y otras no tan buenas pero igualmente válidas. No siempre necesitamos la solución excelente, sobre todo si la implementación de la misma nos impide desarrollar otras cuestiones con la misma efectividad que si no la implementáramos. En nuestro caso, nuestra aplicación es muy sencilla. Muy sencilla en cuanto al modelo de negocio y muy sencilla en cuanto aspectos como la seguridad ó la autenticación. Para estos dos últimos, las soluciones no excelentes son una muy buena opción y creo que aprender a tomar este tipo de decisiones no es sencillo. Creo también que yo he sabido tomar bastantes decisiones como ésta.

A lo largo de mis estudios en la facultad he aprendido mucho. Parte de lo aprendido he podido ponerlo en práctica en las empresas en las que he estado.

Parte de lo aprendido lo he intentado poner en práctica en las empresas en las que he estado. Gran parte de lo aprendido lo he tenido casi siempre presente en todo lo que he hecho. Creo que es tan importante tener presente que las cosas se pueden hacer mejor de lo que las estamos haciendo que hacerlas de la mejor manera posible. Lo primero nos permite mejorar y siempre podemos hacer las cosas mejor de lo que las hacemos.

La Zoraida que cumplió en junio a desarrollar este proyecto no es la Zoraida que ahora escribe su valoración. Durante estos seis meses he aprendido muchísimo. Es por ello que me siento capaz de mirar el trabajo realizado y ser objetiva para juzgar que puedo mejorarlo. Es por esto que me siento muy satisfecha con los resultados obtenidos.

## 10. Bibliografía

### Referencias

- [1] Dolors Costal C., M<sup>a</sup> Ribera Sancho S. y Ernest Teniente L. *Enginyeria del software. Especificació* Edicions UPC, 2000.
- [2] Cristina Gómez, Enric Mayol, Antoni Olivé y Ernest Teniente L. *Enginyeria del software. Disseny I* Edicions UPC, 2003.
- [3] Carles Farré, Antoni Olivé y Carme Quer. *Enginyeria del software. Disseny II* Edicions UPC, 1999.
- [4] Chuck Cavannes y Brian Keeton. *Jakarta Struts. Pocket Reference*. O'Reilly and Associates, 2003.
- [5] Chuck Cavannes. *Jakarta Struts* O'Reilly and Associates, 2002.
- [6] Jon Stephens y Chad Russell. *Beginning MySQL Database Design and Optimization* Novice to Professional (Apress, 2004).
- [7] Jason Brittain y Ian F. Darwin. *Tomcat: The Definitive Guide* O'Reilly and Associates, 2003.
- [8] Página web de Struts <http://struts.apache.org/>
- [9] Página web de Tomcat <http://tomcat.apache.org/>
- [10] Página web de MySql <http://www.mysql.org/>
- [11] Página web de SQL2Java [ql2java.sourceforge.net/](http://ql2java.sourceforge.net/)
- [12] Página web de Bugzilla <http://www.bugzilla.org/>
- [13] Página web de TestLink [testlink.sourceforge.net](http://testlink.sourceforge.net)
- [14] Página web de latex [www.latex-project.org/](http://www.latex-project.org/)

