3

Implementación

3.1 Implementación del Cliente

3.1.1 Hojas de estilo

La definición de la interfaz gráfica del sistema se encuentra centralizada en 2 hojas de estilo: `estilosAdmon.css` y `estilosCliente.css`, las cuales contienen los elementos requeridos para la vista del trabajador y del cliente respectivamente. En la siguiente tabla se muestran estos elementos (similares para ambas hojas pero con definiciones diferentes).

Elemento	Tipo	Descripción
body	Etiqueta (<>)	Cuerpo de las páginas JSP, en él se define la imagen de fondo.
a		Vínculos requeridos en el sistema.
CContenedor	Id (#)	Capa principal de la interfaz que contiene a las demás (contenedor externo).
CTitulo		Capa externa superior que muestra la hora, la sesión y el logo.
CCentro		Capa externa en que se cargan los datos a través de la navegación del sistema.
CMenu		Capa externa que contiene las opciones de menú.
MenuMP		Capa externa que contiene el menú oculto de la Materia Prima.
MenuProd		Capa externa que contiene el menú oculto de las producciones.
Logo		Capa externa que contiene el logo de la PyME.
Sesion		Capa externa que contiene los datos de la sesión actual.
Hora		Capa externa que muestra el reloj.
Contenedor		Capa principal dentro de la navegación en CCentro (contenedor interno).

Tabla 3.1: Elementos de `estilosAdmon.css` y `estilosCliente.css`

Elemento	Tipo	Descripción
Titulo	Id(#)	Capa que contiene el título en el contenedor interno.
Izquierda		Capa que posiciona a la izquierda los datos en el contenedor interno.
Derecha		Capa que posiciona a la derecha los datos en el contenedor interno.
Centro		Capa que centra los datos en el contenedor interno.
Regresar		Capa estática que contiene la redirección a la página anterior (cuando es necesario).
Letra1	Clase (.)	Letra utilizada en las capas externas.
LBotones		Letra utilizada para las tablas.
LTitulo		Letra utilizada en los títulos de cada página.
LSubtitulo		Letra utilizada en caso de ser requeridos subtítulos.
LTexto		Letra utilizada para texto en general.
Campos		Define el estilo de los campos de comunicación con el usuario.
Botones		Define el estilo de los botones.
Botones:hover		Define el estilo de los botones cuando el mouse pasa encima de estos.
BotonDisabled		Define el estilo de los botones están inhabilitados.
LTrans		Letra pequeña utilizada en las tablas que tienen muchos datos.
label.error [1]		Letra que marca algún error en la validación de datos.
input.error, select.error,textarea.error [1]		Define el estilo que se le dará a los campos cuando contengan errores de validación.

Tabla 3.1 (continuación)

1. Elementos necesarios para el plugin `Validate.js`

Elemento	Tipo	Descripción
altaConsulta	Clase (.)	Tabla utilizada para consultas y alta de datos.
tablaEstatica		Tabla que contiene datos de reportes.
tablaEstatica thead		Encabezado de tablas estáticas
tablaEstatica tbody tr:nth-child(even)		Los renglones pares de las tablas.
tablaEstatica tbody tr:nth-child(odd)		Los renglones impares de las tablas.
ul.pageNav li[2]		Estilo de los números en la paginación.
ul.pageNav li a[2]		Estilo de los vínculos en la paginación.
li.currentPage[2]		Estilo del número de la página actual en la paginación.
ul.pageNav li.currentPage a[2]		Estilo del link de la página actual en la paginación.
pager[2]		Capa que contiene los datos de la paginación

Tabla 3.1 (continuación): elementos de `estilosAdmon.css` y `estilosCliente.css`

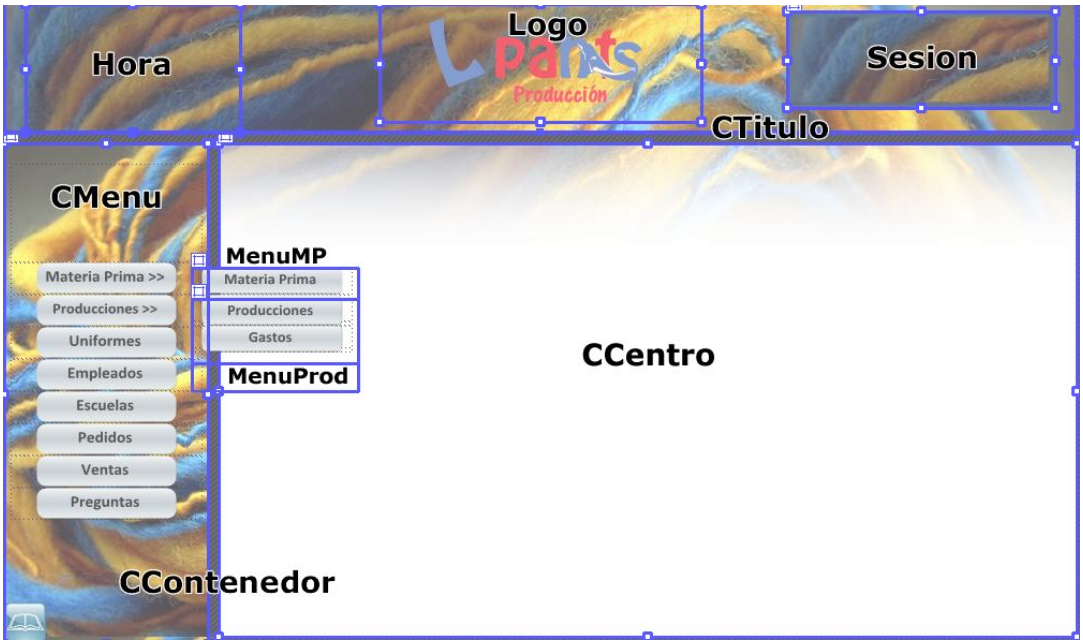

Figura 3.1: Estructura de las capas externas de la interfaz de usuario

2. Elementos necesarios para el plugin `QuickerPage.js`

Figura 3.2: Estructura de las capas internas de la interfaz de usuario

3.1.2 jQuery

Las funciones de jQuery fueron divididas en cuanto a su funcionalidad y área de aplicación en diferentes archivos para ser cargadas por las páginas JSP cuando se requiera. A continuación se enlistan los archivos `.js` con su contenido:

1. funcionesP.js y funcionesC.js

Contienen las funciones generales del área de Producción y Ventas así como la del Cliente, con sus respectivas implementaciones. Contiene las siguientes funciones:

Función	Descripción
cargaPaginaLoad()	Carga en CCentro la página solicitada en el enlace linkExt
muestraSubmenuMP()	Muestra el submenú de la Materia Prima cuando el mouse entra al área indicada.

Tabla 3.2: Contenido de `funcionesP.js` y `funcionesC.js`

Función	Descripción
ocultaSubmenuMP()	Oculta el submenú de la Materia Prima cuando el mouse sale del área indicada.
muestraSubmenuP()	Muestra el submenú de las Producciones cuando el mouse entra al área indicada.
ocultaSubmenuP()	Oculta el submenú de las Producciones cuando el mouse sale del área indicada.
atenua()	Atenúa los botones del menú principal cuando entra el mouse al área.
quitaAtenuacion()	Regresa a su color original los botones del menú principal cuando el mouse sale del área.
$(document).bind("contextmenu",function(e){ })	Deshabilita el uso del botón derecho del ratón.

Tabla 3.2 (*continuación*)

2. funcionesAjax.js

Contiene en su mayoría las funciones que hacen uso de ajax tanto para alguna de las áreas de aplicación de la PyME así como para el cliente. Su contenido se muestra a continuación.

Función	Descripción
convierteAMys()	Convierte a mayúscula cada letra introducida en algún campo una vez que la tecla termina de ser presionada.
detallesMostrarOcultar()	Muestra u oculta de las capas la información contenida en algunas tablas.
cargaPaginaLoad()	Carga en CCentro la página solicitada en el enlace linkInt.
cargaPaginaAjax()	Carga en CCentro la información solicitada a través de un formulario.

Tabla 3.3: Contenido de `funcionesAjax.js`

Función	Descripción
obtenMunicipios()	Llena el combo de los municipios de acuerdo al estado seleccionado.
obtenSucursales()	Llena el combo de las sucursales de acuerdo al proveedor seleccionado.
obtenMP()	Llena el combo de la materia prima de acuerdo al tipo seleccionado.
obtenNiveles()	Llena el combo de los niveles de con que cuenta la escuela seleccionada.
obtenTipoUniforme()	Llena el combo del Uniforme para la combinación dada de escuela, nivel y grado.
obtenTallas()	Llena el combo de las tallas de acuerdo al uniforme seleccionado.
obtenPrecioActual()	Obtiene el precio actual de la talla seleccionada
obtenGrados()	Llena el combo de los grados de acuerdo al nivel seleccionado.
cuentaCaracteres()	Limita el número de caracteres introducidos en un área de texto.
confirmaBorrado()	Envía la confirmación de borrado de datos al usuario antes de ser procesada la petición.
confirmaEntrega()	Envía la confirmación de entrega de un uniforme pedido.
confirmacionPositiva(var acepta, var pagina)	Elige la tarea a hacer una vez que se preguntó por la confirmación.
deshabilitarBoton()	Deshabilita un objeto del tipo submit.
habilitarBoton()	Habilita un objeto del tipo submit.
$(".modificar").each(function(){ })	Envía la confirmación de modificación de datos antes de ser procesada por el servidor.

Tabla 3.3 (*continuación*)

Función	Descripción
$(".validar").each(function(){ })	Valida los campos del formulario antes de ser procesados por el servidor.
$(".paginacion tbody.pagina").quickPager({ })	Lleva a cabo la paginación del cuerpo de la tabla clasificada como *paginacion*.
$('#f1,#f2').datepicker({ })	Configura el calendario para los campos con id f1 y f2 que hacen uso de fechas.

Tabla 3.3 (*continuación*)

3. login.js

Debido a que el vínculo de Inicio de sesión y los datos de sesión se encuentran en su propia capa (Sesion), esta funcionalidad debe ser separada de los datos que son manejados en la capa CCentro.

Función	Descripción
loginLoad()	Lleva a cabo algunas tareas de la sesión especificadas en el vínculo seleccionado.
loginAjax()	Lleva a cabo el inicio de sesión y carga los datos en la capa de Sesion.

Tabla 3.4: Contenido de `login.js`

4. autocompletar.js

Con el fin de agilizar el registro de pedidos de usuarios registrados, fue implementado un *auto-completador* que muestra los clientes que coinciden con las letras que se van introduciendo en el campo de búsqueda.

Función	Descripción
controlaLista(e)	Envía al servidor un criterio de búsqueda diferente cada que el usuario introduce una nueva letra y crea una lista con los resultados coincidentes.

Tabla 3.5: Contenido de `autocompletar.js`

Función	Descripción
ocultaLista()	Cuando se da click en el campo de búsqueda se oculta la lista de elementos que coinciden con el criterio.
seleccionaItem()	Una vez que el elemento buscado es mostrado en la lista y se da click sobre este, el valor aparece en el campo de búsqueda para continuar con la tarea.

Tabla 3.5 (*continuación*)

5. reloj.js

Contiene únicamente el método `reloj()` que obtiene la hora del sistema, la muestra y asigna una imagen para cada etapa del día. Esta función es llamada desde `funcionesP.js` y `funcionesC.js` en un intervalo de 30 segundos.

3.1.3 Plugins de jQuery

Si bien jQuery facilita el uso de javascript para un desarrollo rápido y sencillo, las actuales demandas del web2.0 exigen mucho más de lo que este ofrece: Validación de datos sin necesidad de recargar la página, reproducción de audio y video, paginación de datos, subida de archivos al servidor por medio de ajax son sólo algunas de estas necesidades. Sin embargo, existe una infinidad de extensiones o plugins de código abierto que permiten al programador un ahorro considerable de código para implementar estas funcionalidades. Las extensiones utilizadas para el desarrollo del sistema fueron las siguientes:

3.1.3.1 QuickPager.js

Es muy común que al buscar información en una base de datos, esta sea presentada de manera que el usuario pueda verla de una forma cómoda, si el volumen de datos que se regresa es muy extenso, el usuario puede confundirse e incluso enfadarse por la información. Es por ello que la paginación es un recurso bastante utilizado en todos los sistemas que manejan una buena cantidad de datos. Este plugin contiene toda la lógica de la paginación para que sólo sea necesario

indicar la clase de la tabla a paginar y a través del método `quickPager()` se configura el tamaño de la página, el número de la página por defecto en que se comenzará a mostrar la información y la capa en la que se publicará la numeración.

```
1   $(".paginacion tbody.pagina").quickPager( {
2      pageSize: 8,
3      currentPage: 1,
4      holder: ".pager"
5   });
```

Código 3.1: Uso de `QuickPager.js`

3.1.3.2 Validate.js

Es imprescindible garantizar que los datos que llegan al servidor lo hagan en un formato correcto para evitar problemas posteriores en su manipulación y en la seguridad del sistema. Este plugin contiene validaciones para: números, dígitos, fechas, rangos, valores y por ser de código abierto, el programador puede aumentar las necesarias. Su uso resulta bastante sencillo: En el `form` se debe indicar la clase de la validación y colocar en cada campo el(los) criterio(s) de validación necesario(s) mostrados a continuación:

- required: Indica que el campo es obligatorio
- minlength(longitud): Longitud mínima aceptada para el campo.
- maxlength(longitud): Longitud máxima aceptada para el campo.
- min(valor): Valor mínimo aceptada para el campo.
- min(valor): Valor máximo aceptado para el campo.
- date( ): Formato de fecha (originalmente acepta en formato yyyy/MM/dd pero puede ser modificado a dd/MM/yyyy).
- number( ): Campo que acepta únicamente números.
- digits( ): Campo que acepta únicamente dígitos.
- equalTo(campo): El valor del campo debe coincidir con el de otro.
- letras(): Campo que admite únicamente letras (este criterio fue agregado debido a que no se encontraba definido).

En caso de error, crea una etiqueta `<label>` a un costado del campo e imprime el mensaje que debe ser mostrado al usuario (el mensaje es configurado directamente desde código en el plugin) y a su vez modifica el estilo del campo con error. El estilo del `label` y los campos con error debe ser configurado por el programador en una hoja de estilo.

```
1   <form class="validar" method="post"
2    action="/Uniformes/paginasCliente/PersonaAction.do">
3   <input name="nombre" type="text" class="Campos required letras"
4    minlength="10" maxlength="50" />
5   <input name="calle" type="text" class="Campos required letras"
6    minlength="5" maxlength="50" />
7   <input name="cp" type="text" class="Campos required digits"
8    minlength="5" value="${param.cp}"size="5" maxlength="5" />
```

Código 3.2: Validación de datos con `Validate.js`

```
1   $(.validar).validate()
```

Código 3.2 (*continuación*)

3.1.3.3 Datepicker.js

Es un plugin que muestra un calendario una vez que se da click en el campo destinado para las fechas. Contiene múltiples opciones de configuración tales como: El formato de fecha, la forma en qué será abierto el calendario (por medio de un botón extra o al posicionarse en el campo), el cambio del mes y del año, la velocidad en que aparece, el auto-dimensionamiento, la animación que tendrá al ser abierto, el conteo de semanas, el rango de fechas en que el usuario podrá seleccionar, entre otras.

Contiene, además del archivo js, un archivo css que define el estilo del calendario y que puede ser modificado por el programador. Por medio del método `datepicker()` se establecen las opciones de configuración de los campos f1 y f2 que serán los encargados de recoger las fechas.

```
1     $('#f1,#f2').datepicker({
2         dateFormat: 'dd/mm/yy',
3         changeMonth: true,
4         changeYear: true
5     });
```

Código 3.3: Uso de `DatePicker.js`

Para mayor información y documentación de estos plugins se pueden consultar las siguientes páginas:

- `http://www.geckonewmedia.com/blog/`
- `http://docs.jquery.com/Plugins/Validation`
- `http://docs.jquery.com/UI/Datepicker`

Un problema que se puede observar hasta el momento es que Javascript se convierte en un elemento vital de la aplicación, si este se encuentra desactivado, el sistema no funcionará correctamente y se obtendrán resultados inesperados. Para evitar esto, se revisa que Javascript se encuentre activado con la etiqueta `<noscript>` de HTML, y en caso de no estarlo, se direcciona a una página que da muestra del error para que el usuario lo active (a través de las opciones de configuración de su navegador[3]) o cambie de explorador.

3. Para mozilla: Herramientas>Opciones>Contenido>Habilitar Javascript
   Para IE: Herramientas>Opciones de Internet>Seguridad>Nivel Personalizado>Active Scripting>Habilitar
   Para Opera: Configuración>Opciones>Avanzado>Contenido> Habilitar Javascript
   Para Chrome: Opciones>Avanzadas>Proxy>Seguridad>Nivel Personalizado>ActiveScripting>Habilitar

3.2 Implementación del Servidor

3.2.1 Jerarquía de directorios

Para que la aplicación web pueda funcionar de manera correcta, necesita de una serie de recursos que la complementen: Páginas estáticas HTML, páginas dinámicas JSP, archivos de configuración, hojas de estilo (CSS), servlets, clases compiladas, imágenes, etcétera. La especificación de Sun para las aplicaciones web define por medio de una jerarquía de directorios la forma en que los recursos deben ser situados para ser cargados por el servidor. Este conjunto de directorios puede ser empaquetado en lo que se conoce como un Archivo de Aplicación Web (WAR por sus siglas en inglés) o bien, ubicarlos en la carpeta `webapps` del servidor `tomcat`. A continuación se muestra esta jerarquía.

Directorio	**Descripción**
/	Se trata del directorio raíz de la aplicación (el nombre es definido por el programador) que contiene todas los subdirectorios con las páginas HTML, JSP, las hojas de estilo, código Javascript, imágenes, etcétera
/WEB-INF	Contiene todos los archivos de configuración utilizados en la aplicación (`web.xml`, `struts-config.xml`, `applicationContext.xml`, etcétera).
/WEB-INF/classes	Contiene las clases y JSP compilados.
/WEB-INF/lib	Contiene los ficheros Java (JAR) de las distintas librerías requeridas para la aplicación.

Tabla 3.6 Directorios estándar de aplicaciones web

3.2.2 Módulos

Struts permite la modularización de la aplicación a través de sus diferentes áreas funcionales para que estas puedan ser tratadas con independencia. Cada módulo requiere de su propio archivo de configuración, sus propios actions, forwards y JSPs. La manera en que se lleva a cabo esta acción es la siguiente:

1. Crear el archivo de configuración del módulo
2. Registrarlo en el descriptor de despliegue `web.xml`.
3. Configurar los accesos de los diferentes módulos en las JSPs.

1. Crear el archivo de configuración para cada módulo

Esta definición de un archivo de configuración por cada módulo permite que trabajen con independencia y los distintos objetos asociados puedan ser agrupados.

La aplicación está compuesta de 3 módulos: Producción, Ventas y Cliente.

2. Registrar cada archivo de configuración en `web.xml`

Los módulos son identificados de la siguiente manera:

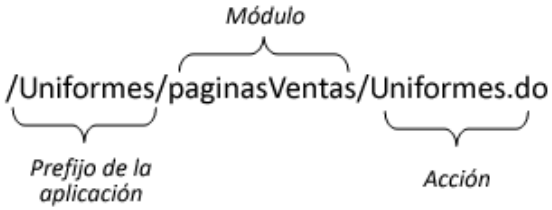

Figura 3.3: Configuración de los diferentes módulos en web.xml.

Y cada uno debe encontrarse debidamente registrado en el descriptor de despliegue como se muestra a continuación:

```
1    <servlet>
2        <servlet-name>action</servlet-name>
3        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
4        <init-param>
5            <param-name>config</param-name>
6            <param-value>/WEB-INF/struts-config.xml</param-value>
7        </init-param>
8        <init-param>
9            <param-name>config/paginasVentas</param-name>
10           <param-value>/WEB-INF/struts-configVentas.xml</param-value>
11       </init-param>
12       <init-param>
13           <param-name>config/paginasCliente</param-name>
14           <param-value>/WEB-INF/struts-configCliente.xml</param-value>
15       </init-param>
16   </servlet>
17   <servlet-mapping>
18       <servlet-name>action</servlet-name>
19       <url-pattern>*.do</url-pattern>
20   </servlet-mapping>
```

Código 3.4: Configuración de los diferentes módulos en `web.xml`.

En el parámetro `config` se registra el módulo por default (en este caso el de producción) junto con su ubicación. Para configurar los módulos restantes se debe especificar en este parámetro el nombre del módulo y el archivo que contiene las definiciones para este.

3. Configurar los accesos de los diferentes módulos en las JSPs.

Para indicar el módulo al que se debe dirigir una acción, este debe ser especificado en la URL (Ver figura 3.3). En caso de que no se detalle entrará a su módulo por defecto.

3.2.3 DispatchAction

Como se mencionó en la sección 2.4 (Struts), cada petición es interceptada por una subclase `Action` y la procesa en su método `execute()` que debe ser sobreescrito por el programador con la funcionalidad específica. Así, para cada petición debe existir un `Action` que sea capaz de atenderla. Por ejemplo:

Dar de alta un nuevo uniforme → NuevoUniforme.do
Ver un uniforme → VerUniforme.do
Borrar un uniforme → BorrarUniforme.do

¿Qué tienen en común estas tres acciones?
La lógica de negocio hacia la que van dirigidas es la misma: Los uniformes.

Struts dispone de una subclase de `Action` que permite procesar peticiones similares dentro de una misma clase en cada uno de sus métodos. Se trata de la clase `DispatchAction` que tiene implementado el método `execute()` por default y se encarga de determinar el método apropiado para procesar la petición por medio de un parámetro cuyo nombre es elegido por el programador y el valor de este debe coincidir con el nombre del método implementado que

deberá tener la misma anatomía del método `execute()`. Su funcionamiento se ejemplifica en la siguiente figura.

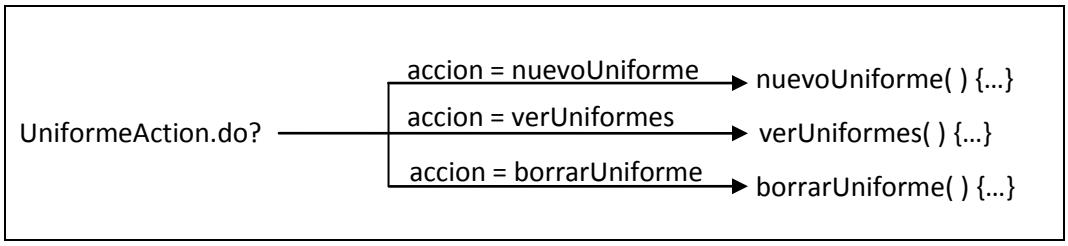

Figura 3.4: Funcionamiento de `DispatchAction`.

Donde:

4 UniformeAction: Es la subclase Action.

5 accion: Es el parámetro en el que se debe indicar el nombre del método que procesará la petición.

De esta forma, se crea una sola clase que maneje las diferentes peticiones en `n` métodos sin la necesidad de crear `n Actions` diferentes. Llevado a código el ejemplo anterior, se vería de la siguiente manera:

```
public class UniformeAction extends org.apache.struts.actions.DispatchAction {

    public ActionForward nuevoUniforme(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
            …
    }

    public ActionForward verUniformes(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
            …
    }

    public ActionForward borrarUniforme(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
            …
    }
}
```

Código 3.5: Ejemplo de subclase `DispatchAction`

La configuración de esta clase en `struts-config.xml` sólo debe contener, como extra, el nombre del parámetro que seleccionará el método indicado. Por lo demás, se hace de manera habitual.

```
1     <action name="UniformeForm" path="/UniformeAction"
2     type="org.springframework.web.struts.DelegatingActionProxy"
3     parameter="accion" scope="request">
4          <!-- Forwards -->
5     </action>
```

Código 3.6: Configuración de la subclase `DispatchAction` en `struts-config.xml`

3.2.4 Arquitectura de la aplicación

Con la finalidad de tener mayor control sobre el contenido de las clases que conforman el sistema. Estas se encuentran distribuidas en 5 paquetes:

1. Action: En este paquete se localizan todas las subclases `Action` (`DispatchAction` hereda a `Action`) que interceptarán la petición del usuario.
2. ActionForm: Contiene las subclases `ActionForm` que transportarán los datos del usuario.
3. Logica: Almacena las clases que se encargan de la lógica de negocio y que son llamadas por las clases contenidas en Action.
4. Persistencia: Contiene las clases que acceden a la información contenida en la base de datos por medio de Hibernate y se comunica con las clases del paquete de la Lógica.
5. Persistencia.Mapeos: Abarca las clases que conforman el modelo de dominio de la aplicación (mapeos de las tablas de la base de datos). Como se muestra en el diagrama de la figura 3.5.

La arquitectura completa de la aplicación se muestra en el diagrama de clases de la figura 3.6.

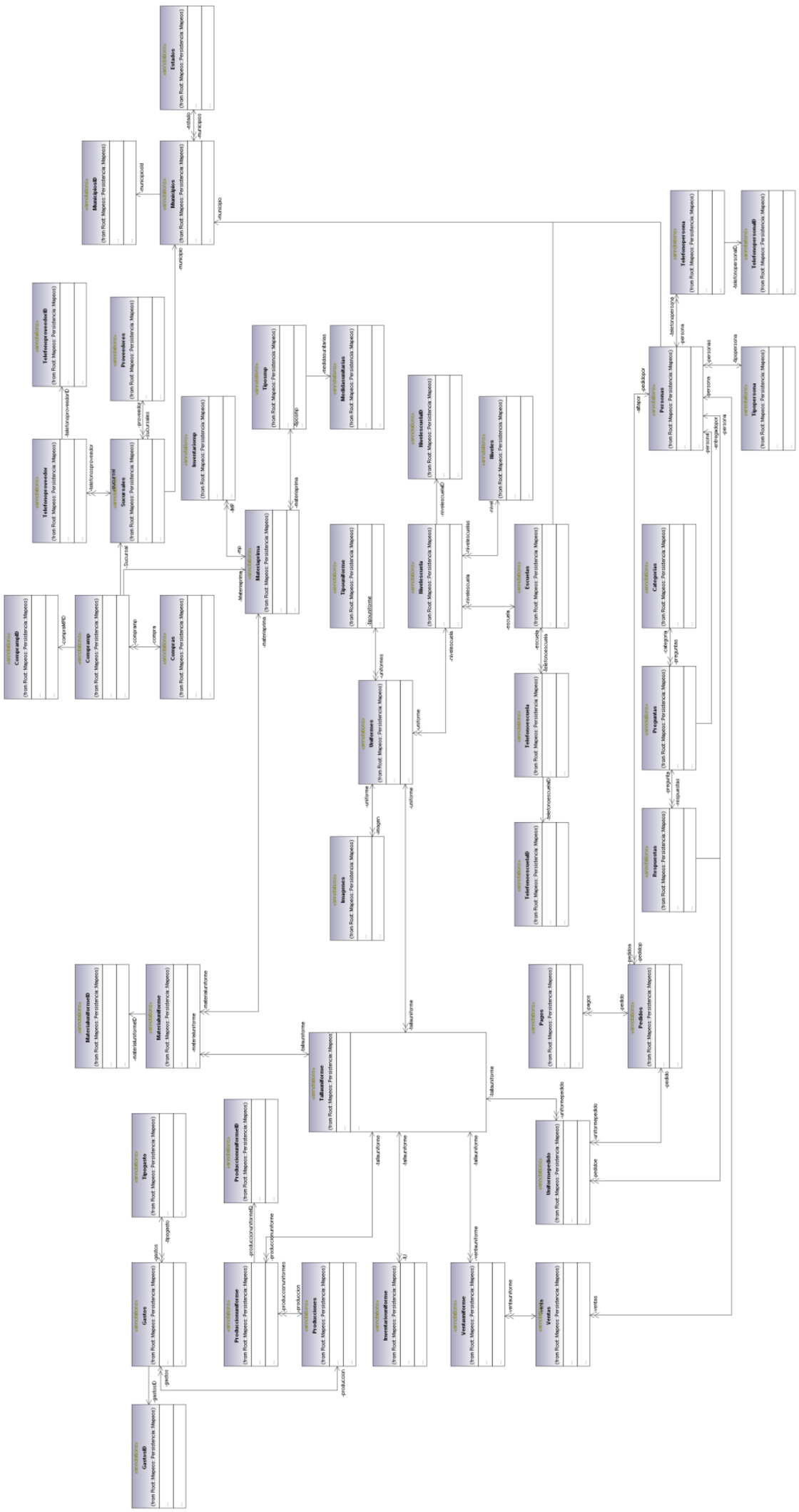

Figura 3.5: Modelo de Dominio de la aplicación

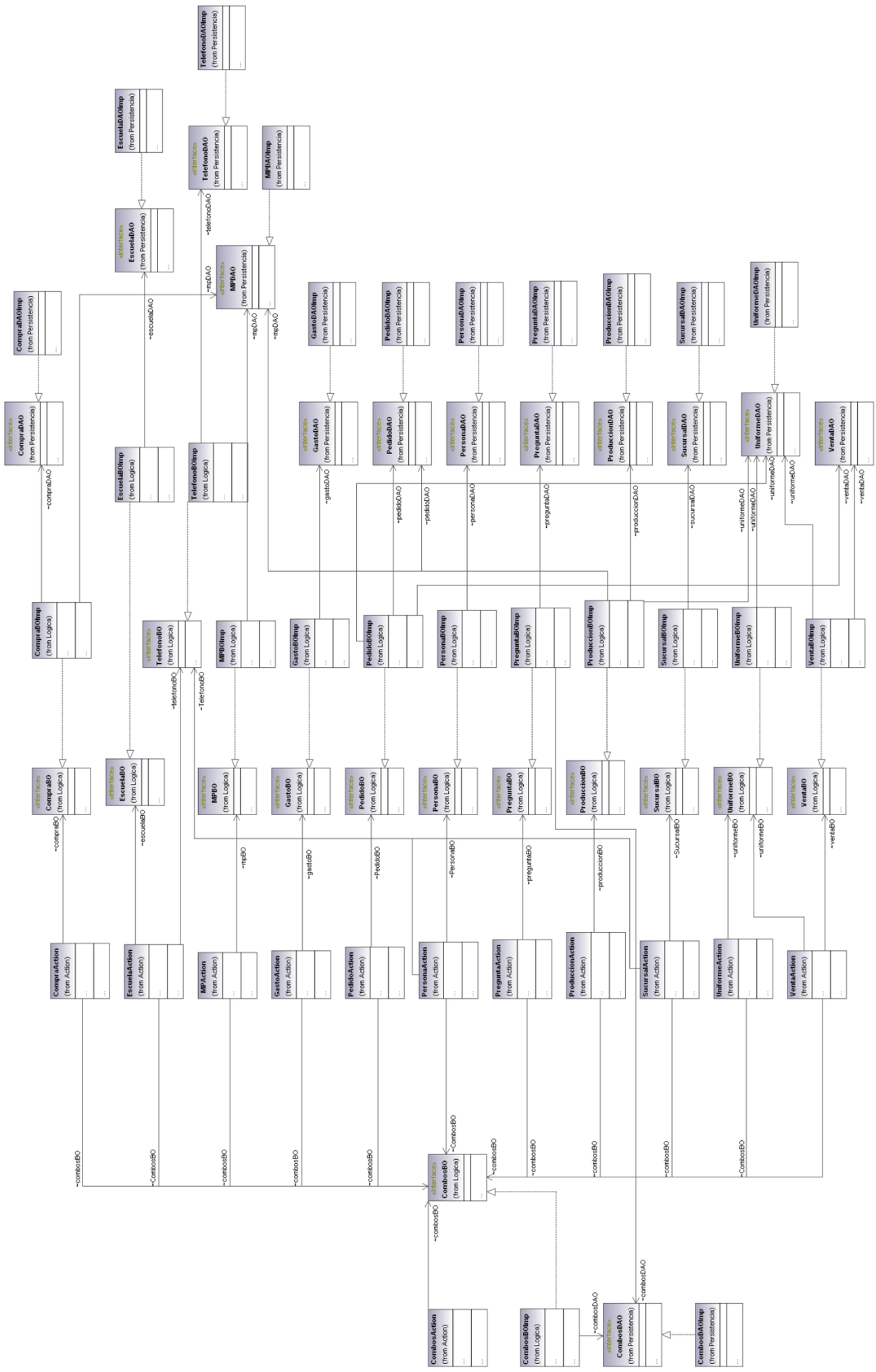

Figura 3.6: Arquitectura de la aplicación

3.2.5 Software involucrado en la construcción

Ambiente de Desarrollo	Herramientas de Diseño	Implementación
– Netbeans 6.9 – Windows 7 – Firefox 3.6 – Chrome 10 – Opera 10 – Internet Explorer 8	– ERWin 4.0 – Photoshop CS2 – Visio 2007 – Umodel 2011 – Fireworks MX – Dreamweaver 8 – Button Shop 4	– Tomcat – jQuery 1.4.2 – Oracle 11g R2 – Struts 1.3.8 – Hibernate 3 – Spring 2.5 – Log4J 1.2 – Kettle 3.1 – Pentaho 3.5

Tabla 3.7: Software involucrado en la construcción