MANUAL DE USUARIO

APLICACIÓN BASE PARA EL DESARROLLO EN J2EE

Edición: **V 1.6**

Autor: **Unidad de Informática Corporativa**
Área de Desarrollo y Mantenimiento

Fecha: **Junio 2011**

Historia del Documento			
Versión: **Beta**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta.		
	Elaborado por:	**Marta Martín Jiménez**	*Fecha*: **Diciembre 2003**
	Revisado por:	**Diego García Carrera**	*Fecha*: **Diciembre 2003**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Diciembre 2003**
Versión: **1.0**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta.		
	Elaborado por:	**Marta Martín Jiménez**	*Fecha*: **Febrero 2004**
	Revisado por:	**Diego García Carrera**	*Fecha*: **Marzo 2004**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Mayo 2004**
Versión: **1.2**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta. Indicando las modificaciones introducidas (con nuevos anexos).		
	Elaborado por:	**José Manuel González Martín**	*Fecha*: **Enero 2005**
	Revisado por:	**Diego García Carrera**	*Fecha*: **Enero 2005**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Enero 2005**
Versión: **1.3**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta. Indicando las modificaciones introducidas		
	Elaborado por:	**Diego García Carrera**	*Fecha*: **Febrero 2005**
	Revisado por:	**Juan Antonio Barras García**	*Fecha*: **Febrero 2005**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Febrero 2005**
Versión: **1.4**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta. Indicando las modificaciones introducidas		
	Elaborado por:	**Diego García Carrera**	*Fecha*: **Marzo 2006**
	Revisado por:	**Juan Antonio Barras García**	*Fecha*: **Mayo 2006**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Mayo 2006**
Versión: **1.5**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta. Indicando las modificaciones introducidas		
	Elaborado por:	**Diego García Carrera**	*Fecha*: **Enero 2007**
	Revisado por:	**Sonia Hernández Sánchez**	*Fecha*: **Febrero 2007**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Febrero 2007**
Versión: **1.6**	*Descripción*: Manual de uso sobre la herramienta que genera una aplicación base, si como una descripción del producto que genera tal herramienta. Indicando las modificaciones introducidas		
	Elaborado por:	**Sonia Hernández Sánchez**	*Fecha*: **Junio 2011**
	Revisado por:	**Diego García Carrera**	*Fecha*: **Junio 2011**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Junio 2011**

Lista de distribución del documento

Nombre	Localización

Índice

1 INTRODUCCIÓN

1.1 Objetivo del documento

La Aplicación Base para el desarrollo de aplicaciones Web en J2EE se trata de una esqueleto de aplicación que incorpora por defecto funcionalidades básicas y el comportamiento estándar que deben seguir las aplicaciones J2EE elaboradas según los estándares de la JCyL.

El objetivo que se pretende alcanzar con este desarrollo es que las aplicaciones de gestión se puedan construir a partir de esta aplicación base sin más que ampliar la funcionalidad específica. De esta manera todas las aplicaciones se construirán sobre la misma arquitectura lo que garantizará que ciertos comportamientos sean homogéneos para todas las aplicaciones.

Este proyecto posee un conjunto de características que son la aplicación de los estándares de desarrollo Web corporativo en la Junta de Castilla y León, así como un amplio número de recursos y herramientas que permiten mejorar la productividad a la hora de afrontar un desarrollo Web a medida y de calidad.

Las aplicaciones generadas a partir de este esqueleto base están asociadas con un completo sistema de seguridad que autorizan la interacción únicamente a los usuarios especificados y a las acciones previamente descritas. Además asegura que cumplen con los estándares corporativos de desarrollo Web y se beneficiarán de disponer de:

- Una gestión eficiente de los ficheros de configuración.
- Un Escritura en los ficheros de log.
- Una Organización del proyecto en directorios y ficheros
- Una calidad en la organización y estilos de la capa de presentación.
- Una arquitectura interna escalable y fácil de mantener.

1.2 Motivación de la Nueva Versión (1.6)

La última versión completa publicada (**Generador de Aplicaciones 1.5**) es de Febrero de 2007, y el número de actualizaciones es numeroso, así como la complejidad de los mismos; y se ve la necesidad de unificar todos estos cambios en una versión de Consolidación que disponga de los interfaces construidos.

El Generador de Aplicaciones Versión 1.5 que se compone de las siguientes librerías internas

jcylfw-1.3.1.jar

jcyldb-1.1.2.jar

jcylutiles-1.3.jar

SistSeguC-2.2.jar

struts-1.1.jar: Junto con todas sus dependencias

log4j-1.2.12.jar

Y desde esta versión, se han ido dando las siguientes recomendaciones de migración:

Últimas versiones publicadas de las librerías internas (jcyl*.jar)

jcylfw-1.6.2.jar

jcyldb-1.4.1.jar

jcylutiles-1.7.1.jar

Otras librerías recomendadas:

log4j-1.2.15.jar

struts-core-1.3.x.jar

struts-taglib-1.3.x.jar

struts-el-1.3.x.jar

struts-tiles-1.3.x.jar

struts-extras-1.3.x.jar (si se utiliza la clase DispatchAction)

Evolución del Sistema de Seguridad Corporativo

SistSeguC-2.7.3.jar

SistSeguR3C-2.7.jar

SistSeguXML-2.0.jar

Manuales de buenas prácticas o migraciones publicadas desde entonces:

APPBASE_ExtraccionParametrosEAR-1.1.pdf

APPBASE_OptimizacionRedespliegues-1.3.pdf

APPBASE_ProblemaPropagacionTrazas-1.0.pdf

APPBASE_Quartz-Scheduler-1.0.pdf

La nueva versión del Generador de Aplicaciones, 1.6, es una versión de consolidación que incorpora todas estas novedades y recomendaciones, lo que facilita el trabajo al programador, al no tener que utilizar versiones anteriores e incorporar manualmente estas actualizaciones.

Se trata de una versión que puede servir de modelo al programador para la migración de aplicaciones generadas con versiones anteriores, y que estén interesados en incorporar todas las novedades y mejoras publicadas.

1.3 Novedades de la nueva versión

Las actuaciones principales que se han llevado a cabo en esta nueva versión son:

- Incorporar las librerías internas actualizadas (jcyl*.jar) al esqueleto base de aplicación.
- Incorporar las librerías actualizadas del sistema de Seguridad Corporativo.
- Aplicar los cambios descritos en el manual

  APPBASE_ExtraccionParametrosEAR-1.1.pdf
- Aplicar los cambios descritos en el manual

APPBASE_OptimizacionRedespliegues-1.3.pdf

- Aplicar los cambios descritos en el manual

  APPBASE_ProblemaPropagacionTrazas-1.0.pdf

- Integrar el componente Quartz-Scheduler-1.0 siguiendo los pasos del manual APPBASE_Quartz-Scheduler-1.0.pdf
- Se han eliminado todas las partes obsoletas del esqueleto base de aplicación (WebApp.zip)
- Se ha eliminado la opción de “Generar OT y OAD”, ya que se trata de un código obsoleto.
- Se ha eliminado la parte GIS del proyecto, ya que esta opción actualmente se distribuye en un módulo independiente, no integrado dentro de la Aplicación Base.
- Se han incorporado los ficheros del proyecto de Jdeveloper ya para la versión 10.1.3 (NombreAplicacion_1013.jws y NombreAplicacion_1013.jpr).

2 ESTRUCTURA DE LA PROPUESTA

2.1 Estructura de la aplicación base

La propuesta contiene la estructura propia de una aplicación Web generada con los estándares de la JCYL:

Figura 1: Estructura Aplicación Base

- Componentes de presentación:
  - hojas de estilo en el directorio **css**
  - imágenes en el directorio **imagenes**
  - scripts de procesamiento en cliente en el directorio **javascript**
  - páginas HTML y páginas JSP a partir del directorio base de la aplicación
- Librerías necesarias, en el directorio **lib.** Se incluyen las librerías de struts versión 1.3 y sus dependencias, librería de log4j, y librerías propias de la aplicación base: **jcylfw-X.X.jar**, **jcylutiles-X.X.jar, jcylbd-1.0.jar** que contiene las clases propias del framework (en su versión correspondiente) y **SistSegu-X.X.X.jar** librería del sistema de seguridad (usando la versión correspondiente).
- Clases java de la aplicación base, en el directorio **src**

- Archivos de configuración. Para facilitar el mantenimiento de las aplicaciones, en esta versión se han sacado todos los parámetros posibles fuera del EAR de la aplicación siguiendo las recomendaciones del manual APPBASE_ExtraccionParametrosEAR-1.1.pdf; por este motivo los ficheros de configuración que van fuera del ear de la aplicación y que deben copiarse en el servidor se han colocado en el directorio configExterna, para facilitar el movimiento de ficheros al programador.
  - **WEB-INF\web.xml**: para definir la aplicación Web según el estándar DTD Web Application 2.3 definido en http://java.sun.com/dtd/web-app_2_3.dtd

- **WEB-INF\nombreAplicacion-config.xml:** donde se definen las características necesarias para el correcto funcionamiento del framework struts según el estándar DTD Struts Configuration 1.3 definido en http://struts.apache.org/dtds/struts-config_1_3.dtd
- **WEB-INF\tiles-def.xml:** donde se definen la organización de las páginas jsp en el caso de utilizar Tiles Struts (opcional y no utilizado de momento)
- **WEB-INF\validation.xml**: donde definir las reglas de validación (usado en el formulario de Login)
- **WEB-INF\validator-rules.xml:** definición de las reglas de validación en cliente basadas en javascript.
- **WEB-INF\config\ConfigurationRepository.xml:** que contiene la referencia a todos los ficheros de configuración de la aplicación, al que llamamos repositorio de configuración. Mas detalles en el apartado correspondiente
- **WEB-INF\config\ConfigurationRepository.xml:** que contiene la referencia a todos los ficheros de configuración de la aplicación, al que llamamos repositorio de configuración. En este fichero indicamos donde se encuentran los ficheros de configuración que hemos sacado del ear.
- **WEB-INF\config\log4jfw.properties:** que especifica la configuración del log interno al sistema.
- **configExterna\app-config.properties:** donde se puede parametrizar la aplicación a desarrollar, mediante la especificación de parámetros en forma nombre-valor en un archivo properties propio de java.
- **configExterna\JCYLPlanificador.properties:** que especifica la configuración de las tareas que se deseen planificar en el tiempo y sus características.
- **configExterna\log.xml:** que especifica la configuración del sistema de Log, basado en el componente log4j.

3 GENERAR APLICACIÓN BASE

En este apartado describiremos paso por paso como comenzar a desarrollar una nueva aplicación J2EE a partir de esta aplicación base.

3.1 Consideraciones Previas

Para poder ejecutar el resultado generado por la herramienta descrita posteriormente se debe tener en cuenta que tal producto se basa o utiliza **el sistema de seguridad para aplicaciones J2EE**. Este sistema se basa en un repositorio de datos independiente donde alojar los usuarios, permisos y descripción de la aplicación así como unos componentes de acceso al mismo.

Por lo tanto en este sistema de seguridad estar dados de alta al menos los elementos básicos de los que se compone la aplicación que son:

1. Registro con el nombre de la aplicación
2. Datos de un Usuario con acceso a esta aplicación con un Rol de acceso.
3. Al menos una función que pertenezca a esta aplicación y accesible por el usuario de entrada al sistema

El almacenamiento de estos objetos puede estar basado en **Base de Datos** o en **ficheros XML.** El Sistema de Seguridad principal es el basado en **Base de Datos**. La versión en ficheros **XML** está disponible para facilitar el desarrollo sin tener instalada la infraestructura completa de Seguridad.

Para conocer en qué consiste este sistema tanto versión BD como XML consultar el **Manual de Usuario del Sistema de Seguridad.**

3.1.1 Creación del esqueleto base de la aplicación

Esta aplicación se debe ejecutar desde la línea de comandos, debemos tener instalado JDK 1.4 o superior.

Consta de los siguientes ficheros:

GenerarAplicacionJ2EE1_6.jar

GenerarAplicacionJ2EE.properties

GestSeguXMLv1.exe

NombreAplicacion.zip

SeguridadBD.zip

SeguridadXML.zip

VariablesGenerarWin.properties

Para ejecutar el programa deberá teclear en línea de comandos lo siguiente: (siendo **X_X : Versión actual)**

```
java –jar GenerarAplicacionJ2EEx_x.jar
```

Nos mostrará la siguiente pantalla y nos pedirá una serie de parámetros:

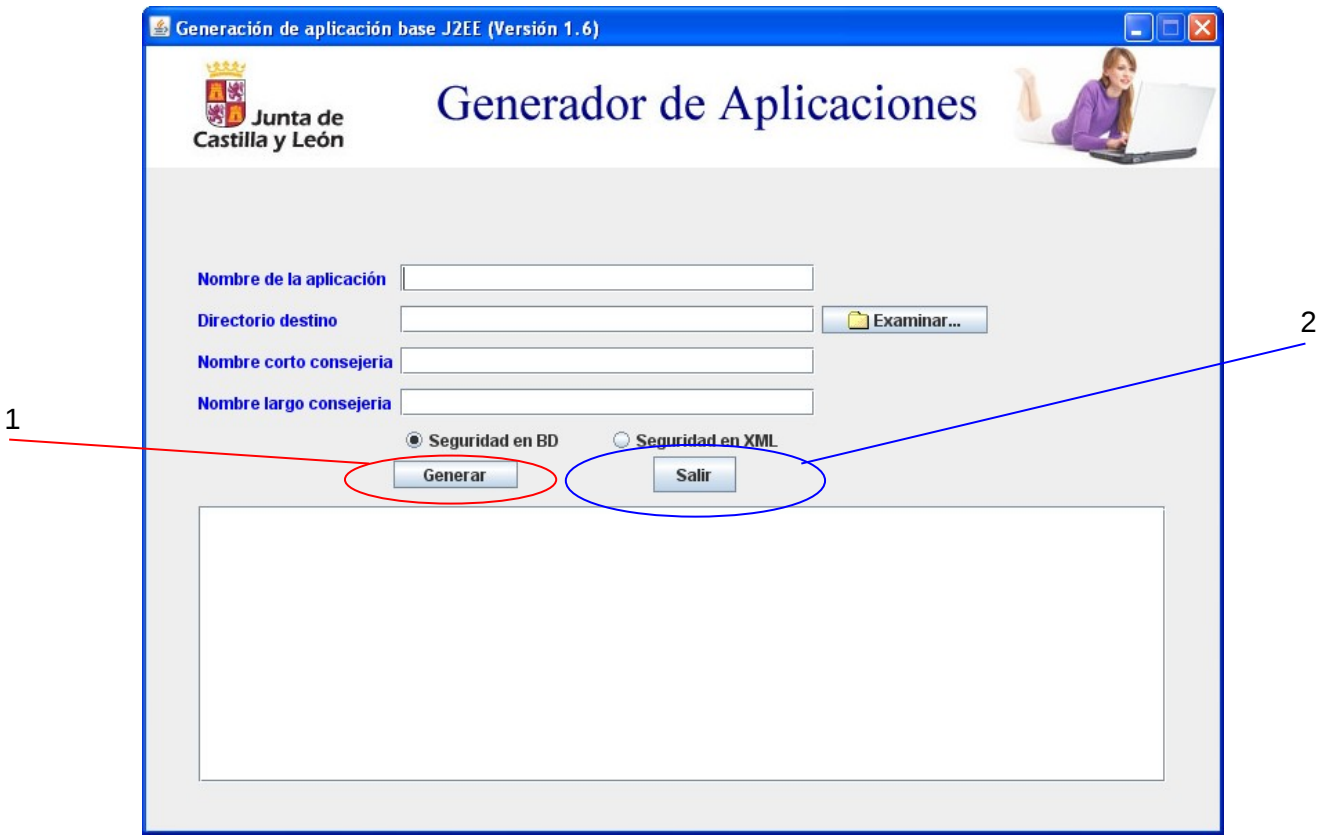


Figura 2: Generador de Aplicaciones J2EE

Parámetros necesarios (todos los campos son obligatorios):

- **Nombre de la aplicación:** *Nombre corto de 4 letras (Mayúsculas), utilizado como identificador de la aplicación que debe coincidir con el introducido en el sistema de seguridad.*
- **Directorio destino:** *Directorio donde queremos generar la aplicación. Deberá estar vacío.*
- **Nombre corto consejería:** *Iniciales de la consejería (minúsculas), necesario para el nombrado de los paquetes java del código fuente generado. Ej: cf, gss, ce, etc.*
- **Nombre largo consejería:** *Nombre completo, descriptivo para mostrarlo en las páginas de la aplicación*
- **Seguridad en BD:** *Incluir las librerías de acceso y ficheros necesarios del sistema de seguridad en la versión Base de Datos.*

- **Seguridad en XML:** *Incluir las librerías de acceso y ficheros necesarios del sistema de seguridad en la versión XML*

A continuación debemos pulsar el botón ***Generar***[1] y cuando el proceso haya acabado nos mostrará una ventana indicándolo.

4 PERSONALIZAR LA APLICACIÓN BASE

Una vez generada nuestra aplicación base es necesario revisar y adaptarla a nuestras necesidades. Para ello se deben configurar los siguientes apartados:

4.1 Copiar los ficheros de Configuración Externa.

Una vez ejecutada la Aplicación Base, para que podamos ejecutar nuestra aplicación de tal forma que encuentre los ficheros de configuración, esta nos los ha copiado en la ruta

datos/applications/dirtrabajo/nombreaplicacion/config/

Los ficheros que se encuentran en el directorio configExterna son una copia, los que realmente utiliza la aplicación son los que se han copiada a la ruta indicada, lo que hay que tener en cuenta a la hora de hacer modificaciones en dichos ficheros.

A la hora de desplegar la aplicación a un servidor de aplicaciones externo, hay que copiarse estos ficheros al servidor.

En el fichero de **ConfigurationRepository.xml** indicamos esta ruta mediante el parámetro:

<initialPath value="/datos/applications/dirtrabajo/nombreaplicacion/config/"/>

Si por algún motivo se desea cambiar esta estructura de directorios, basta con tocar antes de ejecutar la aplicación base en el fichero **GenerarAplicacionJ2EE.properties** la propiedad

Directorio trabajo donde se copian los ficheros de configuración

DIRECTORIO_TRABAJO=\\datos\\applications\\dirtrabajo\\

Respecto a la configuración de este fichero, hay que tener en cuenta, que en la biblioteca jcyl-configuracion, se crea un hilo para gestionar las configuraciones, hilo que no se puede eliminar de ninguna forma, sólo esperando a que pase el tiempo que se haya configurado en atributo contextTimeOut de la etiqueta context del **ConfigurationRepository.xml.**

Si este tiempo es muy pequeño (por defecto viene configurado a 10 segundos), no supone ningún problema, pero si se configura a un tiempo bastante grande (horas o días), tendremos como resultado que la aplicación no se puede descargar hasta que transcurra ese tiempo.

Por lo tanto se recomienda mantener este valor a valores pequeños.

4.2 Adaptaciones en el sistema de Logs.

Debe actualizarse el nombre y las rutas de los ficheros de logs que se desee crear en la aplicación.

Ficheros de configuración de los logs incorporados en la aplicación:

WEB-INF/config/log4jfw.properties: Define un APPENDER (Configuration) con las tareas internas

log.xml: Define 2 APPENDER:

- **Aplicación:** con los mensajes de aplicación. Se deja listo para usarse

- **Auditoria:** Se completa internamente con las acciones permitidas y prohibidas ejecutadas por el controlador Struts.

Se permite añadir cualquier otro appender específico: Subsistemas, Conexiones BD, invocaciones a otros servicios, etc

El sistema deja por defectos los logs en el directorio **WEB-INF\logs. ,** según la variable **${log.directory}.**

<param name="File" value="${log.directory}aplicacion.log" />

Igual que se han sacado los ficheros de configuración de la aplicación, es buena práctica hacer lo mismo con los logs, para facilitar el acceso y lectura de los mismos. Para realizar este cambio basta con indicar en el appender la nueva ruta, sin utilizar la variable $ {log.directory}, quedando de esta manera

<param name="File" value="/datos/applications/logs/nombreaplicacion/aplicacion.log" />

4.3 Propiedades de la aplicación. Cambios en app-config.properties

En este archivo se deben modificar los valores –sólo los valores, no los nombres- de las propiedades que vienen incluidas por defecto en el mismo.

4.3.1 Propiedades generales de la aplicación

Propiedad	Descripción	Ejemplo de Valor
SISTEMA	Debe especificarse la clave de 4 Caracteres que identifica la aplicación y que es la utilizada en el sistema de seguridad.	**GECE**
TITULO	Etiqueta de Título la aplicación que aparece en cabecera de las páginas	
ClaseConfiguracion	Clase que inicializa la configuración propia de la aplicación. Aparece rellena por defecto	
TIEMPO_REFRESCO_ LOGS_ SEGUNDOS	Tiempo en segundos para volver a refrescar el fichero de logs de la aplicación (log.xml)	**60**

4.3.2 Propiedades relativas al sistema de seguridad

Propiedad	Descripción	Ejemplo de Valor
SEGU_HOST	Maquina (Dirección IP, o DNS) donde reside el servidor OC4J que contiene los componentes de Seguridad (Acceso REMOTO)	**localhost**
SEGU_PORT	Puerto para acceder al servidor OC4J como administrador (Acceso REMOTO)	**3001**

SEGU_USER	Identificación ante el servidor OC4J (Acceso REMOTO)	**admin**
SEGU_PASSWORD	Palabra clave (Acceso REMOTO)	**Xxxx**
SEGU_APP_SEGURIDAD	Nombre de la aplicación en el servidor donde ha sido desplegado el componente	**SistSegu**
SEGU_POOL	Repositorio de Datos de Segu por defecto	**pool-name**
SEGU_REFRESCO	Indica si al crear una nueva Sesion hay que refrescar el HastTable de funciones accesibles. Valores posibles: true=Entrada rápida sin refrescar las funciones fase = Actualiza cada entrada en Sesion	**False**
TIPO_LOGIN_DEFECTO	Tipo de autenticación que va a utilizar la aplicación. Valores posibles: publico, password, certificado	**password**
USUARIO_PUBLICO	Cuenta de usuario usada para autenticarse en el caso de acceso público	**anonimo**
PASSWORD_PUBLICO	Contraseña usada para autenticarse en el caso de acceso público	**Anonimo**
TRIPLEDES	Clave del algoritmo TripleDes para Autenticación por certificado	
URL_AUTENTIFICACION_CERTIFICADO	URL del servidor de autentificación para validación con certificado	

4.3.3 Otras propiedades

Propiedad	Descripción	Ejemplo de Valor
CON_AUDITORIA	Activar/Destactivar la AUDITORIA DE ORACLE 9i. Valores posibles SI/NO	**NO**
NOMBRE_DATASOURCE	Nombre del pool DataSource utilizado	**jdbc/DEMOPoolDS**
MENU_ACCESIBLE	Incluye menú javascript o menú modo lista html Valores: SI : Menú Modo Lista NO : Menú Javascript	**NO**
MAX_INTENTOS_FALLIDOS	Máximo número de intentos fallidos	**3**
MINUTOS_PENALIZACION	Tiempo en minutos que se le penaliza por los intentos fallidos completados	**15**
QUARTZ_ONLOAD	Arrancar Planificador Quartz automáticamente (false, true)	**false**
QUARTZ_CRON	Instantes de ejecución del Planificador Quartz mediante expresiones UNIX	**30 * * * * ?**

4.3.4 Añadir propiedades a nuestra aplicación

Además pueden añadirse cuantos parámetros se desee. Posteriormente se podrá acceder a los mismos a través del método:

```
String valor = JCYLConfiguracion.getValor("<NOMBRE_PROPIEDAD");
```

4.4 Descriptor de controlador Struts. Archivo WEB-INF\XXXX-config.xml

En este archivo se deben definir los elementos form-bean y action a medida que se vaya construyendo la aplicación.

4.5 Definir el tipo de login a la aplicación

La aplicación base está preparada para aceptar distintos métodos de entrada a la aplicación.

El código está implementado en página **web/inicio.jsp** y se configura desde el fichero principal de propiedades (**app-config.properties**)

Las propiedades que hay que definir son:

- TIPO_LOGIN_DEFECTO: Tipo de login de entrada por defecto. Valores posibles: publico, password o certificado
  - **TIPO_LOGIN_DEFECTO=publico**
- USUARIO_PUBLICO: Nombre del usuario anónimo
- PASSWORD_PUBLICO: Password del usuario anónimo
- Si la aplicación tiene varias formas de acceso simultáneamente:
  - http://<servidor>/aplicacion/inicio.jsp?acceso=publico
  - http://<servidor>/aplicacion/inicio.jsp?acceso=password

4.5.1 Tipos de acceso a la aplicación.

En la siguiente tabla se determinan los distintos tipos de acceso y su forma de configuración:

Tipo de acceso	Descripción	Propiedades a definir
Usuario / Contraseña	Se trata del acceso clásico por usuario/contraseña	**TIPO_LOGIN_DEFECTO=password**
Acceso público o anonimo	Se inicia la sesión tomando como usuario, publico, las variables USUARIO_PUBLICO/ PASSWORD_PUBLICO	**TIPO_LOGIN_DEFECTO=publico** **USUARIO_PUBLICO: Nombre del usuario anónimo** **PASSWORD_PUBLICO : Password del usuario anónimo**
Acceso mediante Certificados digitales o eDNI	Se inicia la sesión mediante la validación por parte de la plataforma de forma corporativa del certificado digital del usuario, o su Dni electrónico. El sistema de almacenamiento del certificado (navegador o tarjeta) es indiferente.	**TIPO_LOGIN_DEFECTO=certificados** **TRIPLEDES=<Segun Plataforma>** **URL_AUTENTIFICACION_CERTIFICADO=<Según Plataforma>**

Usuario / Contraseña mas Certificados	Se permite elegir entre acceso usuario/pw y certificados en la misma pantalla.	**TIPO_LOGIN_DEFECTO=passcert** **TRIPLEDES=<Según Plataforma>** **URL_AUTENTIFICACION_CERTIFICADO=<Según Plataforma>**

4.6 Otras consideraciones

- No se puede hacer una llamada directa a una página *.jsp.
- Para las **aplicaciones de acceso público (NOVEDAD)**, solamente se requiere especificar las nuevas propiedades **TIPO_LOGIN_DEFECTO, USUARIO_PUBLICO y PASSWORD_PUBLICO.**
- En la clase que realice el **login** (la que sustituya a `es.jcyl.consejeria.NombreAplicacion.actions.NombreAplicacionLoginAction`) no debe alterarse el código que se presenta; sin embargo puede añadirse el código que se estime necesario.
- Se incorpora en código fuente la clase **JCYLConfiguracion.java** dentro del paquete es.jcyl.framework. Esta clase que se distribuía en versiones anteriores dentro de **jcylfwx_x.jar** se ofrece para consultar los pasos seguidos en arranque básico de una aplicación y el uso de las capacidades del componente ConfigurationRepository. Se recomienda no cambiar.

5 DESCRIPCIÓN DE LOS COMPONENTES PRINCIPALES

5.1 Características Básicas

Una vez ejecutada la aplicación y hechos todos los cambios indicados, podemos entrar en JDeveloper y abrir el workspace generado. Este se encuentra dentro del directorio de la aplicación\fuentes.

Existen un fichero .jws (workspace) asociado a las distintas versiones de JDeveloper (9.0.5, 10.1.2 o 10.1.3). Es aconsejable renombrar el fichero .jws y .jpr eliminando la etiqueta de la versión del JDeveloper desde dentro de la herramienta de desarrollo.

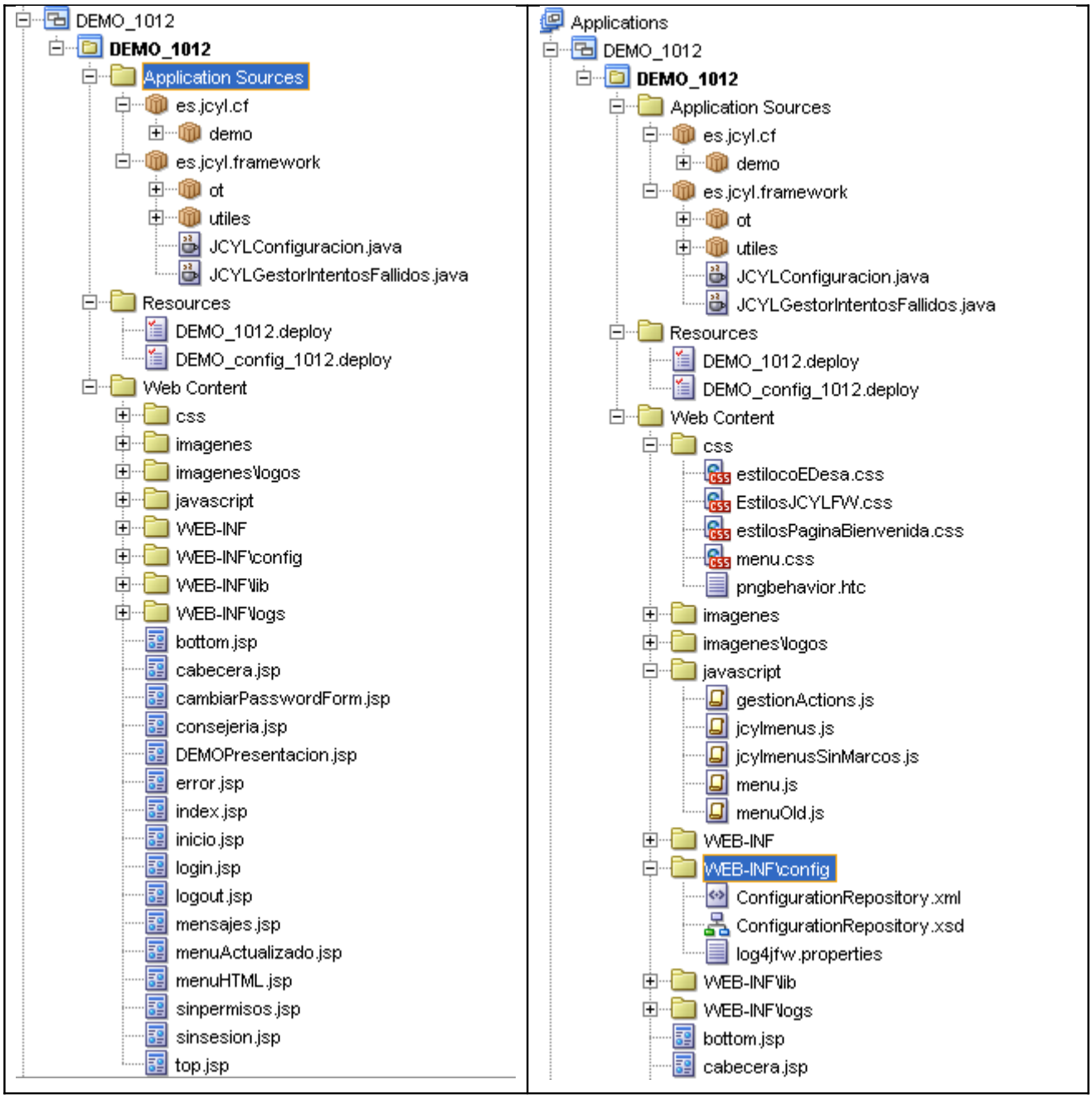


Figura 4: Proyecto Generado

La aplicación está lista para ejecutar.

Si ejecutamos entraremos en la pantalla de login y después de introducir un usuario/password de la aplicación nos mostrará la pantalla de menús (si estos ya están definidos en la B.D.).

Un ejemplo de ejecución es el siguiente:

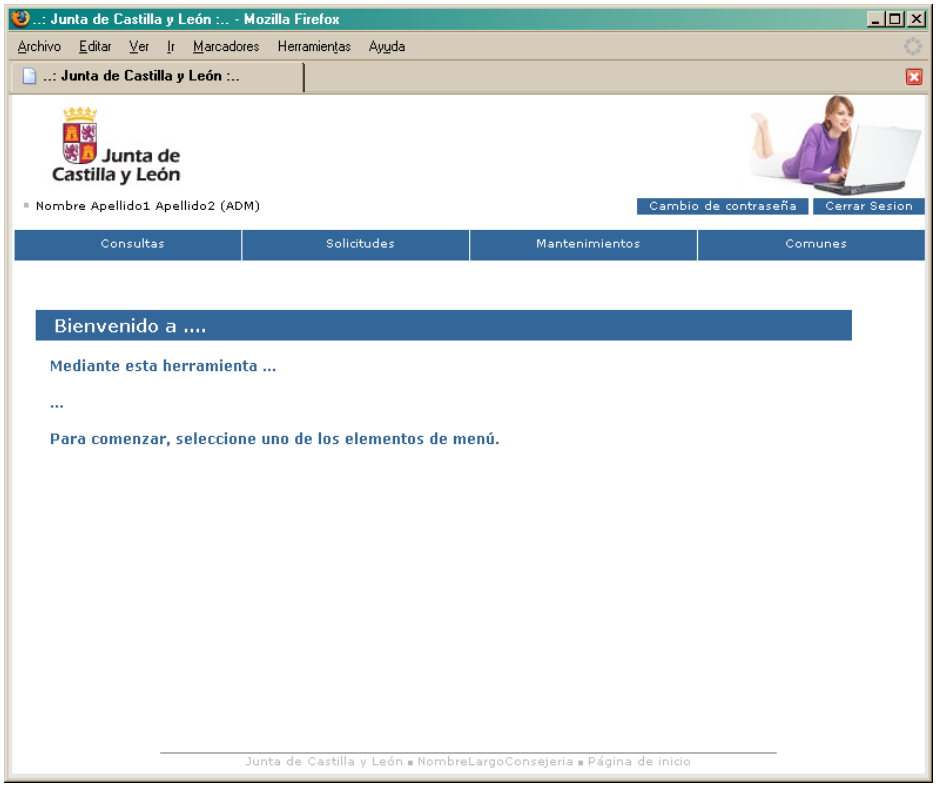


Figura 5: Aplicación Generada

5.1.1 Vista de la aplicación base

5.1.1.1 Relación de las Páginas JSP

Fichero	Propósito
bottom.jsp	Recorte de código del fin de una página cualquiera de tu aplicación. Distingue entre la parte accesible y no accesible
cabecera.jsp	Recorte de código con la tabla e imagen que compone la cabecera de la página
cambiarPasswordForm.jsp	Página de negocio de la aplicación con imagen del cambio de contraseña. Se trata de buen ejemplo para ver cómo organizar los formularios de nuestra aplicación con la nueva estructura. Maquetada con estilos y accesible AA
consejeria.jsp	Código donde se muestra el usuario conectado, extraído de la sesión, y los botones de cerrar sesión y cambio de contraseña. Común para la parte ACCESIBLE Y NO_ACCESIBLE.
error.jsp	Página por defecto para mostrar errores generales
index.jsp	Página principal de la aplicación donde se ubica el marco contenido de la versión NO ACCESIBLE. Pretende ser la página que contenta la complejidad del menú e imágenes de cabecera y pie.

	En el marco contenido irán los formularios simples simplificando del tráfico y tiempo de carga. Página de Bienvenida para la versión ACCESIBLE
inicio.jsp	Nueva página por defecto de inicio que determina la forma de entrada en la aplicación en función del fichero de propiedades
login.jsp	Página para capturar usuario/contraseña
logout.jsp	Nueva página de despedida de la aplicación
mensajes.jsp	Página por defecto para mostrar mensajes de propósito general
menuActualizado.jsp	Código donde personalizar la apariencia y colocación del menú javascript.
MenuHTML.jsp	Código java (scriptlet) que presenta el menú HTML accesible y con datos del sistema de seguridad.
XXXXPresentacion.jsp	Nueva página de presentación. Se recomienda adaptarla a los propósitos de la aplicación final
sinpermisos.jsp	Página por defecto para mostrar el mensaje que muestra que no existe permiso para ejecutar la acción. Esto ocurre cuando se pretende ejecutar una acción que no está inventariada en el sistema de seguridad para el grupo de usuarios predeterminado.
sinsesion.jsp	Página por defecto para mostrar el mensaje que muestra que no existe sesión creada o que la misma ha caducado.
top.jsp	Recorte de código del inicio de una página cualquiera de tu aplicación. Distingue entre la parte accesible y no accesible

5.1.1.2 Hojas de estilo: CSS

Fichero	Propósito
estilosPaginaBienvenida.css	Estilos usados únicamente en la página de bienvenida
menu.css	Estilos usados en el menú Javascript. Descargados y personalizados de la distribución de Tigra Menu v2.0
EstilosJCYLFW.css	Relación de estilos utilizados en el esqueleto base de aplicación y adaptación de los estilos SIAU

5.1.1.3 Principales estilos de EstilosJCYLFW.css

Actuación	JSP afectadas	Estilos afectado (EstilosJCYLFW)
Personalización del banner de cabecera	web/cabecera.jsp imágenes/fondoBanner.png	.midCabecera .midCabecera img .midCabecera h1 .cuadroInfo .conexion
Tamaño del marco contenedor principal	Web/top.jsp	.encuadre

	Web/index.jsp Web/bottom.jsp	.contenidoNoAccesible .contenidoAccesible .cuadroContenedor
Pie de página	Web/bottom.jsp	#footer .lineafooter
Tamaño del area de los formularios	Web/index.jsp	.contenidoNoAccesible .contenidoAccesible #cuadroContenedor
Tamaño del menú accesible y apariencia	Web/index.jsp Web/menuHTML.jsp	.menuAccesible .marcoContenidoAccesible
Formulario de Login	Web/login.jsp	#titulocajaformulario #tituloOpcion #cajaformulario
		Otras hojas de estilo afectadas
Tamaño y características del menú Javascript	Web/menuactualizado.jsp Web/index.jsp	**menu.css**
Pagina de Bienvenida	Web/XXXXPresentacion.jsp	**estilosPaginaBienvenida.css**

5.1.1.4 Menú Javascript

En la versión no accesible del esqueleto base de aplicación se ha migrado el menú javascript. Se ha incorporado **Tigra Menu 2.0 versión FREE**

La descripción y documentación completa del componente se puede consultar en la Web: http://www.javascript-menu.com/

Su adaptación al esqueleto base consiste en:

- Incorporar **javascript\menu.jsp**
- Incorporar **css/menu.css**
- Colocar el menú en pantalla, está implementado en la página **menuactualizado.jsp**
- Nueva técnica para aplicarle los contenidos al menú:
  - En **XXXXLoginAction.java** se obtiene el menú de Segu mediante las siguientes instrucciones:

```
ArrayList FuncionesMenu = ejb_aut.getAutorizacion().getEstructura(nombreAplicacion, usuarioSEGU.getRol());

session.setAttribute("JCYLArrayFunciones", FuncionesMenu);
```

- **es.jcyl.framework.utiles.JCYLSeguTigra.java** : Se incorpora un método que serializa para TigraMenu extrayendo sus datos del atributo "JCYLArrayFunciones"
- **web/menuactualizado.jsp** : se invoca a la serialización y se configuran los parámetros de posición y tamaño

Para ver los detalles consultar el código fuente de los ficheros indicados.

5.1.1.5 Menú HTML

En la **versión accesible** del esqueleto base de aplicación incorpora una lista HTML implementar el menú.

Su adaptación al esqueleto base consiste en:

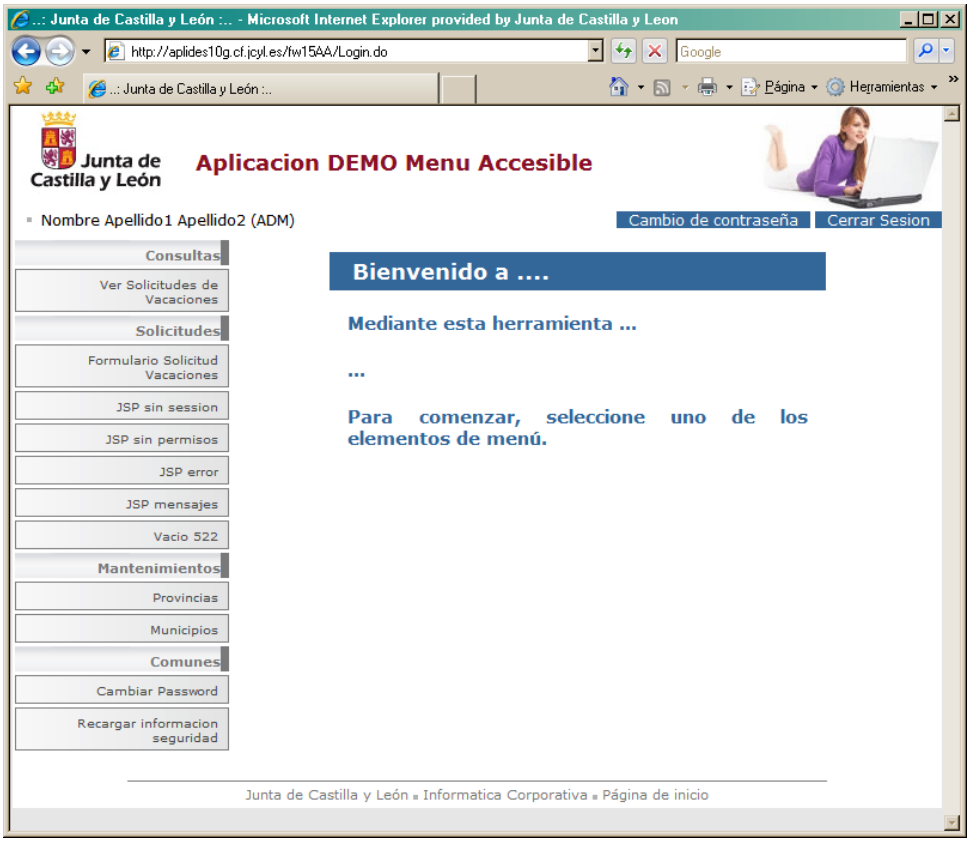
- Nueva técnica para aplicarle los contenidos al menú:
  - En **XXXXLoginAction.java** se obtiene el menú de Segu mediante las siguientes instrucciones:

```
ArrayList FuncionesMenu = ejb_aut.getAutorizacion().getEstructura(nombreAplicacion, usuarioSEGU.getRol());

session.setAttribute("JCYLArrayFunciones", FuncionesMenu);
```

  - **es.jcyl.framework.utiles.JCYLSeguHTML** : Se incorpora un método que serializa el menú obtenido desde seguridad a una lista de opciones en formato HTML
  - **web/ menuHTML.jsp** : se invoca a la serialización **(es.jcyl.framework.utiles.JCYLSeguHTML.getMenuHTML(FuncionesMenu))** y lo presenta en la página destino.
- Incorpora en **css/EstilosJCYLFW.css** un conjunto de estilos predefinidos que le dan la apariencia final. Las clases son:

Estilo	Descripción
.menuAccesible	Marco con el menú a la izquierda accesible
.menuAccesible UL	Apariencia de la etiqueta UL dentro del marco **menuAccesible**
.menuAccesible LI	Apariencia de la etiqueta LI dentro del marco **menuAccesible**
.menuAccesible a	Apariencia de los enlaces del menú
.menuAccesible a:hover	Apariencia de los enlaces del menú resaltado
.menuAccesible .menuNivel_1	Apariencia de los elementos de menú de primer nivel
.menuAccesible .menuNivel_2	Apariencia de los elementos de menú de segundo nivel
.menuAccesible .menuNivel_3	Apariencia de los elementos de menú de tercer nivel
.menuAccesible .menuNivel_4	Apariencia de los elementos de menú de cuarto nivel



Ejemplo de pantalla con menú HTML accesible AA

Para ver los detalles consultar el código fuente de los ficheros indicados.

5.2 Nueva maquetación con criterios de accesibilidad

Se ha reestructurado todo el esqueleto. Este trabajo ha consiste en diseñar todas las páginas (jsp y html) incluidas en el esqueleto base para aplicarles el aspecto del SIAU y realizando una maquetación basada en CSS y CAPAS.

El objetivo perseguido en esta maquetación es ofrecer 2 versiones a elegir en nuestra aplicación final:

Versión NO Accesible que proporciona:

- Menú Javascript TigraMenu 2.0
- Marco IFRAME donde cargar las páginas de negocio y optimizar el rendimiento de la aplicación
- Esta versión es compatible con versiones anteriores del esqueleto base

Versión Accesible AA:

- Incluye menú formato HTML en lista ubicado en la parte izquierda de la pantalla
- Las paginas se muestran completas sin Marco IFRAME
- Esta versión es mas compatible con la apariencia de los portales institucional y temático http://www.jcyl.es

Su implementación ha consistido en:

Definir una nueva propiedad en el fichero de propiedades que gobiernan el proceso:

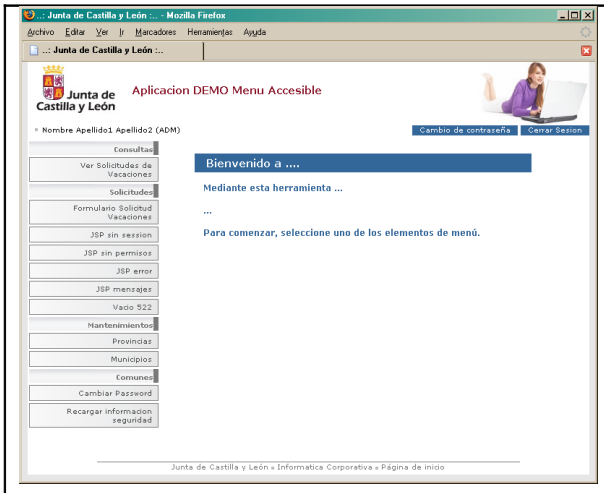
Propiedad	Descripción	Ejemplo de Valor
MENU_ACCESIBLE	**Incluye menú javascript o menú modo lista html** **Valores:** **SI : Menú Modo Lista** **NO : Menú Javascript**	**NO**

Se introduce ese valor en el contexto de la aplicación al iniciar la aplicación.

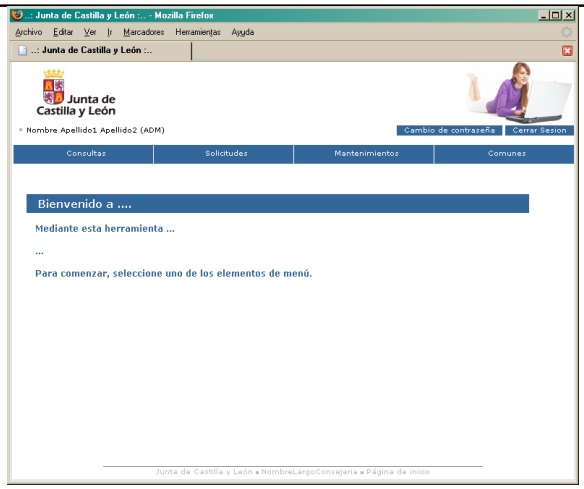
En **es.jcyl.nombreconsejeria.nombreaplicacion.config.NombreAplicacionConfigApp.inicializar** se añaden las siguientes líneas :

```
String menu = es.jcyl.framework.JCYLConfiguracion.getValor("MENU_ACCESIBLE");
  if (menu!=null && menu.equals("SI"))
        application.setAttribute("MENU_ACCESIBLE", "SI" );
   else
        application.setAttribute("MENU_ACCESIBLE", "NO" );
```

Se usa esa variable para configurar las 2 apariencias disponibles:



Versión accesible : Menú Lista sin marco



Versión accesible : Menú Lista sin marco

Los elementos comunes como: Formulario de Login, cambio de password, páginas de mensajes, bienvenida, etc. están diseñados para ser accesible AA

5.3 Personalización en la secuencia de arranque

La versión actual del proyecto ofrece la clase es.jcyl.framework.JCYLConfiguracion.

Esta clase se estaba distribuyendo empaquetada en versiones anteriores al jcylfw-1.3.jar. La secuencia de pasos de arranque, **la cual se recomienda encarecidamente no alterar,** está implementado en el método **public static void inicializar(Servlet servlet)**.

La intención de esta mejora es hacer la secuencia de arranque más transparente a los desarrolladores y mostrar el caso práctico de uso del nuevo componente **GestorConfiguracion.**

Para añadir operaciones especificas de inicialización y finalización global de la aplicación se recomienda seguir usando la clase XXXXConfigApp

5.3.1 Ejecución de código al inicio o fin de la aplicación. Clase XXXConfigApp

La aplicación base generada mediante esta herramienta nos define un entorno en el cual podemos definir qué programa o código queremos que se ejecute tanto iniciar la aplicación como justo antes de la finalización del mismo.

Para ese propósito se ha definidos los métodos **inicializar** y **finalizar** de la clase *es.jcyl.consejeria.nombreaplicacion.config.NOMBREAPLICACIONConfigApp.java.*

El siguiente ejemplo ilustra como incluir código a ejecutar al inicio de la aplicación. Este mismo ejemplo lo detallaremos dentro del ANEXO II.

```
public void inicializar(ServletContext application) throws Exception {
  logger = JCYLConfiguracion.getLogger(JCYLConfiguracion.LOG_APLICACION);
  logger.debug(getClass().getName()+"  inicializar() ");

   // Comprobar si generar versión accesible
  String menu = es.jcyl.framework.JCYLConfiguracion.getValor("MENU_ACCESIBLE");
  if (menu!=null && menu.equals("SI"))
         application.setAttribute("MENU_ACCESIBLE", "SI" );
   else
         application.setAttribute("MENU_ACCESIBLE", "NO" );

  /* Cargar lista de Provincias y almacenarlas en el contexto de la aplicación */
  DEMOConfigLN ln = new DEMOConfigLN();
  java.util.ArrayList  provincias_cyl = ln.RecuperarListaProvincias();
  application.setAttribute("PROVINCIAS_CYL", provincias_cyl);
  logger.debug("Guardardo en Aplicacion Provincia :" +provincias_cyl.toString())
  provincias_cyl = null;
/* Cargar lista de Consejerias y almacenarlas en el contexto de la aplicación */
  java.util.ArrayList  consejerias = ln.RecuperarListaConsejerias();
  application.setAttribute("CONSEJERIAS", consejerias);
  consejerias = null;
  ln = null;
}
```

5.4 Planificador de tareas

Se ha incluido en esta nueva versión, un planificador de tareas más completo y potente, sin eliminar la opción del gestor reducido de tareas existente en versiones anteriores, pero siempre que sea posible, se recomienda utilizar esta nueva opción, por su potencia y poco consumo de recursos.

5.4.1 JCYLPlanificador

Este componente se ha diseñado tratando de reducir el trabajo del programador a la hora de generar las nuevas tareas. En este apartado se va a comentar la forma de crear una nueva tarea para que pueda ser utilizada por el planificador.

Toda tarea que se cree debe extender de la clase `JCYLTarea` e implementar los dos métodos abstractos de esta clase (`ejecutar()` y `setParametros()`) para realizar las acciones para las que se cree la tarea.

- Método `ejecutar()`

  En este método se lleva a cabo todo el trabajo de la tarea. Se debe tener en cuenta que si la tarea es interrumpible debe ser el programador el que establezca puntos de control en los que se compruebe si se ha solicitado la cancelación de la tarea, en cuyo caso se anotará en el log y no se continuará con la ejecución del método. Si la tarea no es interrumpible, aunque se definan estos mismos puntos de control

nunca se activarán puesto que los métodos de la clase padre consideran esta posibilidad.

- Metodo `setParametros()`
  En este método es donde se debe incluir el código necesario por la tarea para hacer uso de los parámetros definidos en el fichero de propiedades. Si no se necesita utilizar parámetros este método puede estar vacío, pero siempre hay que implementarlo.

Una vez creada una nueva tarea, y por tanto implementados los métodos anteriores, se debe incluir la información de la tarea en el fichero de propiedades del planificador para que éste la pueda utilizar.

El **JCYLPlanificador.properties** se encuentra en la ruta donde están los ficheros de configuración externa, y en él se definen los siguientes aspectos:

- Activar / Desactivar el planificador
- Definir la lista de tareas (clases java) con sus aspectos de ejecución:
  - Intervalo de tiempo entre ejecuciones (unidades de tiempo = minutos)
  - Tarea activa o desactiva por defecto
  - Tarea interrumpible o no
  - Nº máximo de ejecuciones
  - Relación de parámetros que necesite le tare

Si se ha decidido que la tarea puede ser interrumpida durante su ejecución se deben insertar puntos de control a lo largo del método ejecutar de forma que se comprueba si se ha solicitado la cancelación de la tarea y si es así se registra en el log y se sale del método permitiendo al planificador continuar con otras tareas. Para comprobar si se ha indicado cancelar se puede utilizar el método de la clase padre `comprobarSalir()`. A continuación se muestra el código ejemplo que se puede utilizar

```
if (comprobarSalir()) {
     JCYLPlanificadorTareas.log("Tarea cancelada");
     setCancelada(true); // !!!OJO!!! esto es fundamental
                                 // puesto que si no se puede bloquear el
                                 // planificador ya que está a la espera
                                 // que la tarea finalice correctamente
                                 // o bien que se haya indicado que la
                                 // cancelación se ha ejecutado
     return;
 }
```

Como se puede ver, también se debe indicar que la tarea ha finalizado para permitir al planificador continuar con otras ejecuciones puesto que si no quedará a la espera de la finalización de la tarea y esto no se producirá nunca.

Cuando se llama al método cancelar se detiene la ejecución del planificador hasta que la tarea haya finalizado o se haya llegado a un punto de control y se haya cancelado.

Ejemplo de una Tarea:

```
public class TareaEjemplo extends JCYLTarea {
    public TareaEjemplo() {
        super();
    }
    protected void ejecutar() {
        JCYLPlanificadorTareas.log("Tarea, " + 3 + ", " + this + ", timed " + 1);
        try {
            Thread.sleep(40000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        if (comprobarSalir()) {
            JCYLPlanificadorTareas.log("Tarea cancelada");
            setCancelada(true);
            return;
        }
        try {
            Thread.sleep(40000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        JCYLPlanificadorTareas.log("Tarea, " + 3 + ", " + this + ", finalizada");
    }
    public void setParametros(String[] parametros) {
    }
}
```

5.4.2 Planificador Quart Scheduler

Quartz es un sistema de programación de tareas que puede ser integrado en cualquier aplicación J2EE o J2SE. Se puede utilizar para crear programaciones simples o complejas, para ejecutar decenas, cientos o miles de trabajos, bajo componentes estándar Java lo que permite ejecutar prácticamente cualquier cosa.

Sus principales características son:

- Es válido tanto para aplicaciones J2EE como J2SE.
- Posee una planificación muy flexible y configurable de tareas, pudiendo ser su periodicidad diaria, mensual, en base a unos días de la semana, en base a un calendario predefinido…
- Posibilidad de uso de listeners y control de fallos.

- Posee un completo API, con diferentes tipos de tareas, de triggers, de Schedulers…

Quartz es de libre uso bajo licencia Apache 2.0. (http://www.quartz-scheduler.org/)

Para poder utilizar este planificador se ha incluido en el generador de aplicaciones:

- Se ha añadido al proyecto la librería **quartz-all-1.6.6.jar**
- Se han añadido dos propiedades al fichero *app-config.properties*

#Arrancar Planificador Quartz automáticamente (true/false)
QUARTZ_ONLOAD=false
#Instantes de ejecución del Planificador Quartz
QUARTZ_CRON = 30 * * * * ?

En este última propiedad se definen los instantes en los que la tarea va a ser ejecutada mediante expresiones del tipo cron de Unix:

segundos, minutos, horas, días del mes, meses, días de la semana, [año]

En el ejemplo (*0 0 16 * * ?*), la tarea está programada para todos los días a las 4 de la tarde.

Información detallada en http://quartz-scheduler.org/docs/tutorials/crontrigger.html

- Se ha desarrollado una clase que implementa el trabajo (*org.quartz.Job*).

```
package es.jcyl.cf.demo.planificador;

import java.util.Map;
import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;

public class PlanificadorJob implements Job
{
public void execute(JobExecutionContext context) throws JobExecutionException
{
   Map dataMap = context.getJobDetail().getJobDataMap();
   PlanificadorJob job =(PlanificadorJob)dataMap.get("schedulerTask");
    //Construimos la tarea programada que queremos que se lance
    //job.tareaProgramada();
}
}
```

- Se ha desarrollado una clase que implementa el Plugin de Struts e instancia el trabajo

```
package es.jcyl.cf.demo.planificador;
```

```
import es.jcyl.framework.JCYLConfiguracion;
import es.jcyl.cf.demo.config.DEMOConfigApp;
import java.text.ParseException;
import java.util.Map;
import javax.servlet.ServletException;
import org.apache.struts.action.ActionServlet;
import org.apache.struts.action.PlugIn;
import org.apache.struts.config.ModuleConfig;
import org.quartz.CronTrigger;
import org.quartz.JobDetail;
import org.quartz.Scheduler;
import org.quartz.SchedulerException;
import org.quartz.impl.StdSchedulerFactory;

public class PlanificadorPlugin implements PlugIn {
    Scheduler scheduler;

          public void init(ActionServlet servlet, ModuleConfig config) throws
ServletException {
       PlanificadorJob task = new PlanificadorJob();
       String startOnLoad = JCYLConfiguracion.getValor("QUARTZ_ONLOAD");
       //specify your sceduler task details
       JobDetail job = new JobDetail();
       job.setName("jobName");
       job.setJobClass(PlanificadorJob.class);
       Map dataMap = job.getJobDataMap();
       dataMap.put("schedulerTask", task);
       try {
          //configure the scheduler time
          CronTrigger trigger = new CronTrigger();
          trigger.setName("runMeJobTesting");
          //trigger.setCronExpression("30 * * * * ?");
          trigger.setCronExpression(JCYLConfiguracion.getValor("QUARTZ_CRON"));
          //schedule it
          scheduler = new StdSchedulerFactory().getScheduler();
          if (startOnLoad != null && startOnLoad.equals(Boolean.TRUE.toString())){
             scheduler.start();
             scheduler.scheduleJob(job, trigger);
             DEMOConfigApp.logger.info(" ** Quartz arrancado");
          }
       }catch(ParseException e){
         e.printStackTrace();
       }catch(SchedulerException e){
         e.printStackTrace();
```

```
    }
  }

  public void destroy() {
    try {
       scheduler.shutdown();
    } catch (SchedulerException ex) {
    ex.printStackTrace();
    }
    scheduler = null;
   }
  }
```

- Mapear el nuevo Plugin en el fichero de configuración de Struts de nuestra aplicación:

  <plug-in className="es.jcyl.cf.demo.planificador.PlanificadorPlugin" />

- Para que al hacer un nuevo despliegue no continúe el hilo del planificador corriendo, sin obligar a detenerlo reiniciando el contenedor, se ha agregado el método *finalizar()* en la clase NombreAplicacionCo*nfigApp.java*:

```
public void finalizar() throws Exception{
        // Paramos el hilo del panificador para que no siga corriendo
        Scheduler scheduler =new StdSchedulerFactory().getScheduler();
        if (scheduler.isStarted()) {
            scheduler.shutdown();
            logger.info(" ** Quartz parado");
        }

       logger.debug(getClass().getName()+"  finalizar() ");
}
```

- Consideraciones a tener en cuenta

Para que el planificador arranque, es necesario inicializar la aplicación tras un nuevo despliegue. Esto se puede conseguir de dos modos:

  - Solicitando reiniciar el contenedor en la solicitud de despliegue; método no recomendado.
  - Poniendo la URL de la aplicación en el navegador (sin necesidad de validarse); método recomendado.

Se cierra el hilo del planificador automáticamente al hacer un nuevo despliegue.

Se ha comprobado que consume pocos recursos.

Para más información consultar **APPBASE_Quartz-Scheduler-1.0.pdf**

5.5 Sistema de Log Estándar (log4j)

La aplicación base incorpora un sistema de Log basado en el componente **log4j.**

La primera y una de las mayores ventajas de cualquier API de logging sobre el tradicional **System.out.println** es la capacidad de habilitar y deshabilitar ciertos logs, mientras otros no sufren ninguna alteración. Esto se realiza categorizando los mensajes de logs de acuerdo al criterio del programador.

Log4J tiene por defecto 5 niveles de prioridad para los mensajes de Log:

- **DEBUG**: Se utiliza para escribir mensajes de depuración, este log no debe estar activado cuando la aplicación se encuentre en producción.
- **INFO**: Se utiliza para mensajes similares al modo "verbose" en otras aplicaciones.
- **WARN**: Se utiliza para mensajes de alerta sobre eventos que se desea mantener constancia, pero que no afectan el correcto funcionamiento del programa.
- **ERROR**: Se utiliza en mensajes de error de la aplicación que se desea guardar, estos eventos afectan al programa pero lo dejan seguir funcionando, como por ejemplo que algún parámetro de configuración no es correcto y se carga el parámetro por defecto.
- **FATAL**: Se utiliza para mensajes críticos del sistema, generalmente luego de guardar el mensaje el programa abortará.

Adicionalmente a estos niveles de log, existen 2 niveles extras que solo se utilizan en el archivo de configuración, estos son:

- **ALL**: este es el nivel más bajo posible, habilita todos los logs.
- **OFF**: este es el nivel más alto posible, deshabilita todos los logs.

5.5.1 Uso del sistema de Log en la aplicación base

En toda clase en la que deseemos usarlo debemos importar la clase:

```
import es.jcyl.nombreconsejeria.nombreaplicacion.config.NOMBREAPLICACIONConfigApp;
```

Cuando queramos invocarlo debemos hacerlo de la siguiente manera:

```
NOMBREAPLICACIONConfigApp.logger.error(mensaje);
```

Error , puede sustituirse por debug, info, etc..

También podemos añadir todos los appenders a mayores que queramos

Para más información sobre cómo usar el componente **log4j,** consultar el documento log4j.doc.

5.6 Autenticación mediante certificado digital.

Con esta modificación las aplicaciones generadas con el framework permiten el logado de los usuarios mediante certificado digital. El sistema de almacenamiento del certificado (navegador o tarjeta) es indiferente.

Los tipos de certificados admitidos son aquellos con los que la Junta de Castilla y León tiene acuerdo. De momento FNMT.

5.6.1 Requisitos previos a tener en cuenta

5.6.1.1 Solicitar acceso a la plataforma de Firma

El primer paso es ponernos en contacto con el administrador de la plataforma corporativa de firma para que nos de de alta la aplicación en el servidor de autenticación.

Debemos de proporcionar los siguientes parámetros a la plataforma:

Nombre de Aplicación: 4 Dígitos con el nombre de aplicación

URL de destino: Página de destino que se debe redirigir la plataforma cuando la autenticación se finalice con éxito

El sistema nos contestará con:

Clave Tripledes: Este parámetro contiene la clave triple des necesario para codificar la información intercambiada entre nuestra aplicación y el servidor de autenticación. El valor nos será proporcionado por el administrador de la plataforma de firma corporativa.

URL con el servidor de autenticación.

Ambos parámetros se recomienda que vayan especificados en el fichero de propiedades de la aplicación. Ver propiedades TRIPLEDES del fichero **app-config.properties**.

5.6.1.2 Usuario de SEGU con DNI Y NIF

La entrada segura a una aplicación requiere una fase de **autenticación** y otra de **autorización**. La primera se realiza mediante la plataforma de firma y el certificado digital. La autorización ha de hacerse mediante el **sistema de seguridad corporativo**.

Para poder resolver esta autorización se requiere que en el repositorio de SEGU esté creado un usuario con acceso nuestra aplicación y que tenga como **obligatorios los campos DNI y NIF** recuperados del Certificado Digital. Es mediante estos campos por los que se relaciona el certificado con repositorio de SEGU.

5.6.2 Funcionamiento

En el momento en el que se invoca a la aplicación esta pasa el control al servidor de autenticación, el cual se encarga de solicitar el certificado y validarlo. Después se devuelve el control a la aplicación indicando el resultado de la validación.

5.7 Ocultación de código JSP

Una práctica muy interesante en ocultar fragmentos de código en nuestras páginas JSP en función de la **información de seguridad** como: Código de Usuario, Rol de acceso, funciones asociadas, etc.

Esta función es posible utilizando adecuadamente la librería de tags de lógica (logic) que proporciona struts. Mediante esta librería nos permite mostrar u ocultar información y controles en las páginas web a determinados usuarios.

Como ejemplos de esta funcionalidad se adjuntan los siguientes fragmentos de código:

5.7.1 Mostrar información a un determinado usuario

Este fragmento de código muestra el botón eliminar si el usuario es operador:

```
<logic:equal name="<%=es.jcyl.framework.JCYLConfiguracion.NOMBRE_ATRIBUTO_USUARIO%>"
property="usuario.c_usr_id" value="operador" >
 <html:submit value="Eliminar"/>
</logic:equal>
```

5.7.2 Mostrar información a un determinado rol

Este fragmento de código muestra el botón nuevo si el rol es ADM:

```
<logic:equal name="<%=es.jcyl.framework.JCYLConfiguracion.NOMBRE_ATRIBUTO_USUARIO%>"
property="usuario.rol" value="ADM">
  <html:submit value="Nuevo"/>
</logic:equal>
```

5.7.3 Mostrar información a un rol que contiene una operación determinada

Este fragmento de código muestra el botón nuevo si el rol es ADM y se permite la operación APPUsuarioAl:

```
<logic:equal name="<%=es.jcyl.framework.JCYLConfiguracion.NOMBRE_ATRIBUTO_USUARIO%>"
property="usuario.rol" value="ADM">
  <logic:match   name="<
%=es.jcyl.framework.JCYLConfiguracion.NOMBRE_ATRIBUTO_FUNCIONES%>"
property="GELI_ADM" value="APPUsuarioAl">
    <html:submit value="Nuevo"/>
  </logic:match>
</logic:equal>
```

5.7.4 Obtener el rol del usuario y mostrar información a un rol que coincida con el del usuario y que contiene una operación determinada

Este fragmento de código muestra el botón nuevo si el usuario tiene un rol que permite la operación APPUsuarioAl:

```
<% es.jcyl.framework.JCYLUsuario miusr=
(es.jcyl.framework.JCYLUsuario)session.getAttribute(es.jcyl.framework.JCYLConfiguracion.NOMBR
E_ATRIBUTO_USUARIO);
String rol = miusr.getUser().getRol();
%>
<br>
  Mi Rol es : <%=rol%> ...
<br>

<logic:match name="<%=es.jcyl.framework.JCYLConfiguracion.NOMBRE_ATRIBUTO_FUNCIONES%>"
property="<%=rol%>"
```

```
value="APPUsuarioAl">
  <html:submit value="Nuevo"/>
</logic:match>
```

5.8 Validaciones basadas en Struts.

A partir de la versión 1.2 se han implementado los cambios necesarios en la **Aplicación Base** para poder utilizar las posibilidades de validación avanzada de formularios mediante Struts. Éste sistema de validación (**Struts Validator**) ya estaba disponible en la versión de Struts utilizada en la aplicación base (Struts 1.1) por lo que el siguiente apartado se encargará únicamente de documentar a grandes rasgos como usarlo.

Para más detalles de las capacidades de **Struts Validator** se recomienda consultar el documento de ***Struts in Action*** apartado *4.9 Configuring the Struts Validator* y en el *apartado 12 Validating user input*, se van describir los pasos a realizar para implantar dicha solución.

5.8.1 Pasos para aplicar la validación basada en Struts

Se van describir los pasos a realizar para implantar dicha solución para la validación de campos tanto en la parte cliente como en la parte servidora, ambas validaciones son independientes por lo que en los puntos se indicaran cuales pertenecen a la **validación cliente**[1] y cuales a la **validación de servidor**[2] y cuando no se indique nada es que pertenecen a las dos.

5.8.2 Actuaciones en los XML con las reglas de validación

Los pasos a seguir añadir una validación basada en reglas con Struts son los siguientes:

1. Añadir en /WEB-INF los ficheros **validator-rules.xml** y **validation.xml** facilitados con la distribución de la **aplicación base 1.2 o posteriores**

   Nota: Para obtener estos ficheros se debe generar una aplicación base vacía en un directorio temporal con el mismo nombre de aplicación que la que se está desarrollando y obtener los ficheros indicados.

2. Del fichero **validation.xml** eliminar, si no se desea utilizar, la información relacionada con las validaciones de los formularios LoginForm (**<form name= "LoginForm"> … </form>**) y CambioPasswordForm (**<form name= "CambioPasswordForm"> … </form>**). Se pueden consultar para ver cómo está implementado en la aplicación base.

3. Insertar en **/WEB-INF/*nombreAplicación*-config.xml** inmediatamente antes del tag **</struts-config>** se debe introducir el tag siguiente:

   ```
   <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
   <set-property
      property="pathnames"
      value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
   </plug-in>
   ```

4. Añadir en el validation.xml una sección <formset> por locale (idioma) como formularios se quieran validar introduciendo entre los tag <form-validation> <form-validation> un código equivalentes al siguiente:

```
<formset>
    <form name="nombreFormulario">
        <field     property="nombrePropiedad"
                              depends="tiposDependencias">
        <arg0 key="${}"/>
        </field>
    </form>
</formset>
```

Cada formulario tendrá su sección **<form name=””> … </form>** por cada formulario que se desee validar.

También tendrá una sección y un **<field> … </field>** por cada campo del formulario que se desee validar. Teniendo en cuenta que los ***tiposDependecias*** pueden ser varios y deben ir separados por comas.

Los tipos de dependencias son las siguientes:

required	minlength	maxlength
invalid	byte	short
integer	long	float
double	date	range
creditcard	email	mask

Para más información de cómo configurar este fichero fijarse en los ejemplos dados en el validation.xml de la aplicación base o en el documento mencionado anteriormente ***Struts in Action***.

En resumen, este fichero es en el que se configuraran todos los campos de todos los formularios donde se quiera hacer validaciones y las validaciones que se deseen asociar a cada campo.

5.8.3 Actuaciones en los ActionsForms y AplicacionResources.properties

En los ActionsForms hay que indicarles que usen estas reglas de validación. Para eso es necesarios revisar:

1. Hacer que los Form que se deseen validar en la parte servidora hereden de **org.apache.struts.validator.ValidatorForm** en vez de **org.apache.struts.action.ActionForm** [2] (Para validación en servidor)

2. Comprobar que la clase compila revisando las entradas **import**

```
import org.apache.struts.validator.ValidatorForm;
```

3. Comprobar que al menos existe el siguiente código en el método **validate(ActionMapping mapping, HttpServletRequest request)** del Form [2] (Para validación en servidor):

```
ActionErrors errores=super.validate(mapping,request);
if (errores==null)
    errores = new ActionErrors();
```

```
    //INICIO: Aquí se introducir el código con los controles
    //añadidos que se deseen realizar
    //FIN.
    return errores;
```

4. Comprobar que en el fichero **AplicationResources.properties** los mensajes asociados a las validaciones introducidos en el **validation.xml** o si se quieren usar los de por defecto introducir la lista siguiente:

```
# Struts Validator Basic Error Messages

errors.required={0} is required.
errors.minlength={0} cannot be less than {1} characters.
errors.maxlength={0} cannot be greater than {1} characters.
errors.invalid={0} is invalid.
errors.byte={0} must be a byte.
errors.short={0} must be a short.
errors.integer={0} must be an integer.
errors.long={0} must be a long.
errors.float={0} must be a float.
errors.double={0} must be a double.
errors.date={0} is not a date.
errors.range={0} is not in the range {1} through {2}.
errors.creditcard={0} is not a valid credit card number.
errors.email={0} is not a valid e-mail address.
```

5.8.4 Actuaciones en la JSP del formulario y validaciones en cliente

Para la **Validación en cliente**[1] existen varias posibilidades dependiendo de cómo se construya la página:

Si existe tag **<html:submit>** se debe:

- Añadir en el tag **<html:form>** el atributo **onsubmit= "validate*NombreFormulario*(this)"**
- Añadir justo después del tag **</html:form>** el tag:

```
<html:javascript formName="NombreFormulario" />
```

Si **no existe tag <html:submit>** se debe:

- Revisar todos los métodos javaScript donde exista la sentencia document.forms[0].submit(); o alguna otra equivalente y añadir a la lógica que tuviese si se cree conveniente hacer la validación en cliente en vez de la sentencia anterior lo siguiente:

```
if(validateNombreFormulario(document.forms[0])){
  document.forms[0].submit();
 }
```

- Además como en el caso anterior justo después del tag **</html:form>** el tag siguiente:

```
<html:javascript formName="NombreFormulario" />
```

Y Recuerda que:

- Debes comprobar en el fichero **/WEB-INF/*nombreAplicación*-config.xml** que el formulario con el que estás trabajando se valide con Struts e indicar la página JSP en la cual retornar en caso de no superar la validación. Esto es, tener los valores *validate=true* e *input="pagina.jsp".* Ejemplo :

```
<action path="/DEMOSolVacacionesAl"
                type="es.jcyl.uic.demo.actions.DEMOSolVacaciones"
                input="DEMOSolVacaciones.jsp"
                name="DEMOFormularioForm"
                scope="request"
                validate="true">
                <forward name="exito" path="/DEMOSolVacaciones.do"/>
</action>
```

- Comprobar que en la página JSP del formulario están las tags donde presentar los mensajes de error definidos en la validación, como por ejemplo:

```
<!-- Propiedad "DNombre" -->
    <tr><td class="textoTituloGris" align="left">
        DNombre
    </td><td align="left">
        <html:text property="DNombre"/> <html:errors property="DNombre" />
    </td></tr>
```

5.9 Gestión de transacciones en aplicación Base.

El objetivo principal de ésta nueva propuesta es mejorar la Gestión de transacciones entre objetos OAD's. Con ella, se pretende no tener que ir pasando la conexión y el usuario por toda las clase de lógica de negocio y OAD's que involucren a una transacción.

La filosofía propuesta para conseguir esto es **asociar tanto la conexión como el usuario al thread que se está utilizando que en una transacción siempre es el mismo**.

En su implementación se han modificado clases generadas en la aplicación base como nuevas versiones y nuevas clases de las librerías internas al framework: (**jcylfw-1.1.jar**, **jcylutiles-1-1.jar**, **jcyldb-1.0.jar, o posteriores**).

El detalle de esas actuaciones es:

- Relacionado con la asociación del usuario al thread.
    - **es.jcyl.framework.JCYLUsuario** (Encargada de asociar al thread el usuario)
    - **es.jcyl.framework.JCYLRequestProcessor** (Encargada de asociar el usuario que se encuentra en la sesión al thread actual)
    - **NombreAplicacionLoginAction** (Encarga de asociar el usuario que se encuentra introduce en la sesión por primera vez al thread actual)
    - **NombreAplicacionConexionDB** (Encarga de realizar la auditoria si fuese necesario para todo tipo de conexión)

- Relacionado con la asociación de la conexión al thread.
    - **NombreAplicacionPruebaAccesoBDOAD** (se ha convertido a patrón singelton y se controlan las transacciones basándose en el clase JCYLGestionTransacciones)
    - **NombreAplicacionPruebaAccesoBDLN** (no se ha convertido a patron singelton pero es recomendable hacerlo por eficiencia y se controlan las transacciones basándose en el clase JCYLGestionTransacciones)
    - **NombreAplicacionConexionDB** (se controlan las conexiones basándose en el clase JCYLGestionTransacciones)

Para ello se han creado las siguientes clases del framework

- Relacionado con la asociación del usuario al thread.
    - **es.jcyl.framework.utiles.JCYLGestionUsuario** (es la que se utilizara para obtener el usuario asociado al thread donde nos encontramos)

- Relacionado con la asociación de la conexión al thread.
    - **es.jcyl.framework.db.JCYLDataBase** (es la que se utilizara para basada en el patrón Proxy obtener la conexión y la asocia al thread en el que se encuentre a través de la clase JCYLGestionTransacciones)

- **es.jcyl.framework.db.JCYLGestionTransacciones** (es la que se utilizara para controlar las transacciones asociadas al thread en el que nos encontramos)

Además se ha modificado el generador de OAD's para que incluya las modificaciones oportunas, para implantar lo explicado como se indica en el apartado siguiente.

La filosofía de de esta solución se basa en asociar objetos al thread que se esté ejecutando (usuario para auditorias, conexión para transacciones) y con ello no hace falta pasarlos a las clases de lógica de negocio ni a los OAD's por tanto si dentro de una misma clase de lógica de negocio se quiere realizar una transacción al principio del método de esta lógica de negocio se abre la transacción (**open**) y al final del método se realiza el **commit** si ha ido bien la transacción o **rollback** si ha fallado la transacción y se cierra la transacción(**close**), mientras que en los OAD's se obtiene la conexión que ha sido abierta en la lógica de negocio y se utiliza.

Si la transacción es tan compleja que implica a varias lógicas de negocio existen dos posibles soluciones basándonos en esta filosofía:

1. Realizar una clase de lógica de negocio "genérica" que se encarga de abrir y cerrar transacción y en medio de estas llamadas invocar a las distintas lógicas de negocio involucradas.
2. Que sea el action el que se encarga de abrir y cerrar transacción y en medio de estas llamadas invocar a las distintas lógicas de negocio involucradas en la transacción.

5.10 Gestión dinámica de configuración de la aplicación

5.10.1 Introducción

Todas las aplicaciones tienen la necesidad de almacenar parámetros operativos de forma externa al código, con el objetivo de facilitar su despliegue en entornos de producción y permitir la adecuada integración con elementos externos (servidores de correo, de ficheros, otras aplicaciones, etc.)

El framework de desarrollo para aplicaciones J2EE de la Junta de Castilla y León gestiona la configuración de una forma centralizada mediante el componente GestorConfiguración descrito en este documento. En esta versión se ofrecen **ya configurado el componente** de forma que tenga un comportamiento y ubicación equivalente a las que tenía en versiones anteriores.

Las principales características que relacionan este componente son:

- Permite a las aplicaciones almacenar su configuración tanto en ficheros de propiedades (.properties) como en ficheros en xml.
- Permite gestionar de forma transparente múltiples ficheros de configuración. Los parámetros pueden estar en uno o varios ficheros, que pueden agruparse por contextos o tratarse de forma centralizada.
- La aplicación tiene acceso a los cambios realizados en los ficheros de configuración sin necesidad de reiniciar la ejecución (recarga en caliente).
- La memoria ocupada por las colecciones de parámetros de una aplicación se optimiza siempre al máximo, gracias a un mecanismo dinámico de carga y desalojo de contextos de configuración.

El acceso a los ficheros de configuración y la gestión de los mismos se realiza gracias a las librerías proporcionadas por el proyecto **commons-configuration** del proyecto Jakarta.

http://jakarta.apache.org/commons/configuration/

5.10.2 Guía Rápida de uso

En este apartado se presenta un ejemplo sencillo de utilización del componente de configuración.

La referencia detallada puede consultarse en el manual específico del componente.

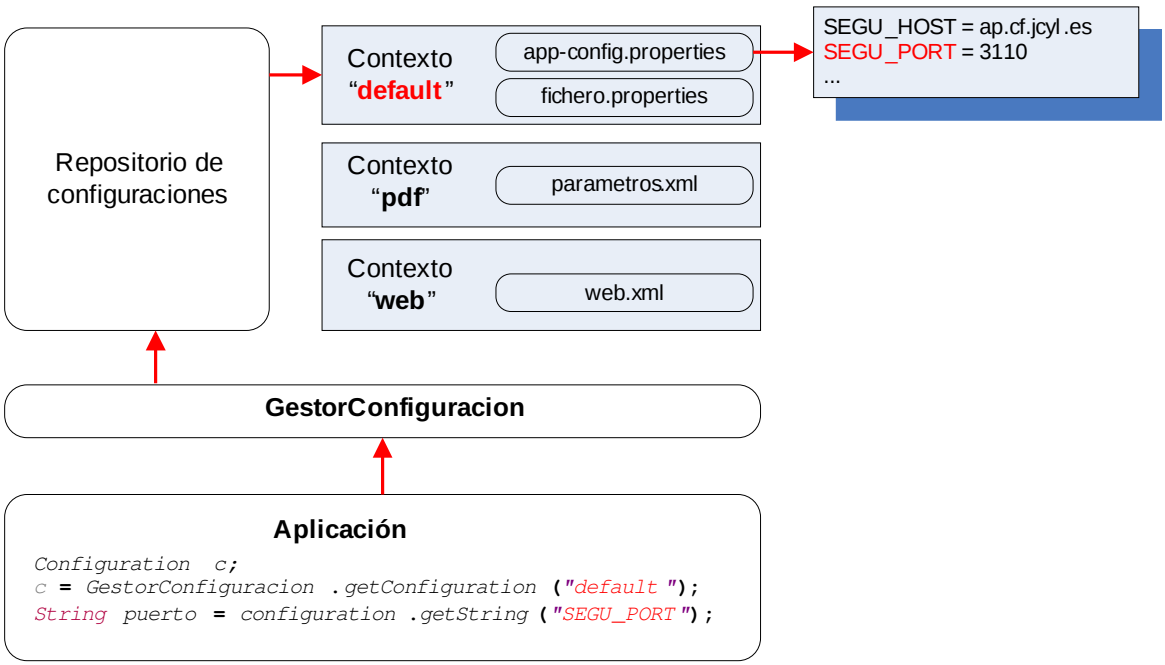
5.10.2.1 Elementos para configuración de la aplicación

Para que una aplicación haga uso del sistema de configuración, debe incorporar un fichero denominado ***repositorio de configuraciones***, que contiene la referencia a todos los ficheros de configuración de la aplicación.

Para desacoplar la aplicación de la forma de ubicar los ficheros, este repositorio define el concepto de **contexto** como una agrupación de ficheros de configuración. Cuando la aplicación desee leer un parámetro de configuración lo hará especificando el contexto en que se encuentra.

La configuración de la aplicación como tal se encuentra repartida en uno o varios **ficheros**, siempre asociados a un contexto. Este componente soporta los tradicionales ficheros .properties y ficheros XML

Esquemáticamente puede verse el conjunto en este diagrama:



Un ejemplo sencillo del fichero que define el repositorio se representa a continuación:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configurations version="1.0" xmlns:xsi='http://www.w3.org/2001/XMLSchema-
instance' xsi:noNamespaceSchemaLocation='/xsd/ConfigurationRepository.xsd'>

   <!-- Configuraciones globales -->
   <globals>
       <!-- Ruta base para todos los ficheros de configuración -->
       <initialPath value="${web-inf.directory}/config/"/>

       <!-- Propiedades globales -->
       <properties>
           <property name="minimumCheckDelay" value="10000"/>
           <property name="contextTimeOut" value="10000"/>
       </properties>
   </globals>

   <!-- Lista de contextos de la aplicación -->
   <contexts>
       <!-- Configuración de la aplicación -->
       <context id="default" name="Configuración de la aplicación">

           <configurationFile
                    id="application"
                    name="Parámetros generales de la aplicación"
                    location="/pruebas/config/prueba.properties"/>

        </context>

       <!-- Configuración del módulo de PDF -->
       <context id="pdf" name="Componente de generación de documentos PDF">

           <configurationFile
                   id="application"
                   name="Parámetros del componente"
                   location="./pdf.xml"/>
       </context>

   </contexts>
</configurations>
```

Este repositorio contiene los siguientes elementos:

- Una lista de propiedades globales que permiten la parametrización del propio gestor de configuraciones.

```
<!-- Configuraciones globales -->
<globals>
   <!-- Ruta base para todos los ficheros de configuración -->
   <initialPath value="${web-inf.directory}/config/"/>

   <!-- Propiedades globales -->
   <properties>
       <property name="minimumCheckDelay" value="10000"/>
       <property name="contextTimeOut" value="10000"/>
   </properties>
</globals>
```

- Un contexto **default** que para la configuración básica de la arquitectura de la aplicación. Contiene un único fichero de configuración basado en un fichero de propiedades estándar, denominado **prueba.properties**.

```
<!-- Configuración de la aplicación -->
<context id="default" name="Configuración de la aplicación">

    <configurationFile
          id="application"
          name="Parámetros generales de la aplicación"
          location="/pruebas/config/prueba.properties"/>
</context>
```

- Un contexto **pdf** para la configuración de un componente para generación de ficheros en formato pdf. Contiene un único fichero de configuración basado en xml denominado **pdf.xml**.

```
<!-- Configuración del módulo de PDF -->
<context id="pdf" name="Componente de generación de documentos PDF">

    <configurationFile
          id="application"
          name="Parámetros del componente"
          location="./pdf.xml"/>

</context>
```

5.10.2.2 Acceso a la configuración desde la aplicación

Para permitir que la aplicación localice el fichero con el repositorio de configuraciones debe ser inicializada indicando la ruta del fichero.

Este proceso debe realizarse una única vez durante la ejecución de la aplicación. Obsérvese que el uso del componente GestorConfiguración se realiza mediante métodos estáticos, por lo que no es necesario almacenar ninguna referencia al mismo.

5.10.2.3 Inicialización para aplicaciones web

En el caso de aplicaciones web, basta con añadir una nueva entrada en el fichero de despliegue web.xml de su aplicación, indicando la ruta donde se encuentra:

```
  <env-entry>
     <env-entry-name>configurationRepository</env-entry-name>
     <env-entry-value>applications/demo/demo/WEB-
INF/config/ConfigurationRepository.xml</env-entry-value>
     <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
```

Desde la aplicación debe obtener este valor y emplearlo para inicializar el componente GestorConfiguracion.

```
// Obtiene la ruta al configuration repository
String ruta = getEnvEntry("configurationRepository");

// Inicializa GestorConfiguracion a partir de la ruta donde se encuentra
GestorConfiguracion.initialize( ruta );
```