JUNTA DE ANDALUCIA

CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA

Desarrollo de Iniciativas de Admón. Electrónica en la JA

Guía de Instalación del Entorno de Desarrollo de la Plataforma de Tramitación Electrónica

Proyecto: Desarrollo de Iniciativas de Admón. Electrónica en la JA	**Fecha : 25 Junio 2008**
Número de páginas: 54	**Autor: Jorge Agudo Praena, Iván Jesús Correo Negrín, José Luis Salas Espina,Dimas Tejero Pozo.**
Aprobada por parte de CJAP por: **Fecha:**	

Versión: <v01r03>

Fecha: **: 25 Junio 2008**

HOJA DE CONTROL

Título	Desarrollo de Iniciativas de Admón. Electrónica en la JA		
Entregable	Guía de Instalación del Entorno de Desarrollo de la Plataforma de Tramitación Electrónica		
Nombre del Fichero	Instalacion_Entorno_Desarrollo_v01r02.doc		
Autor	Jorge Agudo Praena, Iván Jesús Correo Negrín, José Luis Salas Espina, José Manuel Marín de la Rosa, Dimas Tejero Pozo		
Versión/Edición	<v01r03>	**Fecha Versión**	**25 Junio 2008**
Aprobado por		**Fecha Aprobación**	
		Nº Total Páginas	54

REGISTRO DE CAMBIOS

Versión	Causa del Cambio	Responsable del Cambio	Área	Fecha del Cambio
01	Versión inicial	Jorge Agudo Praena, Iván Jesús Correa Negrín, José Luis Salas Espina, Manuel Pérez Coca		11 Junio 2007
02	Guía de desarrollo de componentes funcionales	Jorge Agudo Praena		22 Noviembre 2007
03	Manual del Desarrollador	Dimas Tejero Pozo		25-Junio-2008

CONTROL DE DISTRIBUCIÓN

Nombre y Apellidos	Área	Nº Copias

ÍNDICE

1 Introducción 5

2 Instalación del entorno de desarrollo 6

2.1 Instalación de *Maven* 6

2.2 Instalación de librerías adicionales 8

2.3 Compilar el proyecto 9

2.4 Configuración del *IDE* 10

2.4.1 Configuración del proyecto para *Eclipse* 10

3 Guía de desarrollo de Componentes Funcionales 12

3.1 Introducción 12

3.2 Tipos de componentes funcionales 13

3.3 Especificaciones para la construcción de un nuevo módulo funcional 15

3.3.1 Estructura del archivo de descripción despliegue.xml 17

3.3.2 Contenido del directorio conf 23

3.3.3 Contenido del directorio lib 24

3.3.4 Contenido del directorio webapp 24

3.3.5 Información accesible a través de la sesión 27

4 Creación de un Proyecto Vertical y Manual del Desarrollador 29

4.1 Introducción: 29

4.2 Tecnología utilizada en Plataforma de Tramitación. 31

4.3 Servicios 33

4.4 Alta Expediente 35

4.5 Tareas 41

4.5.1 Tareas tipo Incorporar documento 41

4.5.2 Tareas tipo Generar documento 41

4.5.3 Tareas Web o de adquisición datos y tipo Otros 41

4.6 Condiciones 45

4.7 Acciones 46

4.8 Tareas tipo Web mediante XSD 47

4.9 Desarrollo de Utilidades en Plataforma de Tramitación. 51

4.9.1 Introducción 51

4.9.2 Especificaciones para la construcción de una nueva utilidad 51

5 Notas sobre la Guía de Instalación del Entorno de Desarrollo de la Plataforma de Tramitación Electrónica 54

1 Introducción

El objetivo del presente documento es indicar los pasos a seguir para instalar y configurar el entorno de desarrollo para la construcción de la Plataforma de Tramitación Electrónica desarrollada para la consejería de Justicia y Administración Electrónica.

También se incluye una guía de desarrollo de componentes funcionales específicos y dependientes de las familias de tramitación, donde se describen los distintos tipos de módulos, y se especifican los pasos a seguir para la construcción de nuevos módulos funcionales.

2 Instalación del entorno de desarrollo

2.1 Instalación de *Maven*

Para el control de dependencias del código fuente así como la construcción del archivo ejecutable de la aplicación se hará uso de la herramienta *Maven*, que puede ser descargada de forma gratuita desde el sitio web del proyecto (http://maven.apache.org). La versión utilizada de dicha herramienta será la 2.0.6, pudiendo utilizarse cualquier versión posterior a la 2.0, ésta incluida. Existen versiones de dicha herramienta tanto para plataforma Windows como para Linux y la instalación en ambos casos consiste en:

- Descomprimir el fichero descargado en cualquier directorio del sistema donde se desee instalar la herramienta.
- Añadir la ruta completa el subdirectorio bin que incluye la herramienta al *path* de nuestro sistema.
- En caso de trabajar en una red con acceso a través de un *proxy*, será necesario configurar el uso del mismo por parte de la herramienta ya que *Maven* tratará siempre de descargar las librerías de las que depende el proyecto que se quiere construir de los servidores *Maven* distribuidos por todo el mundo. Para ello, es necesario editar el fichero de configuración global settings.xml que se encuentra en el directorio conf y configurar el elemento *proxy* con los datos necesarios para conectarse a los servidores *Maven* disponibles y poder realizar las descargas necesarias a través del *proxy* que se quiera utilizar. Una posible configuración sería:

```
[...]
  <proxy>
    <active>true</active>
    <protocol>[protocolo de acceso al proxy, p. ej. “http”]</protocol>
    <host>[dirección del proxy]</host>
    <port>[puerto de conexión del proxy]</port>
    <username>[usuario de conexión al proxy(opcional)]</username>
    <password>[contraseña del usuario de conexión (opcional)]</password>
    <nonProxyHosts>[direcciones que no pasan por el Proxy, p. ej. “localhost”]</nonProxyHosts>
  </proxy>
[...]
```

- Ejecutar el comando mvn –version para comprobar que la herramienta ha sido correctamente instalada. En caso de que así se, se mostrará en pantalla un mensaje como el siguiente mensaje o similar:

```
Maven version: 2.0.6
```

Puede encontrarse documentación detallada sobre la configuración, uso y ajuste de *Maven* en la página del proyecto (http://maven.apache.org).

2.2 Instalación de librerías adicionales

Maven intentará descargar de manera automática aquellas librerías con las que el proyecto que se quiere procesar tiene declarada alguna dependencia, buscando en aquellos repositorios que se han configurado en el descriptor del proyecto. Sin embargo, determinadas librerías pueden no encontrarse disponibles online por lo que deberán instalarse de forma manual. Para ello, hay que ejecutar todos los ficheros de instalación install_*.bat (MS Windows) o install_*.sh (Linux) situados en la subcarpeta del proyecto instalacion/entorno/maven/scripts/[windows | linux], a fin de que dichas librerías que no pueden ser descargadas se instalen correctamente en el repositorio local que *Maven* crea en nuestro equipo local. En caso de que el proceso se complete correctamente, aparecerá en pantalla un mensaje como el siguiente:

```
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'install'.
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [install:install-file] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] [install:install-file]
[INFO] Installing [ruta al fichero jar instalado] to [ruta al directorio del repositorio maven local donde se instala el fichero jar]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Thu Nov 22 11:41:44 CET 2007
[INFO] Final Memory: 3M/5M
[INFO] ------------------------------------------------------------------------
```

2.3 Compilar el proyecto

Ejecutar el comando mvn compile en el directorio raíz del proyecto (se podrá comprobar que en dicho directorio existe un archivo llamado pom.xml). De esta forma, se comprobará que todas las dependencias han podido ser resueltas por *Maven* y todo el código fuente del proyecto puede ser compilado sin problemas, en cuyo caso se obtendrá el siguiente mensaje.:

```
[INFO] ————————————————————
[INFO] BUILD SUCCESSFUL
[INFO] ————————————————————
[INFO] Total time: 7 seconds
[INFO] Finished at: Mon Jun 11 18:04:55 CEST 2007
[INFO] Final Memory: 4M/7M
[INFO] ————————————————————
```

Nota: *la primera vez que se ejecute este comando,* Maven *descargará aquellas librerías que estén declaradas como dependencias en el descriptor del proyecto y las instalará en un repositorio local, por lo que es posible que el proceso tarde un tiempo considerable en ejecutarse completamente.*

2.4 Configuración del *IDE*

Los descriptores de proyectos utilizados por *Maven* (ficheros pom.xml) son independientes del entorno de desarrollo integrado (*IDE*) que se utilice por lo que, a partir de los mismos, hay que generar los ficheros de configuración del proyecto que utilice el *IDE* que se quiera utilizar durante el desarrollo.

2.4.1 Configuración del proyecto para *Eclipse*

Los pasos para poder trabajar en el proyecto con el *IDE Eclipse* son los siguientes:

1. Se debe mover el proyecto (con el fichero pom.xml en su raíz) al directorio correspondiente al *workspace* de *Eclipse* sobre el que se vaya a trabajar.
2. Para que *Eclipse* pueda resolver correctamente la ruta al repositorio local donde se encuentran resueltas las dependencias que hay declaradas en el descriptor del proyecto (fichero pom.xml), se debe definir en el entorno la variable de configuración M2_REPO con la ruta donde se encuentra el repositorio local para que *Eclipse* pueda encontrar todas las librerías referenciadas. Esto puede hacerse manualmente en el propio *IDE* o bien dejar que sea *Maven* quien lo haga ejecutando el siguiente comando:

```
mvn -Declipse.workspace=<ruta-al-workspace-de-eclipse> eclipse:add-maven-repo
```

Nota: *la ruta al* workspace *puede ser relativa.*

3. Ejecutar desde el directorio raíz del proyecto, donde se encuentra el fichero pom.xml, el siguiente comando:

```
mvn -Dwtpversion=1.5 eclipse:eclipse
```

Esto creará los ficheros .project y .classpath así como el directorio .settings que contienen la información necesaria para que *Eclipse* reconozca el proyecto como un proyecto propio del *IDE* con naturaleza de proyecto web.

Nota: *la herramienta de configuración del proyecto es* Maven *y no el* IDE *específico utilizado durante el desarrollo, por lo que ninguno de los ficheros generados para la utilización de un determinado* IDE *debería almacenarse en el repositorio donde se encuentre el código fuente del proyecto.*

4. Crear un nuevo proyecto del tipo *"Java Project"*, introducir el mismo nombre que tenía el proyecto *Maven* y dejar que el propio asistente de creación de proyectos importe el ya existente con la configuración recién creada.

3 Guía de desarrollo de Componentes Funcionales

3.1 Introducción

El objetivo de esta sección es detallar las especificaciones y pautas a seguir para la construcción de nuevos componentes funcionales específicos que se quieran instalar sobre la Plataforma de Tramitación. Para ello, se establecerá una clasificación de los tipos de módulos que se pueden usar en la plataforma y se dará una visión global de la filosofía y estructura que se persigue con este sistema de ampliación de funcionalidades de la plataforma en base a *plugins* o módulos.

Estas pautas son de obligado cumplimiento para lograr la construcción de un módulo compatible con la plataforma de tramitación y pueden ser completadas con otras recomendaciones relacionadas con normas de codificación y diseño de patrones.

Este documento es susceptible de ser evolucionado y perfeccionado a lo largo de la vida del proyecto motivado por nuevas exigencias y/o experiencias en las diferentes implantaciones que se realicen de la Plataforma de Tramitación.

3.2 Tipos de componentes funcionales

Por su naturaleza funcional, existen dos tipos de componentes funcionales o módulos en la Plataforma de Tramitación, aunque ambos son desarrollados de la misma forma:

- Componentes funcionales de tramitación, globales a cualquier familia de procedimientos.
- Componentes funcionales verticales, específicos de un procedimiento o familia de procedimientos.

En la *ilustración 1* tenemos un ejemplo de escritorio, en el podemos ver los distintos módulos genéricos que existen, como son: información usuario, datos del expediente, datos de la fase, transiciones posibles...etc.

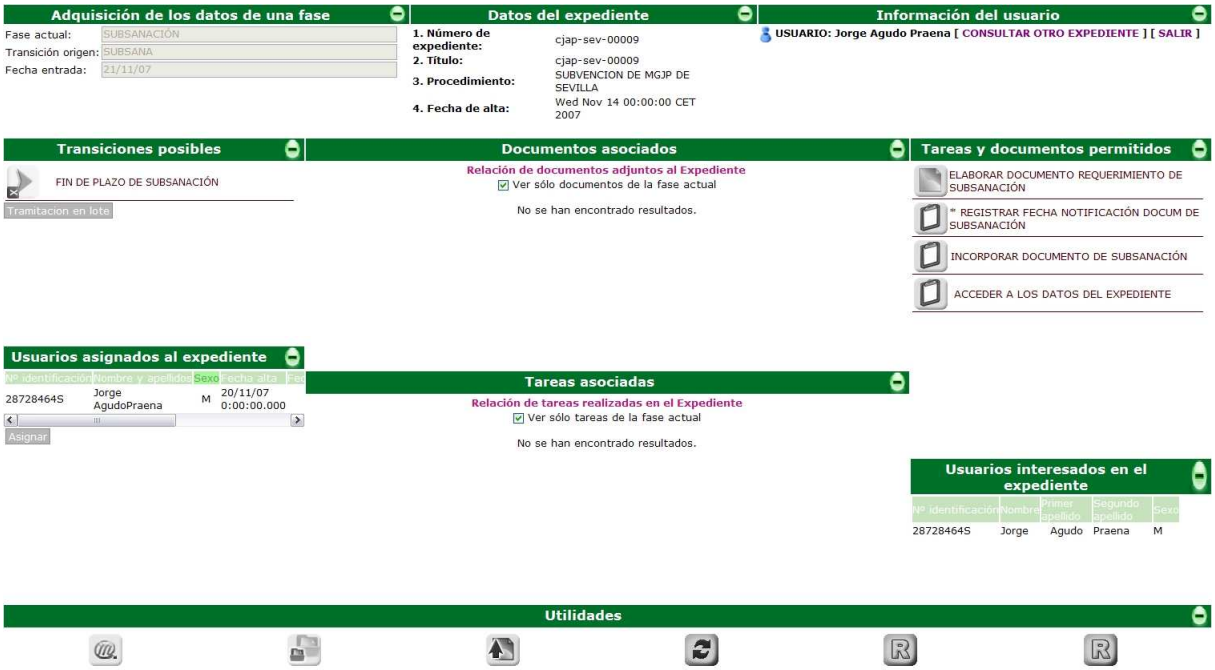


Ilustración 1: Interfaz de la Plataforma de Tramitación

Por su forma de presentación, existen cuatro tipos de *plugins* para la Plataforma de Tramitación:

- *Portlets* presentados en una zona determinada del escritorio de tramitación.
- Utilidades presentadas en forma de iconos dentro del *portlet* genérico para tal fin (titulado *Utilidades*).

- Externos al escritorio de tramitación y accesibles a través de una determinada dirección URL definida en el propio módulo.
- Sin representación alguna como, por ejemplo, módulos empleados para cargar librerías de utilidades necesarias.

3.3 Especificaciones para la construcción de un nuevo módulo funcional

Para que la Plataforma de Tramitación pueda desplegar correctamente un determinado componente, éste debe seguir un formato y estructura predeterminada. La cual se empaqueta en un archivo comprimido en formato ZIP. Dicha estructura es la siguiente:

- Un archivo de descripción del módulo en la raíz del directorio que lo contenga llamado despliegue.xml. Su inclusión es obligatoria para todo módulo que e se quiera instalar.
- Un directorio con archivos de configuración llamado conf. Dicho directorio deberá contener los archivos de configuración tanto de *Struts* como *Struts2*, dependiendo de la tecnología con la que haya sido desarrollado el módulo. Su inclusión es obligatoria aunque su contenido puede ser nulo.
- Un directorio con librerías necesarias para el correcto funcionamiento del módulo llamado lib. Su inclusión es obligatoria aunque su contenido puede ser nulo.
- Un directorio con archivos *JSP*, *JavaScript*, imágenes, hojas de estilo *CSS*, etc. necesarios para la construcción del módulo. Su inclusión es obligatoria aunque su contenido puede ser nulo.

A continuación se puede observar la estructura de un componente funcional simple que calcula la fase actual del expediente que se está tramitando:

Nombre	Tipo	Tamañ...	Tien...	Tam...	R...	Fecha
conf	Carpeta de a...	0 KB		0 KB	0%	
lib	Carpeta de a...	0 KB		0 KB	0%	
webapp	Carpeta de a...	0 KB		0 KB	0%	
despliegue.xml	Documento XML	1 KB	No	1 KB	53%	05/09/2007 14:19

Ilustración 2: Contenido de un módulo de ejemplo

Que, al descomprimirse, quedaría así:

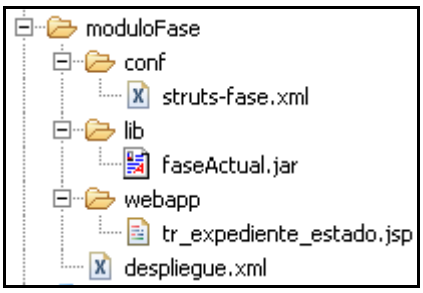


Ilustración 4: Jerarquía de carpetas del módulo de ejemplo

Una vez se haya instalado el módulo mediante la herramienta de configuración disponible en el área de administración de la Plataforma de Tramitación, el archivo zip se descomprime dejando los archivos que lo componen en diferentes ubicaciones que deben tenerse en cuenta para manejar todas las rutas que escribamos en nuestro código (jsp, javascript, xml..) según las siguientes reglas:

- Los archivos y directorios ubicados dentro del directorio webapp se desplegarán en un nuevo directorio con el mismo nombre que el módulo instalado. Dicho directorio se encontrará dentro del directorio modulos de la aplicación web correspondiente a la Plataforma de Tramitación, donde existirá una subcarpeta por cada módulo instalado en la aplicación.
- El contenido del directorio lib del módulo instalado se copiará dentro del directorio WEB-INF/lib de la aplicación web correspondiente a la Plataforma de Tramitación. Para evitar que al modificar este directorio con nuevas librerías se recargue la aplicación es conveniente tener configurado el servidor de aplicaciones con la opción de recarga de contexto desactivada.
- El contenido del directorio conf se copiará en el directorio WEB-INF/modulos/[nombreModuloInstalado].

De forma adicional, el proceso de instalación también modificará el archivo de configuración de la aplicación (archivo WEB-INF/web.xml) al instalar módulos desarrollados con tecnología *Struts*, introduciendo una entrada similar a la siguiente (suponiendo que se haya instalado un módulo de nombre *portafirmas* con un fichero de configuración de reglas de *Struts* llamado struts-config-portafirmas.xml dentro de su directorio conf):

```
[...]

<init-param>
  <param-name>config/modulos/portafirmas</param-name>
  <param-value>/WEB-INF/modulos/portafirmas/struts-config-portafirmas.xml</param-value>
</init-param>
```

```
[...]
```

Del mismo modo, para módulos desarrollados con tecnología *Struts2* se modificará el archivo de configuración de reglas de *Struts2* (/WEB-INF/classes/struts.xml) al instalar módulos desarrollados con dicha tecnología, introduciendo una entrada similar a la siguiente (suponiendo que se haya instalado un módulo de nombre *portafirmas* con un fichero de configuración de reglas de *Struts2* llamado struts-portafirmas.xml dentro de su directorio conf):

```
[...]

<include file="../modulos/portafirmas/struts-portafirmas.xml"/>

[...]
```

3.3.1 Estructura del archivo de descripción despliegue.xml

El archivo despliegue.xml es el que describe un módulo que quiera ser instalado en la Plataforma de Tramitación y donde se detallan los aspectos informativos y descriptivos del mismo. El formato que deben seguir es el siguiente:

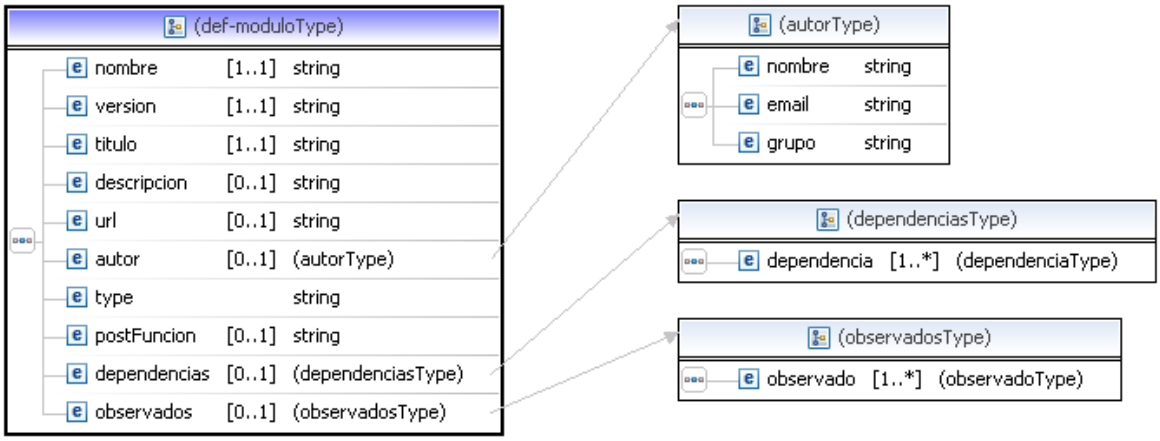


Ilustración 5: Formato de diseño del archivo despliegue.xml

Los campos que estructuran este documento en formato xml son:

- **nombre**: nombre del módulo (deberá ser único en la Plataforma de Tramitación para que el módulo se pueda instalar correctamente).
- **version**: número de la versión del módulo.
- **titulo**: título mostrado en el escritorio de tramitación en caso de presentarse el módulo como un *portlet* en el mismo.
- **descripción**: texto descriptivo del módulo.
- **url**: dirección correspondiente a la página inicial del módulo.
- **autor**: datos del autor del módulo.
- **type**: tecnología con la que se ha desarrollado el módulo (puede tomar los valores STRUTS-1, STRUTS-2 o NONE).
- **tipoInstalacion**: forma de visualización del módulo (puede tomar los valores PORTLET, EXTERNO, UTILIDADES y NONE).
- **postFuncion**: permite ejecutar una función javascript tras la recarga del módulo.
- **dependencias**: conjunto de dependencias del módulo con otros módulos ya instalados en la Plataforma de Tramitación. Para que un módulo pueda ser instalado correctamente se deberán encontrar instalados previamente todos aquellos módulos de los que dependa.
- **observados**: conjunto de módulos por los que tiene que ser informado el módulo cuando se produzcan cambios en ellos.

El archivo despliegue.xml deberá cumplir el formato descrito en el fichero validación.xsd que se presenta a continuación:

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="def-modulo">

    <xs:complexType>

      <xs:sequence>

        <xs:element name="nombre" type="xs:string" minOccurs="1"/>

        <xs:element name="version" type="xs:string" minOccurs="1"/>

        <xs:element name="titulo" type="xs:string" minOccurs="1"/>
```

```
<xs:element name="descripcion" type="xs:string" minOccurs="0" maxOccurs="1"/>

<xs:element name="url" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xs:element name="icono-on" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xs:element name="icono-off" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xs:element name="autor" minOccurs="0" maxOccurs="1" >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="email" type="xs:string" />
      <xs:element name="grupo" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="type">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="STRUTS-1"/>
      <xs:enumeration value="STRUTS-2"/>
      <xs:enumeration value="NONE"/>
    </xs:restriction>
     </xs:simpleType>
</xs:element>
<xs:element name="tipoInstalacion" minOccurs="0" maxOccurs="1" default="PORTLET">
```

```
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration  value="PORTLET"/>
          <xs:enumeration value="EXTERNO"/>
          <xs:enumeration value="UTILIDADES"/>
          <xs:enumeration value="NONE"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="postFuncion" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="dependencias" maxOccurs="1" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="dependencia" minOccurs="1" maxOccurs="unbounded" >
            <xs:complexType>
              <xs:sequence>
                  <xs:element name="modulo" minOccurs="1" maxOccurs="1">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="nombre" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
```

```
                    <xs:element name="version" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="observados" maxOccurs="1" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="observado" minOccurs="1" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence maxOccurs="1" minOccurs="1">
                    <xs:element name="nombre" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```
</xs:schema>
```

A continuación se muestra un ejemplo de descriptor para un posible módulo llamado modulofase:

```
<?xml version="1.0" encoding="UTF-8" ?>
<def-modulo>
  <nombre>fase</nombre>
  <version>1.0</version>
  <titulo>Módulo de fase actual</titulo>
  <descripcion>Indica la fase actual del expediente</descripcion>
  <autor>
    <nombre>Everis</nombre>
    <email>icorrean@everis.com</email>
    <grupo>Everis</grupo>
  </autor>
  <type>STRUTS-2</type>
  <postFuncion></postFuncion>
  <dependencias>
    <dependencia>
      <modulo>
        <nombre>comun</nombre>
      </modulo>
      <version>1.0</version>
    </dependencia>
  </dependencias>
</def-modulo>
```

3.3.2 Contenido del directorio conf

Este directorio debe contener los archivos de configuración del módulo según la tecnología utilizada para su desarrollo (archivos de configuración de *Struts* o *Struts2*, según el caso). Para un posible módulo llamado modulofase, se podría incluir el siguiente fichero si fuera desarrollado con tecnología *Struts2*:

```
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="fase" namespace="/fase" extends="struts-default">
    <action name="elegirEstadoExpediente" method="elegirEstadoExpediente"
      class="com.gtel.agenda.bean.FaseActual">
      <result name="success">/modulos/moduloFase/tr_expediente_estado.jsp</result>
      <result name="error">/agenda/comun/error.jsp</result>
      <interceptor-ref name="basicStack" />
    </action>
    <action name="listarEstadosExpediente" method="listarEstadosExpediente"
      class="com.gtel.agenda.bean.FaseActual">
      <result name="success">/modulos/moduloFase/tr_expediente_estado.jsp</result>
      <result name="error">/agenda/comun/error.jsp</result>
      <interceptor-ref name="paramsPrepareParamsStack" />
    </action>
    <action name="mostrarFunciones" method="mostrarFunciones"
      class="com.gtel.agenda.bean.FaseActual">
      <result name="success">/agenda/tr_funciones.jsp</result>
      <result name="error">/agenda/comun/error.jsp</result>
      <interceptor-ref name="paramsPrepareParamsStack" />
    </action>
  </package>
```

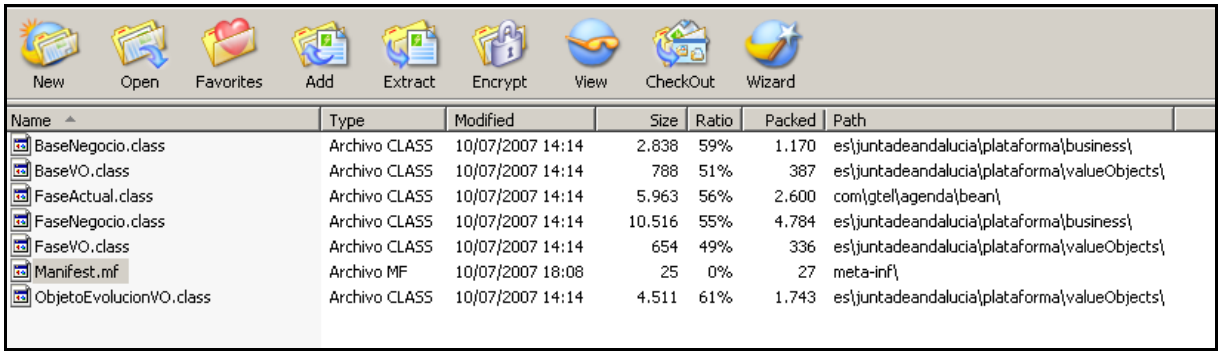
```
</struts>
```

Hay que tener en cuenta que todas las clases a las que hagan referencia los archivos de configuración (*Beans*, *ActionsBeans*, etc.) deberán ir empaquetadas en un archivo .jar situado en el directorio lib del módulo y las rutas deberán indicarse de forma relativa teniendo en cuenta la estructura de carpetas a la hora del despliegue.

3.3.3 Contenido del directorio lib

Este directorio deberá contener las librerías de las que haga uso el módulo. Se pueden adjuntar tanto aquellas generadas por su módulo como algunas librerías necesarias para su correcto funcionamiento. A su vez, en este directorio se deberá cualquier posible archivo de propiedades necesario para el correcto funcionamiento del módulo. Todas las clases utilizadas por un módulo deberán ir en un archivo .jar.

Siguiendo el ejemplo de módulo presentado anteriormente, la librería que utilizada sería faseActual.jar, y tendría el siguiente contenido:



Name	Type	Modified	Size	Ratio	Packed	Path
BaseNegocio.class	Archivo CLASS	10/07/2007 14:14	2.838	59%	1.170	es\juntadeandalucia\plataforma\business\
BaseVO.class	Archivo CLASS	10/07/2007 14:14	788	51%	387	es\juntadeandalucia\plataforma\valueObjects\
FaseActual.class	Archivo CLASS	10/07/2007 14:14	5.963	56%	2.600	com\gtel\agenda\bean\
FaseNegocio.class	Archivo CLASS	10/07/2007 14:14	10.516	55%	4.784	es\juntadeandalucia\plataforma\business\
FaseVO.class	Archivo CLASS	10/07/2007 14:14	654	49%	336	es\juntadeandalucia\plataforma\valueObjects\
Manifest.mf	Archivo MF	10/07/2007 18:08	25	0%	27	meta-inf\
ObjetoEvolucionVO.class	Archivo CLASS	10/07/2007 14:14	4.511	61%	1.743	es\juntadeandalucia\plataforma\valueObjects\

Ilustración 6: Librerías de Clases

3.3.4 Contenido del directorio webapp

En este directorio deberán incluirse todos los ficheros *JSP*, *JavaScript*, imágenes, etc... para el correcto funcionamiento del módulo. Se recomienda:

- Definir una cierta estructura de carpetas, como por ejemplo situar todos los ficheros *JavaScript* de una carpeta que cuelgue de webapp y que se denomine js, al igual que las imágenes en una carpeta llamada imagenes o img que cuelgue del directorio webapp. Los ficheros *JSP* pueden colgar directamente del directorio webapp.
- Todas las rutas que se especifiquen en un nuevo módulo deben ser relativas.
- Cuando se quiera recargar tanto el módulo que estamos desarrollando como los observados por éste, se puede hacer uso de las siguientes funciones *JavaScript*:
    - cargaPortletUnico_[nombreModuloDesarrollado]() – se encarga de actualizar el *portlet* del módulo que estamos desarrollando.
    - cargaPortlet_[nombreModuloDesarrollado]() – se encarga de actualizar los *portlets* que están observando al módulo que estamos desarrollando.

Continuando con el ejemplo anterior, se podría crear una página llamada *tr_expediente_estad.jsp*, con el siguiente contenido:

```
<%@ page contentType="text/html;charset=windows-1252"%>
<%@ taglib prefix="s" uri="/struts-tags" %>

<table border="0" width="98%">
  <s:if test="%{listaSize == 0}">
    <tr>
      <td>
        <p class="texto_estado_1">
          Estado
        </p>
      </td>
    </tr>
  </s:if>
  <s:else>
    <tr>
      <td>
        <p>
          Fase actual:
```

```
      </p>
    </td>
    <td>
      <s:if test="%{listaSize == 1}">
        <p style="color: #FF0022">
          <strong><s:property value="FASEACTUAL.descripcionFaseActual"/></strong>
        </p>
      </s:if>
      <s:elseif test="%{listaSize > 1}">
        <p class="botonListaFases" onclick="listarFases();">
          <s:property value="descripcionFaseActual"/>
        </p>
      </s:elseif>
    </td>
  </tr>
  <tr>
    <td>
      <p>
        Transición origen:
      </p>
    </td>
    <td>
      <p>
        <strong><s:property value="FASEACTUAL.transicionOrigen"/></strong>
      </p>
    </td>
  </tr>
  <tr>
    <td>
      <p>
        Fecha entrada:
      </p>
```

```
      </td>
      <td>
        <p>
          <strong><s:date name="FASEACTUAL.faseActual.FECHAENTRADA" format="dd-MM-yyyy"/></strong>
        </p>
      </td>
    </tr>
  </s:else>
</table>
```

3.3.5 Información accesible a través de la sesión

Para facilitar el desarrollo de módulos, se incluye en la variable de sesión usuario_en_sesion la información relativa al usuario que haya iniciado sesión en la Plataforma de Tramitación. Dicha información estará contenida en un objeto del tipo es.juntadeandalucia.plataforma.web.UsuarioWeb que tiene los siguientes campos que pueden ser consultados en cualquier momento:

- **usuario**: objeto de tipo IUsuario que representa al usuario en el sistema de la Plataforma de Tramitación.
- **expediente**: objeto de tipo IExpediente que representa el expediente que el usuario está tramitando en un momento dado. Sólo tendrá un valor válido cuando el usuario se encuentre en el escritorio de tramitación después de haber seleccionado un expediente para su procesamiento.
- **faseActual**: objeto de tipo IFaseActual que representa la fase actual del expediente que el usuario está tramitando en un momento dado. Sólo tendrá un valor válido cuando el usuario se encuentre en el escritorio de tramitación después de haber seleccionado un expediente para su procesamiento.
- **sistema**: objeto de tipo ISistema que representa el sistema al que se conectó el usuario al iniciar su sesión en la Plataforma de Tramitación.
- **listaPerfiles**: lista de objetos de tipo Pefil que representa todos los perfiles que tiene el usuario asignado en el motor de tramitación.
- **nombreUsuario**: cadena con el nombre de usuario en el motor de tramitación.
- **nombreLargo**: cadena con el nombre completo del usuario en el formato *Apellido1 Apellido2, Nombre*.

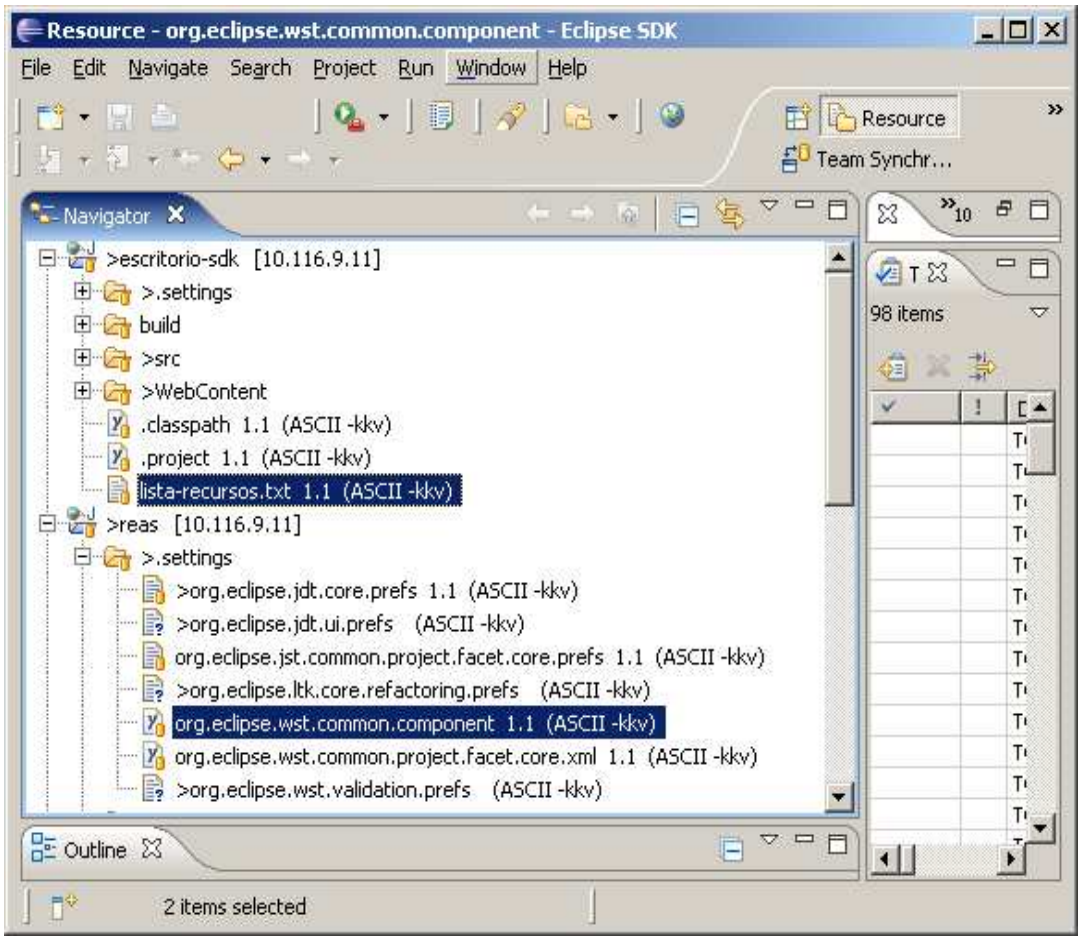
4 Creación de un Proyecto Vertical y Manual del Desarrollador

4.1 Introducción:

En el siguiente apartado se detallan todos los pasos a seguir para una vez realizada la reingeniería y modelado de un procedimiento o familia de procedimientos y cargar dicho procedimiento en Trew@, realizar la programación del alta de expediente, tareas, acciones y condiciones necesarias para el funcionamiento del procedimiento en la Plataforma de Tramitación.

Para ello es conveniente crearnos un proyecto vertical a la Plataforma de Tramitación. Nuestro proyecto vertical importará la lista de recursos de Plataforma de Tramitación (ListaRecursos.txt), así tendremos embebida la Plataforma de Tramitación y la usamos como un recurso. Conseguimos así independizar nuestro proyecto, abstrayéndolo de posibles cambios o versiones de la Plataforma de Tramitación. De esta forma varios proyectos verticales pueden usar la misma Plataforma de Tramitación.

El proyecto vertical se creará desde Eclipse, será del tipo Dynamic Web Project. Dentro del paquete : ".settings" y en el fichero : "org.eclipse.wst.common.component" importaremos la lista de recursos de la plataforma de tramitación:



El archivo: "org.eclipse.wst.common.component", con los recursos importados es: (no se muestra entero:)

```
<?xml version="1.0" encoding="UTF-8"?>
<project-modules id="moduleCoreId" project-version="1.5.0">
<wb-module deploy-name="reas">
<wb-resource deploy-path="/" source-path="/WebContent"/>
<wb-resource deploy-path="/WEB-INF/classes" source-path="/src"/>


<wb-resource deploy-path="/" source-path="../escritorio-sdk/WebContent"/>
<wb-resource deploy-path="/administracion" source-path="../escritorio-sdk/WebContent/administracion"/>
<wb-resource deploy-path="/administracion/adminSolr" source-path="../escritorio-sdk/WebContent/administracion/adminSolr"/>
<wb-resource deploy-path="/administracion/buscador" source-path="../escritorio-sdk/WebContent/administracion/buscador"/>
<wb-resource deploy-path="/administracion/componentes" source-path="../escritorio-sdk/WebContent/administracion/componentes"/>
<wb-resource deploy-path="/administracion/instalaciones" source-path="../escritorio-sdk/WebContent/administracion/instalaciones"/>
<wb-resource deploy-path="/administracion/instalaciones/errores" source-path="../escritorio-
sdk/WebContent/administracion/instalaciones/errores"/>
<wb-resource deploy-path="/administracion/modulos" source-path="../escritorio-sdk/WebContent/administracion/modulos"/>
<wb-resource deploy-path="/administracion/modulos/altaConfiguraciones" source-path="../escritorio-
sdk/WebContent/administracion/modulos/altaConfiguraciones"/>
<wb-resource deploy-path="/administracion/modulos/listadosConfiguracion" source-path="../escritorio-
sdk/WebContent/administracion/modulos/listadosConfiguracion"/>
<wb-resource deploy-path="/administracion/modulos/mantenimientoModulos" source-path="../escritorio-
sdk/WebContent/administracion/modulos/mantenimientoModulos"/>
<wb-resource deploy-path="/administracion/modulos/procedimientosXSD" source-path="../escritorio-
sdk/WebContent/administracion/modulos/procedimientosXSD"/>
<wb-resource deploy-path="/administracion/modulos/propiedadesModulos" source-path="../escritorio-
sdk/WebContent/administracion/modulos/propiedadesModulos"/>
<wb-resource deploy-path="/administracion/modulos/tareasXSD" source-path="../escritorio-
sdk/WebContent/administracion/modulos/tareasXSD"/>
<wb-resource deploy-path="/administracion/sincronizacion" source-path="../escritorio-
sdk/WebContent/administracion/sincronizacion"/>
<wb-resource deploy-path="/administracion/visibilidad" source-path="../escritorio-sdk/WebContent/administracion/visibilidad"/>
<wb-resource deploy-path="/agenda" source-path="../escritorio-sdk/WebContent/agenda"/>
<wb-resource deploy-path="/agenda/funciones" source-path="../escritorio-sdk/WebContent/agenda/funciones"/>
<wb-resource deploy-path="/decorators" source-path="../escritorio-sdk/WebContent/decorators"/>
<wb-resource deploy-path="/editor" source-path="../escritorio-sdk/WebContent/editor"/>
<wb-resource deploy-path="/editor/config" source-path="../escritorio-sdk/WebContent/editor/config"/>
<wb-resource deploy-path="/editor/config/fuentes" source-path="../escritorio-sdk/WebContent/editor/config/fuentes"/>
<wb-resource deploy-path="/editor/config/imagenes" source-path="../escritorio-sdk/WebContent/editor/config/imagenes"/>
<wb-resource deploy-path="/editor/css" source-path="../escritorio-sdk/WebContent/editor/css"/>
<wb-resource deploy-path="/editor/imagenes" source-path="../escritorio-sdk/WebContent/editor/imagenes"/>
<wb-resource deploy-path="/editor/imagenes/editor" source-path="../escritorio-sdk/WebContent/editor/imagenes/editor"/>
<wb-resource deploy-path="/editor/imagenes/editor/letras" source-path="../escritorio-
sdk/WebContent/editor/imagenes/editor/letras"/>
<wb-resource deploy-path="/editor/imagenes/fondos" source-path="../escritorio-sdk/WebContent/editor/imagenes/fondos"/>
<wb-resource deploy-path="/editor/imagenes/popup" source-path="../escritorio-sdk/WebContent/editor/imagenes/popup"/>
<wb-resource deploy-path="/editor/js" source-path="../escritorio-sdk/WebContent/editor/js"/>
<wb-resource deploy-path="/error" source-path="../escritorio-sdk/WebContent/error"/>
```

4.2 Tecnología utilizada en Plataforma de Tramitación.

La tecnología de desarrollo utilizada en el proyecto Plataforma de Tramitación es Struts 2.

Struts es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC en la plataforma J2EE. Una de las características principales es que permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas que emplean Java lo convierte en una herramienta muy utilizada en desarrollos de este tipo.

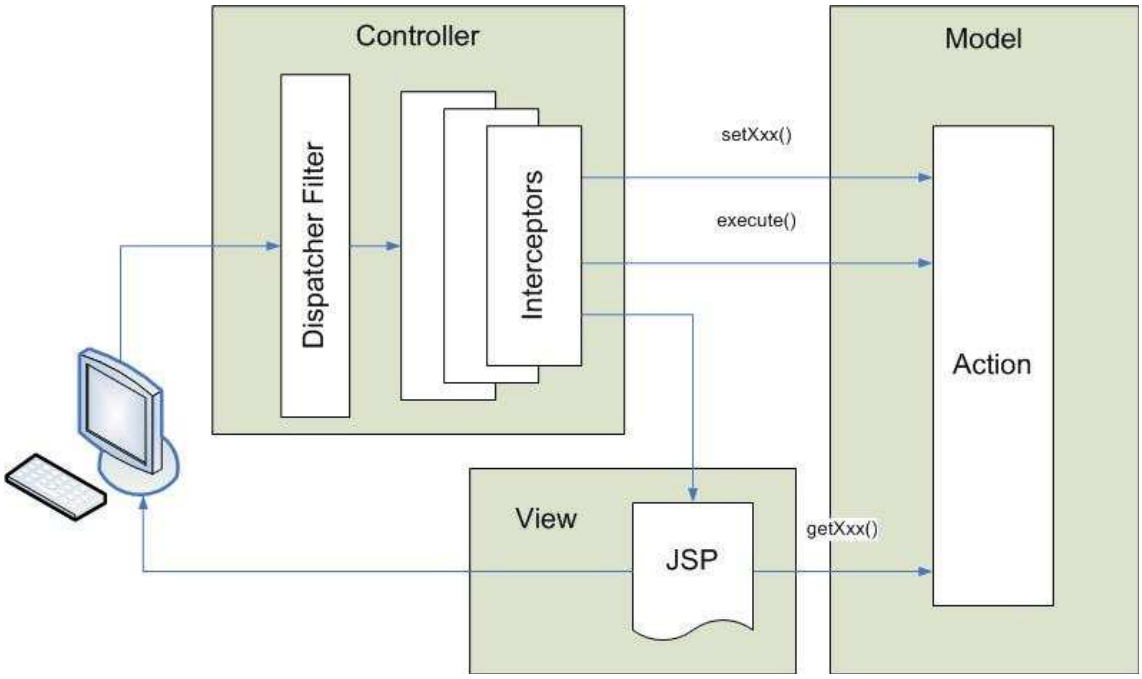


Ilustración 1: Patrón MVC

El objetivo de Struts es brindar soporte para el desarrollo de aplicaciones Web bajo el patrón MVC. La filosofía de trabajo se basa en 3 conceptos "construir" (build), "desplegar" (deploy) y "mantener" (maintain).

Tanto Struts 1 como Struts 2 proveen tres componentes claves:

- Un "request handler" que mapea clases Java a URIs de aplicaciones Web.
- Un "response handler" que mapea nombres lógicos a páginas del servidor u otros recursos web.
- Un conjunto de tags que permiten crear aplicaciones robustas basadas en formularios. En Struts 2, las tres componentes han sido rediseñadas y mejoradas manteniendo las mismas características de la arquitectura que en Struts 1.

Struts 2, la tecnología utilizada en el desarrollo de Plataforma Tramitación, presenta las siguientes características propias:

- **Diseño mejorado:** Todas las clases de Struts 2 están basadas en interfaces. Las interfaces del núcleo son independientes de la arquitectura HTTP.
- **Valores por defecto:** La mayoría de los elementos de configuración poseen un valor por defecto que puede ser fijado (y no hay que preocuparse por configurarlo nuevamente).
- **Resultados mejorados:** A diferencia de los antiguos "ActionForward" de Struts 1, los resultados de Struts 2 son tipificados permitiendo preparar el response en los casos que sea necesario.
- **Tags mejorados:** Con las nuevas etiquetas de Struts 2, es posible crear páginas consistentes con menor escritura de código.
- **Soporte para Ajax:** El "theme" ajax permite a las aplicaciones interactuar con el servidor mediante requerimientos asincrónicos.
- **Inicio rápido:** Muchos cambios pueden ser realizados dinámicamente sin ser necesario el reinicio del contenedor web.
- **Test de las Acciones:** Las acciones de Struts 2 son independientes de la arquitectura HTTP con lo cual pueden ser fácilmente testeadas mediante pruebas de unidad sin la necesidad de recurrir a objetos que simulen el comportamiento de los objetos reales.
- **Integración con Spring:** Las acciones pueden ser creadas como beans de Spring.
- **Plugins:** Las extensiones de Struts 2 pueden ser agregadas simplemente copiando un "jar". No es necesaria una configuración manual.
- **Formularios:** En Struts 2 no existe más el concepto de formulario tal como existía en Struts 1. Ahora es posible usar cualquier JavaBeans contenido en una acción o escribir directamente sobre los atributos de una acción. Por otro lado, ya no es necesario que todas las propiedades sean sólo String. Struts 2 convierte automáticamente los valores cargados en el formulario web a los tipos de valores que posea el JavaBean referenciado.
- **Acciones POJO:** Cualquier clase puede ser usada como una acción. Ya no es necesario implementar una interfaz o subclasificar alguna clase del framework.

4.3 Servicios

Los servicios son clases Java encargadas de gestionar la información, separando la lógica de datos de la lógica de negocio, cumpliendo así con el patrón MVC (Modelo – Vista – Controlador).

Para hacer que los servicios sean flexibles y puedan ser modificados de forma sencilla, inicialmente se creará una interfaz Java donde se definirán los métodos necesarios para interactuar con la lógica de datos de la forma que convenga y si fuese necesario, realizar tantas clases que implementen dicha interfaz permitiendo la comunicación con la lógica de datos de múltiples formas.

El primer paso es crear una interfaz Java en la que se definan los métodos que va a ofrecer el servicio como se muestra en el siguiente ejemplo:

```
package es.juntadeandalucia.plataforma.interfaces.alta;

public interface IAltaExpedientesService extends IConfigurableService {
        String generacionNumeroAlta(String tipoProcedimiento, UsuarioWeb user);
        void setOtrosDatos(String otrosDatos);
    String getOtrosDatos();
    boolean tienePermisos(String idUser, String permiso);
    boolean borrarExpediente(TpoPK idExpediente) throws TrException;
}
```

Una vez definida la interfaz, es necesario crear una clase Java que implemente dicha interfaz.

Una vez creado el servicio, es necesario definir un fichero llamado applicationContext-xxxx.xml que contenga los servicios creados..

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE beans
      PUBLIC "-//SPRING//DTD BEAN//EN"
      "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
        <!– Servicios disponibles en la plataforma de tramitación –>
```

```
    <bean id="altaExpedienteService"
    class="es.juntadeandalucia.plataforma.services.alta.AltaExpedientesServiceImpl"
    singleton="false">
            <property name="tramitacionService">
                <ref bean="tramitacionService"/>
            </property>
            <property name="cacheApiService">
                <ref bean="cacheApiTramitadorService"/>
            </property>
      </bean>

    <bean id="tareasService"
    class="es.juntadeandalucia.plataforma.services.tareas.TareasServiceImpl" singleton="false">
            <property name="cacheApiService">
                <ref bean="cacheApiTramitadorService"/>
            </property>
      </bean>
</beans>
```

En este fichero se define el identificador del servicio, mediante la etiqueta "bean id", así como la ruta en la que se encuentra la clase que implementa dicho servicio, mediante la etiqueta "class".

Por último, cuando se precise utilizar servicios ya creados, se incluirá en la definición de las clases que vayan a usarlos el identificador del servicio, es decir, debe aparecer en la etiqueta "ref bean" dicho identificador. En el caso mostrado, el servicio altaExpedienteService utiliza los servicios tramitacionService y cacheApiTramitadorService.

4.4 Alta Expediente

En el caso que la solicitud de expediente se realice de forma presencial o de oficio, será necesario desde Plataforma de Tramitación dar de alta el expediente en cuestión.

En el menú principal de Plataforma de Tramitación, existe la opción "*Alta Expediente*" como se muestra en la siguiente ilustración:

El proceso de Alta de expediente se divide en dos partes fundamentales:

- Una ventana de alta genérica donde se solicitan los siguientes datos:
  - Tipo de Procedimiento **(Obligatorio)**
  - Fecha de alta de expediente. Campo que se genera de forma automática.
  - Título del expediente **(Obligatorio)**. Nombre del expediente
  - Unidad orgánica **(Obligatorio)**. Unidad orgánica encargada de tramitar el expediente.
  - Observaciones **(Opcional)**.

- En función del tipo de procedimiento seleccionado en la ventana de Alta Genérica, se redirigirá la aplicación a una ventana de Alta específica que contendrá los datos necesarios del tipo de procedimiento seleccionado.

Para realizar un alta de expediente será necesario modificar y crear los siguientes ficheros:

- Fichero de propiedades "xxx.properties". En este fichero será necesario incluir una propiedad donde se incluya el identificador del procedimiento en Trew@. Con este identificador, se podrá diferenciar el tipo de procedimiento para redirigir la aplicación a la ventana de alta específica, el fichero de propiedades utilizado para el alta de expedientes es *procesoAltaExpedientesAction.properties*. A continuación se muestra un ejemplo de una propiedad introducida para un tipo de procedimiento:

```
#IDs actuales de los procedimientos
LicenciaObra=1  //identificador del procedimiento en Trew@
```

- altaXXX.jsp. Formulario de adquisición de datos específico para cada procedimiento. En este formulario se debe definir mediante las etiquetas de Struts, el nombre del action que se ejecutará cuando se realice el "submit" del formulario. La página JSP necesaria para el alta, se ha denominado *altaLicenciaObra.jsp*, para el procedimiento "Licencia de Obra". A continuación se muestra un ejemplo de una parte de un formulario donde está definida la acción (action) que se ejecutará cuando se realice el "submit":

```
<s:form id="moduloAltaExp" name="formulario" theme="simple" method="post"
enctype="multipart/form-data" action="guardaExpedienteLicObr" validate="true"
cssStyle="width:100%" >
```

- strutsXXX.xml. En este fichero se incluyen las reglas de navegación entre la ventana del alta genérica de expediente y la ventana del alta específica de cada procedimiento indicado en el formulario de adquisición de datos específico ".jsp". Se define la relación entre el nombre del action, el nombre de la clase java y el método en el cual se van a tratar los datos. En este caso se ha denominado *struts-alta-expediente.xml*. A continuación se muestra un ejemplo de las reglas de navegación del fichero struts-alta-expediente.xml:

```
<action name="guardaExpedienteLicObr" method="guardaExpediente"
        class="procesoAltaExpedientesLicenciaObrasAction">

          <result name="input">altaLicenciaObra.jsp</result>

          <result name="error">/administracion/error.jsp</result>

        <result name="success"
        type="redirect">/busqueda/irABuscadorGenerico2.action?
```

```
            mostrarNumeroExpedienteCreado=${numero}&amp;
            refExpedienteCreado=${refExpedienteCreado}</result>
            <interceptor-ref name="defaultStack" />
        </action>
```

En el código anterior se indica, que cuando se invoque la acción "action name", se debe ejecutar el método que se le ha indicado en la etiqueta "method" que pertenece a la clase definida en la etiqueta "class".

- struts.xml: en este fichero se deben incluir los struts previos, es decir, aquellos strutsXXX.xml que se han ido creando. Así, cuando se inicie la aplicación, mediante este fichero, se irán mapeando todas las acciones incluidas en los diversos strutsXXX.xml. A continuación se presenta la manera de incluir los diferentes struts que se han creado en este fichero.

```
<include file="struts-alta-expediente.xml"/>
```

- XXX.java. En esta clase es necesario definir un método con el mismo nombre asignado a la etiqueta "method" en la regla del action del fichero strutsXXX.xml. Ese método tendrá que realizar todo lo necesario para crear el expediente tanto en Trew@ como en el cliente Solr. El cliente Solr es un servicio de indexación y búsqueda, siendo necesario enviarle la información tal y como se encuentre definida en el fichero "schema.xml". Para crear el expediente, esta clase usará los servicios necesarios para interactuar con Trew@.

  A continuación se muestra parte de código de un método definido en una clase denominada *ProcesoAltaExpedientesLicenciaObrasAction.java*.

```
public class ProcesoAltaExpedientesLicenciaObrasAction extends    ConfigurableAction
implements SessionAware{

public String guardaExpediente() throws  BusinessException, ArchitectureException {

    if (!resultado.equals("success"))
            return "error";
    else{
            configurarServicio(user, consultaExpedienteService);
           nuevoExpediente =
           consultaExpedienteService.obtenerExpedientes(altaExpedientesService.getIdExpe
           dienteCreado().toString(), null, null);

            if (nuevoExpediente == null)
                    return "error";
            else{
                numero= nuevoExpediente.get(0).getNumeroExpediente();
```

```
                    setRefExpedienteCreado(nuevoExpediente.get(0).getRefExpediente());

                    setNumExpCreado(nuevoExpediente.get(0).getRefExpediente());
                        asociarInteresado();
                        return resultado;
                        }
                }
            }
        }
```

- applicationContext-XXX.xml: en este fichero se definen los identificadores de las clases Java, así como todos los servicios empleados por dichas clases. Para tener una aplicación organizada y estructurada se definen dos ficheros de este tipo, uno de ellos que contenga todas las acciones y otro que contenga todos los servicios. En este caso se han definido dos ficheros. Por un lado se ha creado el fichero *applicationContext-actions.xml*, donde se definen las clases Java que gestionarán las altas de los diferentes tipos de procedimientos y las acciones necesarias en las tareas. Mientras que por otro lado se ha creado el *applicationContext.xml*, donde se definen las clases Java que implementan los servicios desarrollados por y para el proyecto vertical.

  A continuación se muestra un ejemplo del *applicationContext-actions.xml:*

```
<bean id="procesoAltaExpedientesLicenciaObrasAction"
class="es.juntadeandalucia.plataforma.actions.ProcesoAltaExpedientesLicenciaObrasAction"
singleton="false">
        <property name="consultaExpedienteService">
                <ref bean="consultaExpedienteService"/>
        </property>
        <property name="altaExpedienteService">
                <ref bean="altaExpedienteService"/>
        </property>
        <property name="gestionInteresadosService">
                <ref bean="gestionInteresadosService"/>
        </property>
</bean>
```

  y del *applicationContext.xml*

```
<bean id="altaExpedienteService"
class="es.juntadeandalucia.plataforma.services.alta.AltaExpedientesServiceImpl"
singleton="false">
            <property name="tramitacionService">
                    <ref bean="tramitacionService"/>
            </property>
```

```
            <property name="cacheApiService">
                <ref bean="cacheApiTramitadorService"/>
            </property>
        </bean>

    <bean id="tareasService"
    class="es.juntadeandalucia.plataforma.services.tareas.TareasServiceImpl" singleton="false">
            <property name="cacheApiService">
                <ref bean="cacheApiTramitadorService"/>
            </property>
        </bean>
```

- XXX-validation.xml. Fichero donde se realizan las validaciones de los campos requeridos en el formulario del alta de expediente. El fichero debe tener el mismo nombre que la clase java donde se manejan los campos de la página JSP del alta en este caso: *procesoAltaExpedientesAction-validation.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>

    <!DOCTYPE validators PUBLIC

            "-//OpenSymphony Group//XWork Validator 1.0.2//EN"

            "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

    <validators>
      <!– field name="selTipoExpediente">
            <field-validator type="requiredstring">
                <param name="trim">true</param>
                <message>Debe seleccionar el tipo de expediente</message>
            </field-validator>
      </field –>

      <field name="selProcVersion">
            <field-validator type="requiredstring">
                <param name="trim">true</param>
                <message>Debe seleccionar el procedimiento</message>
            </field-validator>
      </field>

      <field name="fecha">
            <field-validator type="requiredstring">
                <param name="trim">true</param>
```

```
            <message>Debe seleccionar la fecha de alta</message>
        </field-validator>
    </field>

    <field name="titulo">
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>El título del expediente es requerido</message>
        </field-validator>
    </field>
</validators>
```

4.5 Tareas

En Trew@, existen 4 tipos de tareas que se enumeran a continuación:

1. Tareas de incorporar documentos.
2. Tareas de generación de documentos.
3. Tareas Web o de adquisición de datos.
4. Tareas tipo Otros.

A continuación se detallan las actividades a realizar para la implementación de las diferentes tareas en la Plataforma de Tramitación.

4.5.1 Tareas tipo Incorporar documento

Para las tareas tipo incorporar no es necesario ningún desarrollo en la Plataforma de Tramitación. Una vez modelada la tarea en Trew@, automáticamente en el módulo funcional de "Documentos y Tareas a realizar" se obtendrá la tarea para incorporar documentos.

4.5.2 Tareas tipo Generar documento

Una vez modelada este tipo de tarea en Trew@, en el módulo funcional de "Documentos y Tareas a realizar" se obtendrá la tarea para generar documentos. El documento generado es una plantilla en WebOffice con un conjunto de variables.

El único desarrollo necesario en las tareas de generación de documentos es el cálculo de variables presentes en dicho documento. Estas variables tienen el formato $$Nombre_de_la_variable$$. En Trew@, cada variable tiene asociada una clase Java y un método. Será necesario definir la clase y el método de forma que dicho método devuelva la cadena de caracteres que sustituirá a esa variable.

4.5.3 Tareas Web o de adquisición datos y tipo Otros

Para estos tipos de tareas será necesario programación. Se utilizará un servicio llamado *tareasService*. Dicho servicio se encargará de obtener y guardar toda la información necesaria para realizar las tareas tipo web y otros.

Inicialmente se debe crear un fichero struts-xxxxx.xml en el que se encuentran definidas todas las tareas, debiendo incluirse posteriormente dicho fichero en el fichero *struts.xml*. A modo de ejemplo, para el procedimiento "Licencia de Obra", se ha definido el fichero *struts-config-tareasLicObr.xml*, mostrando a continuación una parte del fichero:

```
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>

        <constant name="struts.objectFactory" value="spring" />
        <constant name="struts.devMode" value="true" />

        <package name="tareas-licenciaObras" namespace="/agenda/tareas"
                extends="struts-default">

                <global-results>
                        <result name="error">/administracion/error.jsp</result>
                </global-results>

                <action name="tareaEntradaLICOBRA_O_CONFIRMAR_CIERRE"
                        method="inicio" class="tareaLICOBRAction">
                        <result name="success">
                                tareaEntradaLICOBRA_O_CONFIRMAR_CIERRE.jsp
                        </result>
                        <interceptor-ref name="basicStack" />
                </action>

                <action name="tareaEntradaLICOBRA_O_ENVIAR_NOTIFICACION"
                        method="inicio" class="tareaLICOBRAction">
                        <result name="success">
                                tareaEntradaLICOBRA_O_ENVIAR_NOTIFICACION.jsp
                        </result>
                        <interceptor-ref name="basicStack" />

                </action>
```

Será necesario incluir este fichero en el struts.xml de la siguiente manera:

```
<include file="struts-config-tareasLicObr.xml"/>
```

Por último, para tareas Web o de Adquisición de datos, además de definir la regla de Struts para mostrar la página JSP deseada, es necesario crear otro "action" en el fichero struts-xxxxx.xml donde se

definirá una clase y un método donde se tratarán los datos introducidos en el formulario, de forma similar al proceso de Alta de Expediente.

Para realizar una tarea de tipo Otros, primero es necesario definir un "action" con el mismo nombre que tenga la tarea en Trew@, en el fichero struts-xxxxx.xml. Mediante esa regla, se indicará la página Web que debe mostrarse. En este tipo de tareas se muestra un mensaje con la descripción de la tarea indicada en Trew@. La siguiente página JSP también es necesaria desarrollarla.

A continuación se muestra una página JSP tipo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title><s:text name="Necesario realizar la tramitación por lotes" /></title>
<!– HOJAS DE ESTILOS –>
<link rel="stylesheet" type="text/css"    href="../../instalaciones/css/hojaEstiloAplicacion.css" />
</head>
<body>
        <s:div id="capa_ejecutarTarea">
                <center>
                        <div id="contenedorCabecera">
                                <div id="cabeceraPaginaAplicacion"></div>
                                <div id="rellenoCabeceraPagina"></div>
                        </div>
                </center>

                <s:div id="espacioAlto"> 
                </s:div>
                <br>
                <s:div id="bordePpal" cssStyle="center">
                </s:div>

                <s:div id="nuevaTarea" cssStyle="width:100%;text-
align:left;height:100px;overflow:auto;" theme="simple">
                        <s:property value="tarea"/>
                </s:div>
                <s:div cssStyle="width:50%;text-align:center;">
                        <s:submit value="Aceptar" onclick="self.close()"/>
          </s:div>
```

```
        </s:div>
</body>
</html>
```

En esta página JSP, el campo más importante es <s:property value="tarea"/>, ya que al cargar el JSP, se realizará la llamada a un método llamado getXXXX(), en este caso sería getTarea(), definido en la clase indicada en el action, en este caso class="tareaLICOBRAction",y se mostrará el resultado devuelto por dicho método.

4.6 Condiciones

La programación de una condición consiste en habilitar una de varias transiciones en función del valor de una variable obtenida en una tarea web o en el alta del expediente.

Inicialmente hay que definir la condición en TREWA, indicando el nombre de la condición a implementar, la ruta del paquete en la que se va a encontrar y el nombre de la función que ejecutará la lógica para decidir si se cumple o no la condición, habilitando o deshabilitando la condición.

Una vez que se haya definido, lo único que se debe hacer es crear la clase que se ha indicado e implementar el método para que, en función de esa variable, devuelva un '1' o un '0'. Si devuelve '1' significa que la condición se ha cumplido y la variable tiene el valor esperado, mientras que si devuelve '0', la variable no tiene el valor esperado y se deshabilita la condición.

A continuación se muestra un ejemplo de la codificación de una condición del procedimiento "Licencia de Obra".

```
public Integer obtTrPRIOCU_COND_RESOLUCION_POSITIVA (BigDecimal id){
        Integer cond = 0;
        try{
                datosXml = apiui.obtenerOtrosDatos(new TpoPK(id), "E");
                OtrosDatosExpedientesLICOBR nuevos = new OtrosDatosExpedientesLICOBR
(datosXml);
                if(nuevos.getPriocu_res_acta().equalsIgnoreCase("Positiva"))
                        cond = 1;
        }
        catch(Exception e){
                cond = 0;
        }
        return cond;
}
```

4.7 Acciones

Las acciones son eventos lanzados cuando ocurre un hecho indicado en TREWA y a partir del cual se puede realizar lo que sea necesario para el correcto funcionamiento del procedimiento. Se pueden asociar las acciones a tareas o a transiciones.

Para programarlas se hace exactamente lo mismo que en el caso de las condiciones, explicadas en el punto anterior, es decir, definir en TREWA el nombre de la acción, el paquete y el nombre del método que será necesario implementar.

Y al igual que las condiciones, una vez definidas, lo único que se debe hacer es crear la clase que se ha indicado e implementar el método para que, una vez se realice la llamada al mismo, devuelva un '1' indicando que se ha ejecutado correctamente toda la lógica implementada en la acción.

A continuación se muestra un ejemplo de la codificación de una acción del procedimiento "Licencia de Obra".

```
public String obtAcLICOBRA_AC_INCORPORA_JUR(BigDecimal id) {

	try{
		datosXml = apiui.obtenerOtrosDatos(new TpoPK(id), "E");
		OtrosDatosExpedientesLICOBR nuevos = new OtrosDatosExpedientesLICOBR
(datosXml);
		nuevos.setLicobra_doc_jur("S");
		apiui.actualizarOtrosDatos(new TpoPK(id), "E", nuevos.toString());
	}
	catch(Exception e){
		System.out.println("Error en obtAcLICOBRA_AC_INCORPORA_JUR()");
	}
	return "1";
}
```

4.8 Tareas tipo Web mediante XSD

Además del procedimiento que se describe en este documento, existe otra forma de generar formularios de adquisición de datos mediante el estándar XSD. Inicialmente, es necesario crear un fichero con extensión ".xsd" que tenga la misma estructura que los otros datos que vaya a contener el procedimiento. El campo "Otros datos", es un campo que se introduce en la tabla TR_EXPEDIENTES de Trew@ donde se guardan todos los datos asociados al expediente que han sido introducidos vía tareas de adquisición de datos y en las ventanas de alta de expediente.

Un fichero ".xsd" consiste en un documento XML que mantiene un esquema o esqueleto de cómo construir otras instancias de documento. Como ejemplo se utilizará el ".xsd" creado para el procedimiento "*Licencia de Obra*" .

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="OtrosDatosExpedientesLICOBR">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="tipoActuacion" type="xs:string" />
        <xs:element minOccurs="1" name="telefono" type="xs:string" />
        <xs:element minOccurs="1" name="unidadAfectada" type="xs:string" />
        <xs:element minOccurs="1" name="fechaIncidencia" type="xs:string" />
        <xs:element minOccurs="1" name="motivo" type="xs:string" />
        <xs:element minOccurs="1" name="fechaElaboracionDoc" type="xs:string" />
        <xs:element minOccurs="1" name="municipio" type="xs:string" />
        <xs:element minOccurs="1" name="aplicacionPresupuestaria" type="xs:string" />
        <xs:element minOccurs="1" name="nif" type="xs:string" />
.....
        <xs:element minOccurs="1" name="priocu_fecha_acuse" type="xs:string" />
        <xs:element minOccurs="1" name="priocu_res_recursos" type="xs:string" />
        <xs:element minOccurs="1" name="priocu_rec_presentados" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Las etiquetas más importantes de este fichero son:

- <xs:element name="OtrosDatosExpedientesLICOBR">
  La etiqueta name tiene que ser igual al valor que se le haya puesto en la etiqueta name del fichero *.betwitx, es decir, a

  <element name="OtrosDatosExpedientesLICOBR">

  Perteneciente al fichero OtrosDatosExpedientesLICOBR.betwitx. Mediante un fichero betwitx se define el conjunto de variables y la estructura de las mismas que serán almacenadas en el campo "Otros Datos" de una tabla de la base de datos de TREW@.

- Entre las etiquetas <xs:complexType> </xs:complexType> deben definirse todas las variables que hayan sido definidas dentro del fichero *.betwitx. Para el ejemplo anterior se define el elemento:

  <xs:element minOccurs="1" name="tipoActuacion" type="xs:string" />

  Que representa al elemento definido en nuestro fichero OtrosDatosExpedientesLICOBR.betwitx

  <element name="tipoActuacion" property="tipoActuacion"/>

- Además, todas las variables tienen que tener el mismo tipo, es decir type="xs:string"

Una vez que se haya generado el fichero .xsd será necesario asociarlo al procedimiento, accediendo a la herramienta de administración y seleccionando en el menú la opción "*Asociar plantilla de tarea a un procedimiento*" como se muestra en la siguiente ilustración:

Ilustración 2. Menú de la herramienta de administración

Una vez elegida la opción, aparecerá la siguiente ventana:

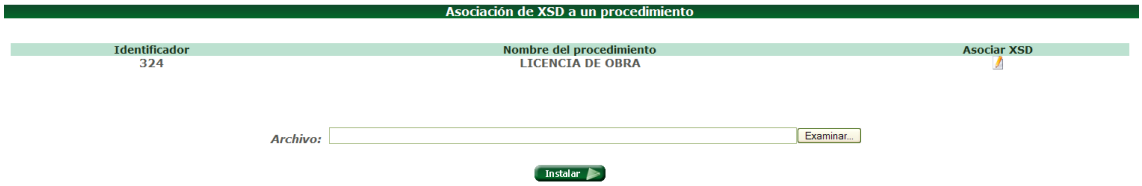


Ilustración 3. Asociación de plantilla xsd a un procedimiento

Pulsando en el icono de la columna "**Asociar XSD**" aparecerá la ventana inferior en la que buscar el fichero .xsd a asociar.

Una vez asociado el fichero XSD, será necesario construir el formulario con el que se permita al usuario introducir datos en el expediente. Para eso será necesario definir diferentes ficheros *.xml, que simularán a ventanas JSP. Para facilitar la creación de los ficheros XML, es necesario disponer de un programa de edición de XML, capaz de validar los ficheros XML a partir de un fichero XSD.

A continuación se presenta un fichero XML de ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<formulario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///C:/Documents%20and%20Settings/jasperap/Escritorio/Pruebas%20XSD/validacionTareasXSD.xsd">
```

```
  <componente idCampo="licobra_tasa" tipoValidacion="string" multivalor="no"
referenciaDinamica="string" tipo="select">
    <select>
      <etiqueta>Tasa Pagada</etiqueta>
      <valor>Si</valor>
      <valor>No</valor>
    </select>
   </componente>
</formulario>
```

Destacar que será obligatorio borrar el comentario inicial que por defecto, el programa *"Liquid XML Studio"* introduce en la creación de ficheros XML. Además, para la construcción de los ficheros XML se utilizará un fichero XSD que seguirá una determinada estructura. El fichero XSD se denomina **validacionTareasXSD**".

Es necesario que exista un campo "*idCampo*" que coincida con uno de los elementos definidos en el XSD asociado al procedimiento y dentro de la etiqueta componente se define el tipo de componente, que puede ser uno de los que se encuentran en la plantilla XSD que se usa para validar. Una vez creado el XML, hay que asociarlo a una tarea de una fase concreta. Para ello, desde el menú de la herramienta de la administración y seleccionando la opción "*Asociación de tareas automáticas*" como se muestra en la *Ilustración 4.* aparecerá la siguiente ilustración:

Ilustración 4. Asociación de xml a una tarea

Existen dos opciones posibles de selección:

- **Añadir un nuevo formulario mediante un XML.**

Permite agregar ***sólo*** un formulario ya creado y definido por el diseñador. Será necesario seleccionar la tarea permitida a la que desea asociar el formulario.

- **Añadir un nuevo formulario mediante un ZIP.**

Permite incluir, de una sola vez, una batería completa de formularios. En este caso, se comprueba en primer lugar, la validez del fichero comprimido seleccionado por el usuario administrador, y en segundo lugar, para cada uno de los ficheros contenidos, se verifica su validez semántica.

En este caso, es ***imprescindible*** que el formulario generado contenga ***obligatoriamente*** relleno un metacampo indicador de la tarea permitida a la que se quiera asociar. El nombre de este metacampo es ***'nombreFase'***.

4.9 Desarrollo de Utilidades en Plataforma de Tramitación.

4.9.1 Introducción

Las utilidades son herramientas, disponibles desde la plataforma de tramitación, que añaden funcionalidad adicional para el usuario, como por ejemplo el acceso al applet de Model@ para seguir el flujo del procedimiento, la posibilidad de adjuntar un documento al expediente en cualquier fase del mismo o mostrar la evolución del expediente.

En este apartado se detallará cómo desarrollar una nueva utilidad en Plataforma de Tramitación, indicando los ficheros necesarios para su construcción.

4.9.2 Especificaciones para la construcción de una nueva utilidad

Para el desarrollo de una nueva utilidad visible desde Plataforma de Tramitación, será necesario crear y/o modificar los siguientes ficheros:

- Un fichero XML de descripción de la utilidad (por ejemplo: *datos_expediente.xml*).

```
<modulo>
    <def-modulo>
            <id>101</id>
            <nombre>datos_expediente </nombre>
            <version>0.1</version>
            <titulo>Datos Expediente</titulo>
            <descripcion>Datos del expediente</descripcion>
            <url>../modulos/datosExpediente/datosExpediente.action</url>
            <icono-on>../agenda/imagenes/usuarios2.gif</icono-on>
            <icono-off>../agenda/imagenes/usuarios.gif</icono-off>
            <tipoInstalacion>UTILIDADES</tipoInstalacion>
            <autor>
                    <nombre>Javier</nombre>
                    <email>javier.cardenas.garcia@everis.com</email>
                    <grupo>staff</grupo>
            </autor>
            <type>STRUTS-2</type>
            <postFuncion></postFuncion>
    </def-modulo>
    <instancia>
            <alto>25%</alto>
            <ancho>-1</ancho>
            <visible>true</visible>
            <localizacion>2</localizacion>
            <posicion>-1</posicion>
```

```
    </instancia>
</modulo>
```

En este fichero se configuran datos relacionados con la utilidad y servirá para la instalación de la misma. Para que Plataforma de Tramitación pueda encontrar el fichero e instalarlo, éste debe estar en la siguiente ruta:

reas\WEB-INF\classes\es\juntadeandalucia\plataforma\modulospredefinidos

Entre los datos introducidos en este fichero se debe tener en cuenta principalmente:

- El **id** debe ser único.
- El **nombre** debe coincidir con el nombre del fichero (sin la extensión).
- La **url** debe contener la dirección del action que se lanzará al lanzar la utilidad y que estará definido en un fichero de struts que será necesario crear y que se describirá posteriormente.
- En **tipoInstalación** se indicará UTILIDADES.
- En **type** se indicará STRUTS-1 o STRUTS-2 dependiendo del framework utilizado.

- Un fichero de Struts-2: permite separar la capa de negocio de la capa de presentación, es decir, en este fichero se relaciona la ventana JSP de la utilidad con la clase que se encargará de toda la lógica de negocio, siendo este XML el punto de unión.

  Por ejemplo: *struts-datosExpediente.xml*

```
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
 <constant name="struts.objectFactory" value="spring" />
 <constant name="struts.devMode" value="true" />

<package name="datosExpediente"
  namespace="/modulos/datosExpediente" extends="struts-default">

     <action name="datosExpediente" method="datosExpediente"
     class="datosExpedienteAction">

            <result name="success">
                  /modulos/datosExpediente/datosExpediente.jsp
    </result>
            <result name="error">
                  /modulos/datosExpediente/datosExpedienteERROR.jsp
    </result>
            <interceptor-ref name="basicStack" />
    </action>
</package>
```

```
</struts>
```

- Es necesario añadir al fichero struts.xml, en la ruta del fichero anterior:

  Por ejemplo:

  ```
  <include file="struts/modulos/datosExpediente/struts-datosExpediente.xml"/>
  ```

- Una clase Java que contendrá toda la lógica de negocio de la utilidad. En ella se incluirán todos los métodos referenciados en el fichero de struts de la utilidad, además del resto de métodos auxiliares que sean necesarios.
- Un fichero JSP que será la página que se lanzará al pulsar sobre la utilidad. Para ello, en el fichero de struts de la utilidad es necesario definir un action que relacione un método definido en la clase Java anterior y la página JSP, de forma que el método devuelva un valor que lance la JSP.
- Añadir al fichero applicationContext-XXXX.xml que utilice Plataforma de Tramitación,, un nuevo "Bean" definiendo el identificador de la clase Java, así como todos los servicios que ésta use.

  Por ejemplo:

  ```
  <bean id="datosExpedienteAction"
  class="es.juntadeandalucia.plataforma.actions.modulos.datosExpediente. Action"
  singleton="false">
              <constructor-arg ref="tareasService"></constructor-arg>
              <constructor-arg ref="consultaExpedienteService"></constructor-arg>
              <constructor-arg ref="logService"></constructor-arg>
      </bean>
  ```

Con estos ficheros, se dispone de todo lo necesario para desarrollar cualquier utilidad, aunque en función de la dificultad y amplitud de la misma, el contenido de los ficheros y el número de éstos (por ejemplo, podría contener varios JSP) es variable.

Por último, es necesario indicar que en el fichero tr_funciones.jsp, es donde se definen las características de la ventana que se abrirá al pulsar sobre la utilidad indicando si se abrirá en un popup, en una ventana nueva, etc.

5 Notas sobre la Guía de Instalación del Entorno de Desarrollo de la Plataforma de Tramitación Electrónica

El presente documento puede estar sujeto a continuas mejoras por lo que se recomienda consultar la web de Soporte a la Administración Electrónica de la Junta de Andalucía (*plutón*, https://ws024.juntadeandalucia.es/pluton/index.jsp) para obtener la versión más actualizada del mismo con ejemplo, últimas actualizaciones en la especificación de requisitos para módulos funcionales, etc.