Junta de Andalucía

Plataforma de Tramitación w@ndA 2.5.7

Manual del Desarrollador

Versión: v01r01
Fecha: 17/10/2024

HOJA DE CONTROL

Proyecto	Plataforma de Tramitación w@ndA 2.5.7		
Documento	Manual del Desarrollador		
Nombre del Fichero	PTW257E_MDE_Manual_del_Desarrollador_PTw@ndA_v1r01.odt		
Autor	UTE		
Versión/Revisión	v01r01	**Fecha Versión**	17/10/2024
Aprobado por	ADA	**Fecha Aprobación**	30/07/24

REGISTRO DE CAMBIOS

Versión	Causa del Cambio	Responsable del Cambio	Área	Fecha del Cambio
v01r00	Creación del documento	UTE	UTE	30/07/24
v01r01	Corrección formato	UTE	UTE	17/10/2024

CONTROL DE DISTRIBUCIÓN

Nombre y Apellidos	Cargo	Área
Manuel Escobar Montes	Jefe de Servicio	ADA
José Ignacio Cortés Santos	Director de Proyecto	ADA
Felipe José Gallego Rivera	Director de Proyecto	ADA
María Luisa Rubio Campanario	Director de Proyecto	ADA
Alejandro Martínez Marcos	Director de Proyecto	ADA
Ignacio Gordillo Díaz	Jefe de Proyecto	UTE

ÍNDICE

1 INTRODUCCIÓN

La Plataforma de Tramitación w@ndA está diseñada bajo una arquitectura modular en la que es posible incorporar módulos funcionales cuyo objetivo sea implementar funcionalidades específicas del organismo en la que se implanta.

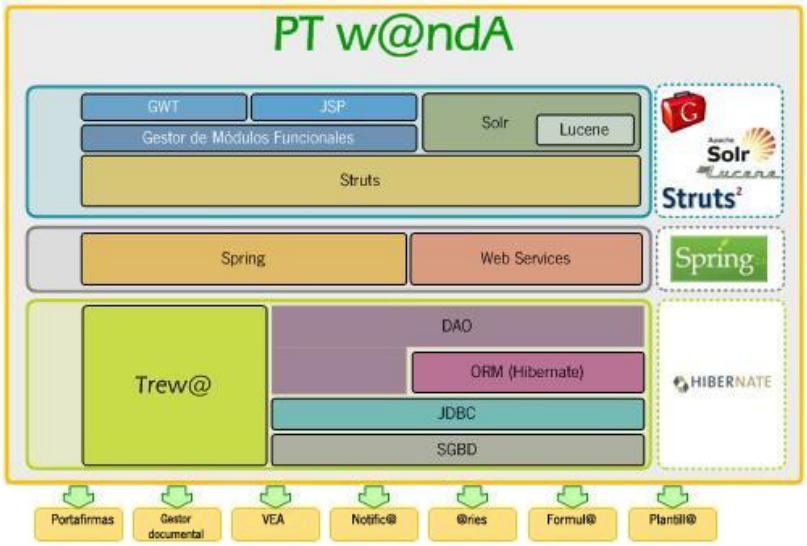
El objetivo del presente documento es detallar los pasos a seguir para instalar y configurar el entorno de desarrollo, en primera instancia para utilizar el componente Plataforma de Tramitación, y posteriormente para la construcción de componentes verticales sobre la propia Plataforma de Tramitación Electrónica w@ndA.

A continuación se describirán los requisitos necesarios para comenzar el desarrollo de módulos funcionales, la estructura que deben tener, y la forma particular de realizar el desarrollo. Durante el documento se describen recomendaciones y directrices a seguir para la correcta construcción del módulo funcional.

Por último, se incluye una guía de desarrollo de componentes funcionales específicos verticales particulares a procedimientos pertenecientes a las diversas familias de tramitación w@ndA, explicitando los distintos tipos de módulos existentes y los pasos a seguir para su construcción.

2 TECNOLOGÍA UTILIZADA EN PLATAFORMA DE TRAMITACIÓN W@NDA

La arquitectura tecnológica de desarrollo utilizada en el proyecto Plataforma de Tramitación w@ndA es la mostrada a continuación.



Debido a que el objeto de este documento es servir de manual de desarrollo para la construcción de módulos funcionales, se introducirá a continuación la tecnología que más utilizarán los desarrolladores de módulos: 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.

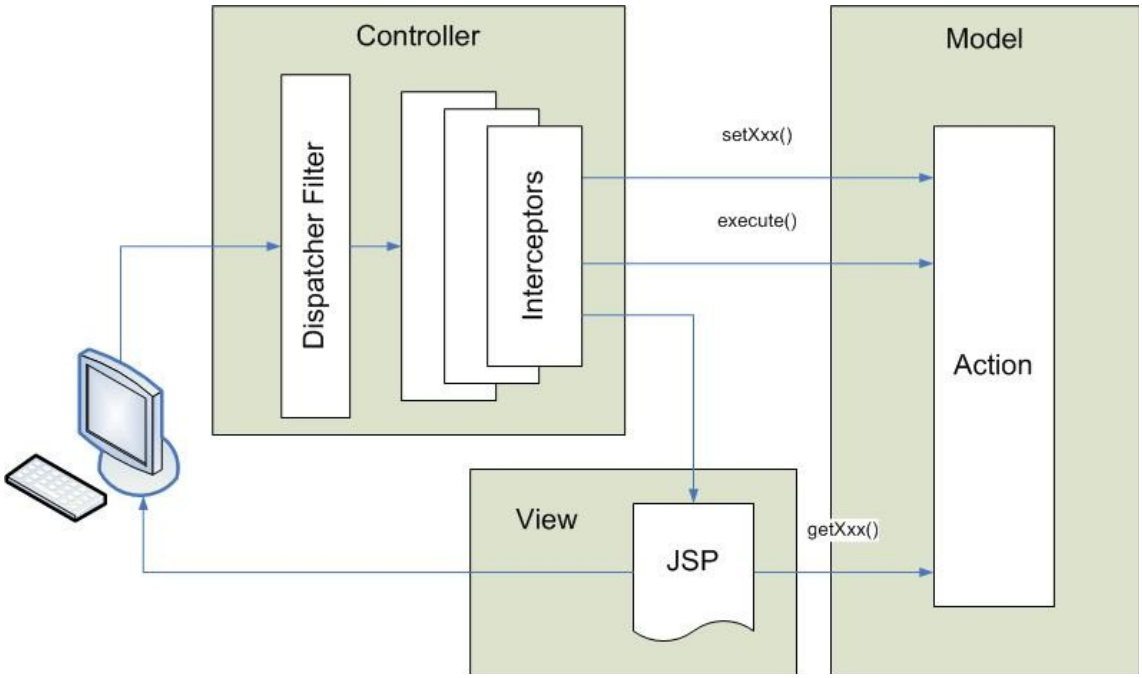


Ilustración 1: Patrón MVC

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).

Struts 2 proveen tres componentes claves:

- Un “request handler” que vincula clases Java a URIs de aplicaciones web.
- Un “response handler” que vincula nombres lógicos a páginas del servidor u otros recursos web.
- Un conjunto de etiquetas que permiten crear aplicaciones robustas basadas en formularios.

En Struts 2, los tres componentes han sido rediseñados y mejorados 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 en 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 asíncronos.

- **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 testadas mediante pruebas unitarias 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.

3 REQUISITOS PREVIOS

El desarrollo de módulos funcionales debe iniciarse cuando se cumplan los siguientes requisitos, necesarios para la correcta implementación de estos:

- Máquina virtual Java versión 8. URL de descarga: http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html
- Despliegue con el Servidor de Aplicaciones WildFly 15.0.1Final URL de descarga (enlace http://download.jboss.org/wildfly/1 5 .0.1.Final/wildfly- 1 5 .0.1.Final.zip)
- Acceso a base de datos con esquema compatible Trew@ 2.6.7.1
- Base de datos Oracle soportada por Hibernate 5.4.2 (10g, 11g) para el modelo de datos propio de la Plataforma de Tramitación, el cual se dividirá en dos esquemas: uno para las tablas paramétricas y otro para las configurables.
- Instalación completa de la aplicación Plataforma de Tramitación w@ndA en un entorno de desarrollo.
- Instalación completa de Trew@ con la definición completa y parametrización de procedimientos, usuarios, puestos de trabajo, organismos, etc… a utilizar en el sistema a utilizar para el desarrollo.
- SDK proporcionado con PTw@ndA para realizar el desarrollo de módulos funcionales.

Adicionalmente a los componentes w@ndA, puede opcionalmente requerirse el acceso a Formul@ para integrarse con el motor de formularios para desarrollar las tareas de manipulación de datos.

4 INSTALACIÓN DEL ENTORNO DE DESARROLLO

4.1 Instalación del servidor de aplicaciones WildFly

Acceder a la página de descarga de WildFly (http://download.jboss.org/wildfly/1 5 .0.1.Final/wildfly- 1 5 .0.1.Final.zip) y obtener la versión de WildFly 15.0.1 Final.

Descomprimir el .zip en un directorio local.

4.2 Instalación de Eclipse

Acceder a la página de descarga de eclipse http://www.eclipse.org/downloads/ y obtener la última versión del "Eclipse IDE for Java EE Developers".

Descomprimir el .zip en un directorio local.

4.3 Instalación de la Plataforma de Tramitación w@ndA

Para el desarrollo de módulos funcionales, se partirá de la última versión estable de la plataforma de tramitación. Para ello debe acceder a la página del área técnica de Plataforma de Tramitación https://ws024.juntadeandalucia.es/ae/adminelec/areatecnica/wanda y solicitar la descarga del cd con todos los recursos de Plataforma de Tramitación w@ndA.

- Las **librerías** utilizadas por la plataforma de tramitación. Entre ellas, destaca el fichero `ptwanda-core-XX.jar`, que contiene la lógica de negocio de la aplicación.
- **Páginas web** de la Plataforma de Tramitación w@ndA. Normalmente se tratan de ficheros JSP (*Java Server Pages*) utilizando el framework de presentación Struts 2.
- **Recursos web** de la aplicación: imágenes, hojas de estilo, librerías JavaScript.
- **Archivos** de **configuración**, tanto de la propia aplicación como de los diferentes *frameworks* utilizados en su construcción (Struts, Spring, Hibernate, log4java, etc.).

De la misma página del área técnica se podrá descargar las últimas versiones de los manuales de "Instalación y Despliegue" y "Administración".

Seguir paso a paso el Manual de Instalación y Configuración de PTw@ndA para su correcto despliegue.

4.4 Instalación de Apache Maven

Apache Maven es una herramienta de software para la gestión y construcción de proyectos. Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus

dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Para la instalación de apache Maven remitimos a la página oficial, concretamente en el siguiente enlace:

http://maven.apache.org/download.html

Uno de los pasos será configurar el repositorio al cuál se conectará Apache Maven para descargar las dependencias especificadas en los módulos funcionales, así como los plugins que necesite para realizar sus operaciones.

En el directorio conf, dentro del apache-maven utilizado, se halla un fichero settings.xml que deberá ser editado.

En el caso de utilizar un proxy para la conexión a Internet habrá que indicarlo en la etiqueta <proxies> de la siguiente manera:

```
<proxy>
     <id>idProxy</id>
     <active>true</active>
     <protocol>https</protocol>
     <host>5.2.0.123</host>
     <port>80</port>
     <nonProxyHosts>localhost</nonProxyHosts>
</proxy>
```

Todas las dependencias de la Plataforma de Tramitación son descargadas del repositorio de CHAP. Para hacer uso de este repositorio habrá que indicarlo en la etiqueta <mirrors>:

```
   <mirror>
          <id>archiva.default</id>
          <url>https://ws024.juntadeandalucia.es/maven/repository/internal</url>
          <name>Repositorio Maven SCAE/CJAP</name>
          <mirrorOf>*</mirrorOf>
   </mirror>
```

El repositorio de CHAP utiliza un certificado de FNMT que no viene por defecto en la jdk. Para ello será necesario descargarse este certificado de la url anterior e importarlo en el almacén de claves de la máquina virtual. En el punto 3.3.3.3 del manual de instalación de plataforma se explica cómo descargar e importar un certificado.

5 CONFIGURACIÓN DEL ENTORNO DE DESARROLLO PARA ECLIPSE

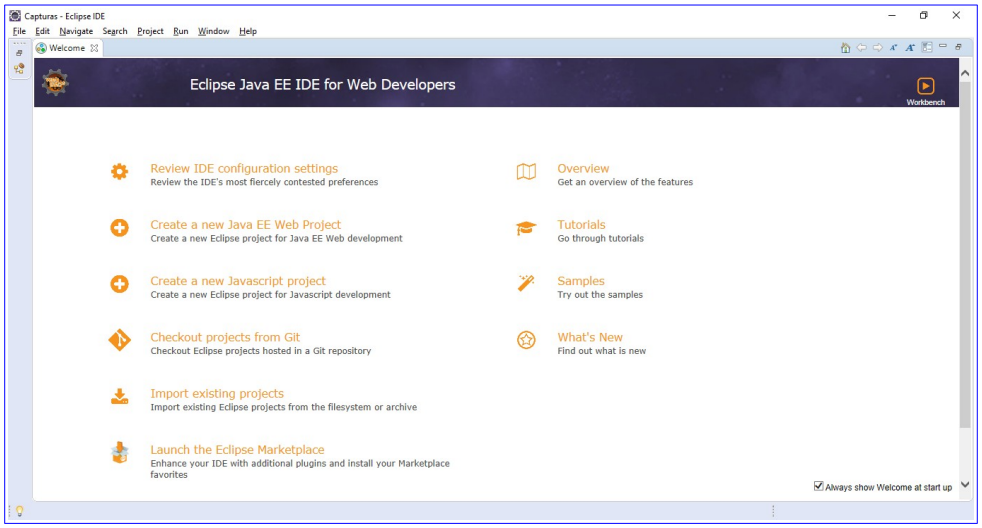
5.1 Introducción

El objetivo de esta sección es detallar los pasos a seguir para montar el entorno de desarrollo necesario para desarrollar módulos funcionales en PTw@ndA con Eclipse, detallar errores que se pueden dar, así como la configuración del servidor de aplicaciones WildFly, donde desplegar y posteriormente se ejecutará la Plataforma de Tramitación.

5.2 Importación de Plataforma de Tramitación

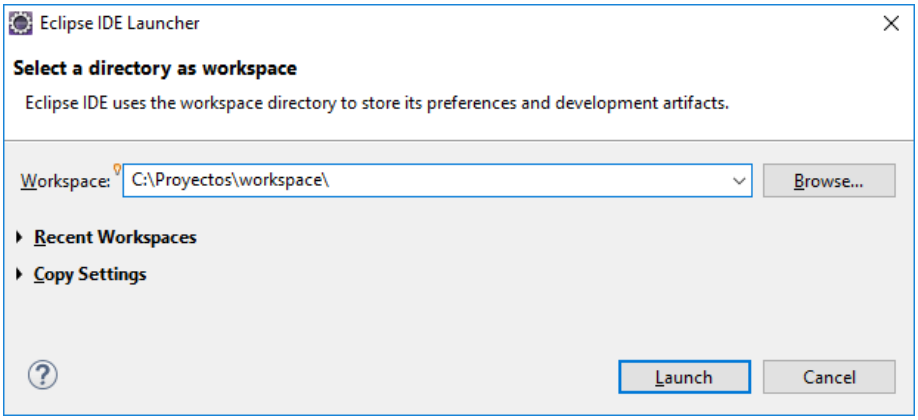
A continuación se especifican los pasos a seguir para importar el proyecto **ptwanda-web** en *Eclipse*.

1. Apertura de la aplicación eclipse:



2. Crear un espacio de trabajo "workSpace" donde se va a cargar el proyecto PTw@ndA:

Para ello se accede a la opción de menú "File → Switch Workspace…", donde introducimos el nombre de una carpeta donde se va a cargar el proyecto PTw@ndA.

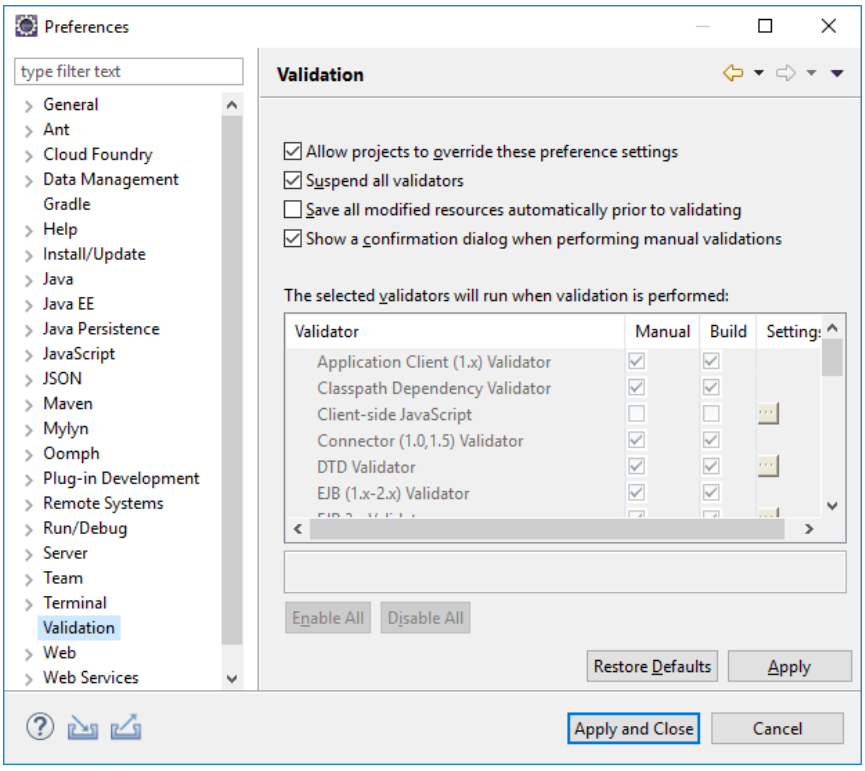


Una vez dado al botón “OK”, se cerrará el eclipse y volverá abrirse, con la nueva configuración de workspace.

3. Desactivación de las validaciones del Proyecto:

Esta opción aunque no sea requerida, ni afecta al comportamiento del proyecto Plataforma de Tramitación, se comenta que se pueden deshabilitar las opciones de validación (XML, HTML, WSDL, …) para aumentar la rapidez de compilación, despliegue, … del proyecto.

Para ello se accede a la opción de menú “Window → Preferences”

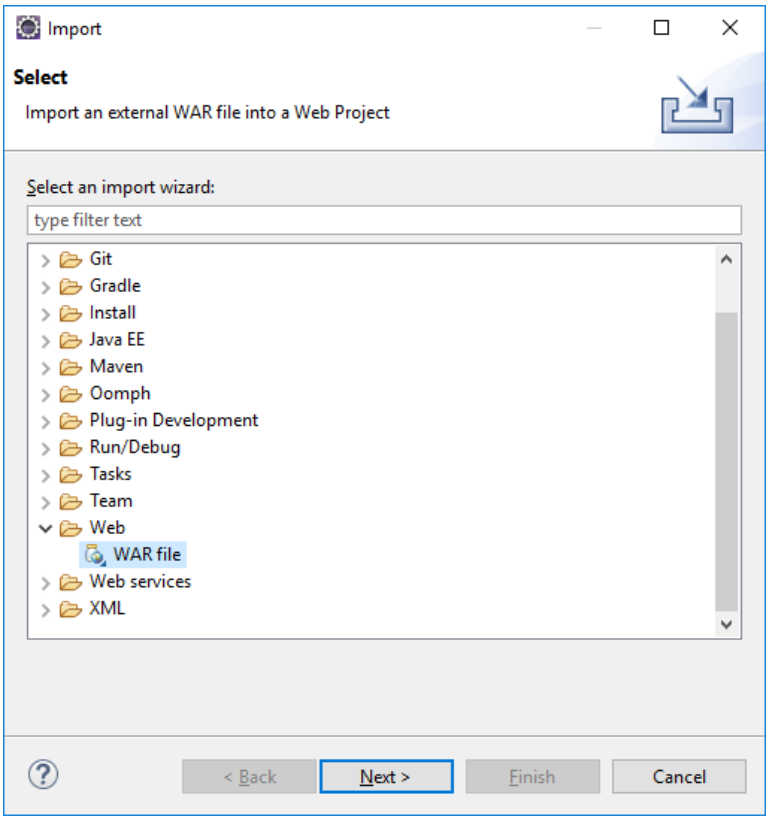


Sobre la opción “Validation” se pulsa sobre el check “Suspend all validators” para que se deshabiliten todas las opciones de validación.

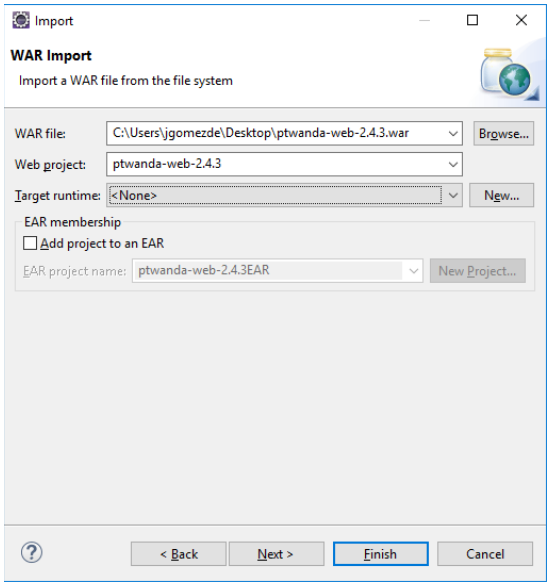
4. Importación del .war de Plataforma de Tramitación w@ndA:

Al abrirse de nuevo el eclipse, se cierra la pestaña de “Welcome”, y sobre la vista “Java” y/o “J2EE” que aparece, se pasa a importar el proyecto de Plataforma de Tramitación desde el fichero .war descargado.

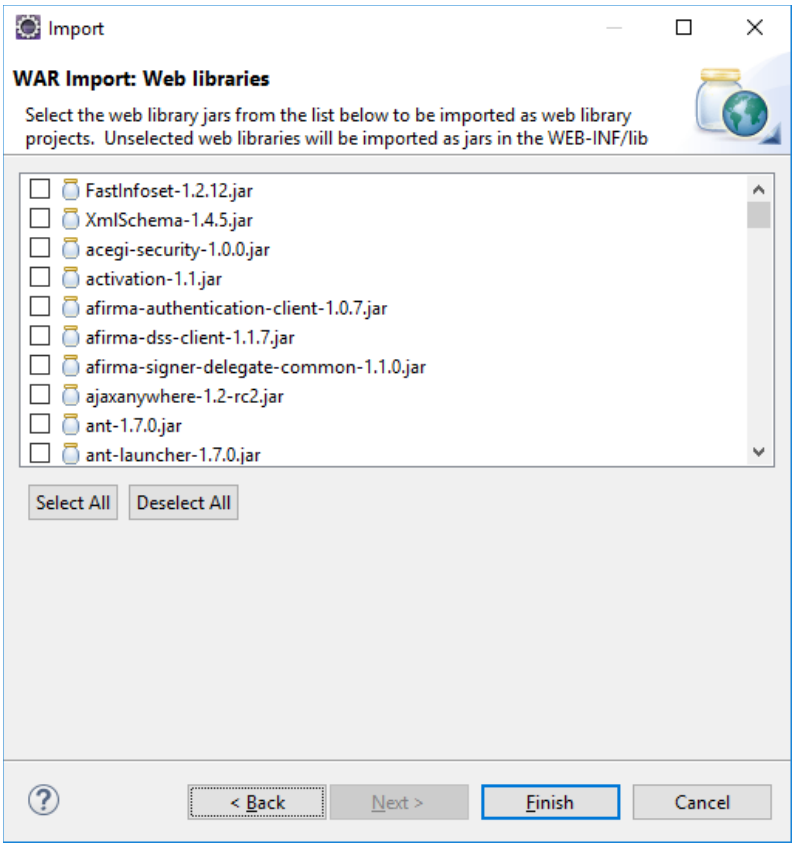
Para ello se accede a la opción de menú “File → Import…”



Se selecciona la opción “Web → WAR file” y pulsamos sobre botón “Next >”



Se selecciona el .war de Plataforma de Tramitación. Desde el botón “Browse…” se selecciona el .war que se tenga en el equipo copiado. Por defecto, eclipse denomina al “Web project” como ptwanda-web, igual al nombre del .war distribuido. Se pulsa sobre el botón “Next >”



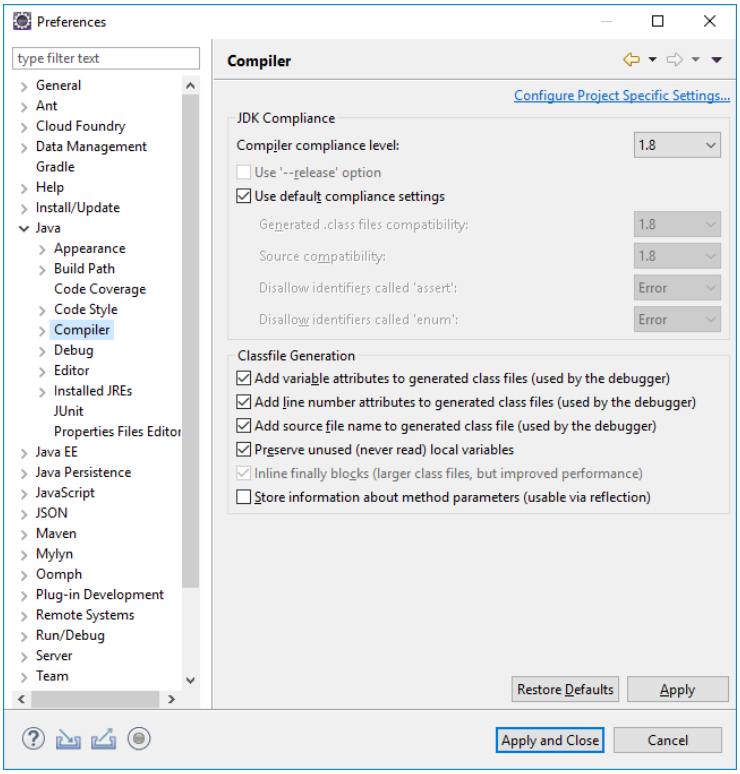
A continuación aparece la ventana con el conjunto de librerías que se van a importar. No hay que realizar ninguna acción en esta ventana. Se pulsa el botón “Finish”, y entonces se comienza a importar el proyecto de Plataforma de Tramitación.

Este proceso puede llevar varios minutos.

5. Corrección de errores al importar el Proyecto de Plataforma de Tramitación:

Una vez importado el proyecto pueden aparecer errores de compilación. Se debe utilizar para compilar un nivel 8.0.

Para solucionar este problema se accede a la opción de menú “Window ◊ Preferences”. Sobre la opción “Java - Java Compiler”, se elige para la opción “Compile compliance level” la 1.8 de la combo:



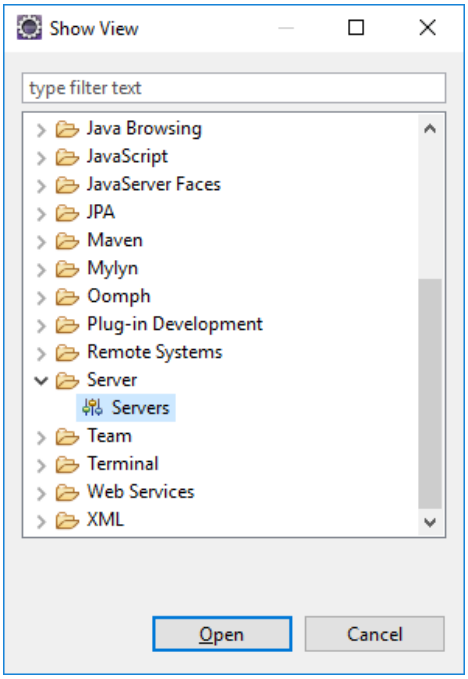
Se pulsa sobre el botón “OK”. Eclipse nos indicará que se volverá a construir el proyecto con esta opción de compilación, la cual se acepta.

5.3 Configuración de un servidor de aplicaciones

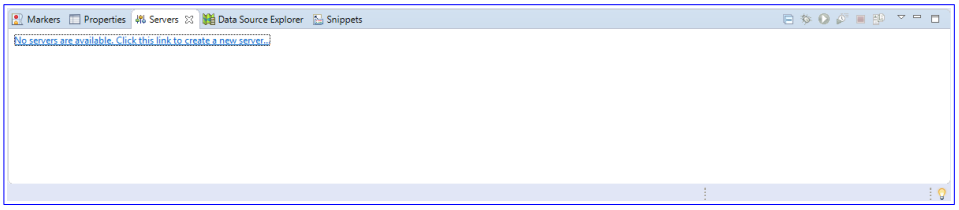
Una vez importado el proyecto de Plataforma de Tramitación, se pasa a configurar dentro de eclipse un servidor de aplicaciones en este caso, un servidor WildFly. Los pasos a seguir son los siguientes:

1. Abrir la vista de servidores de eclipse.

Para ello se accede a la opción de menú “Window → Show View → Other …”



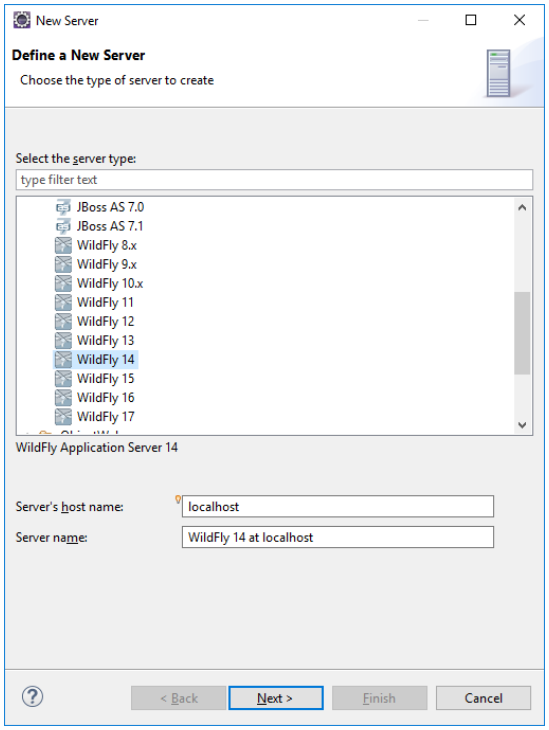
Se elige la opción “Servers” dentro de la carpeta “Server”. Se abrirá una nueva pestaña sobre el eclipse:



2. Crear un nuevo servidor para el proyecto, dentro de eclipse:

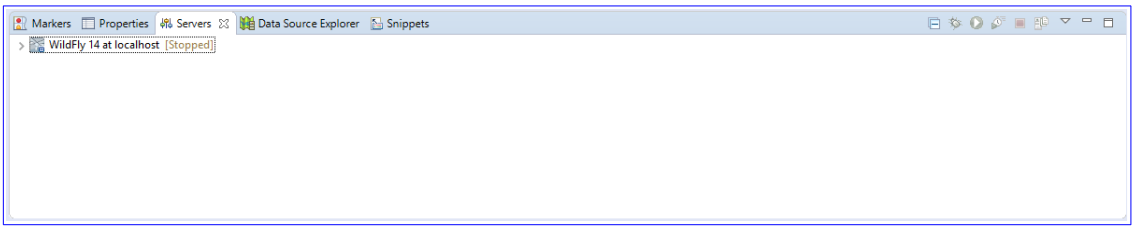
Sobre esa pestaña se accede mediante el botón secundario del ratón, y se selecciona la opción “New → Server”:

Se abrirá una ventana modal, sobre la que se selecciona el tipo de servidor, en nuestro caso JBoss Community → WildFly 15



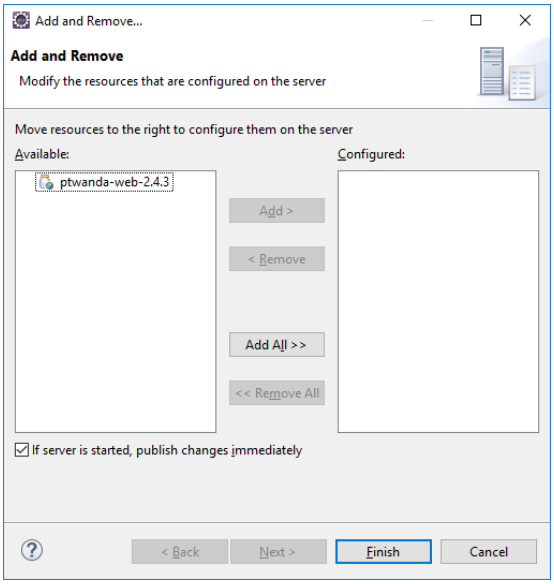
Se pulsa en el botón “Next > Next >”. Sobre la siguiente ventana que aparece se indica el nombre del servidor, al mismo tiempo que se selecciona la carpeta del servidor WildFly sobre la que se va a desplegar la aplicación, y la configuración sobre la que se desplegará (se indicará como fichero de configuración **standalone-full.xml**).

Se pulsa sobre el botón “Finish”. Se observará que se crea un servidor, así como en la parte del explorador los ficheros necesarios para configurar el mismo:



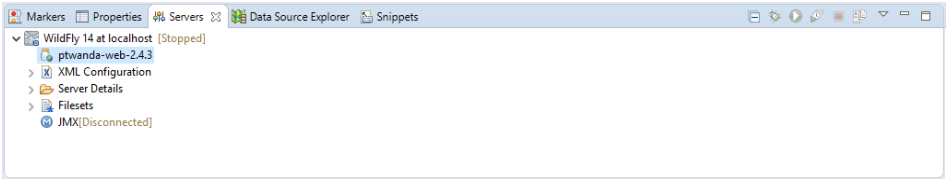
3. Añadir el proyecto “ptwanda-web” al servidor creado.

Sobre el servidor creado, se accede mediante el botón secundario del ratón, y se selecciona la opción “Add and Remove Projects”, abriéndose la siguiente ventana modal:



Se selecciona de la lista de “Available projects” el proyecto “ptwanda-web” y con el botón “Add >” lo añadimos a la lista de “Configured projects”. Se pulsa sobre el botón “Finish”.

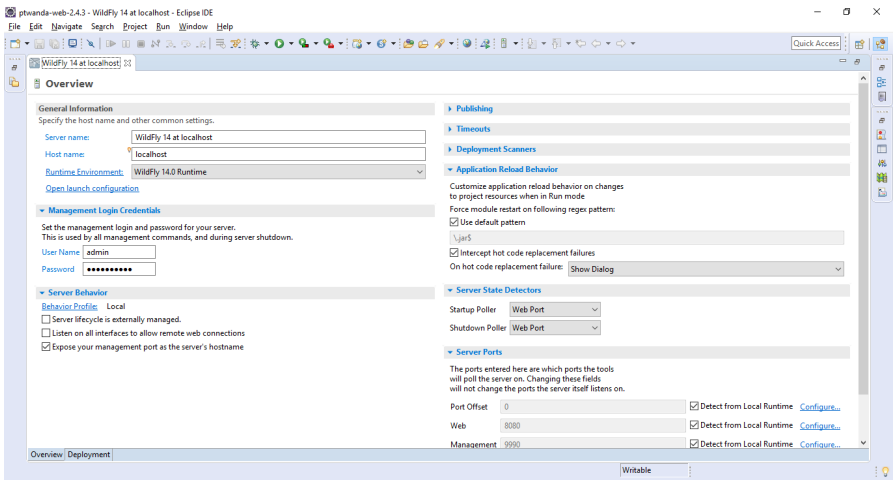
Se observará en la vista de servidores como se ha añadido el proyecto “ptwanda-web” al servidor:



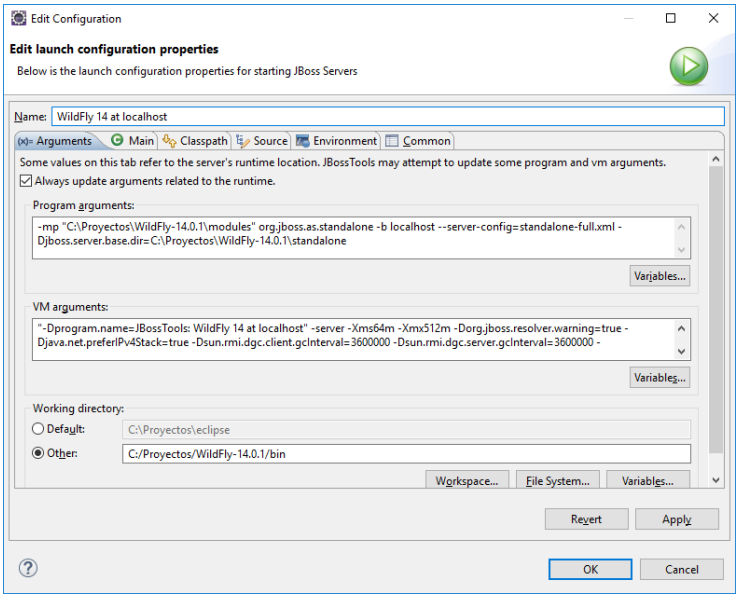
4. Configuración de Solr para la indexación.

Para el correcto funcionamiento de la indexación en la aplicación es necesario realizar la configuración del motor de indexación y búsqueda. Para ello hacemos referencia al manual de instalación, al punto **3.3.3.6 Configuración del motor de indexación y búsqueda.**

Es importante destacar que las propiedades que deben establecerse en el fichero `${WILDFLY_14.0.1_FINAL}/bin/standalone.conf`, hay que establecerlas desde el propio Eclipse. Para ello accedemos a las propiedades del servidor haciendo doble clic sobre el mismo:



En la sección General Information pinchamos sobre el enlace **Open launch configuration:**



En el cuadro de texto **VM arguments** habrá que establecer el valor de las siguientes propiedades:

-Dsolr.solr.home=$RUTA_BASE_SOLR

5.4 Arranque de Plataforma de Tramitación w@ndA

Una vez configurado el servidor de aplicaciones, el siguiente paso es iniciarlo para acceder a la plataforma de tramitación.

Los pasos para llevar a cabo el correcto inicio del proyecto son:

1. Publicación del proyecto.

Sobre la vista de servidores se pulsa sobre el icono  , o botón secundario del ratón sobre el servidor creado y se selecciona la opción “Publish”, para realizar la publicación.

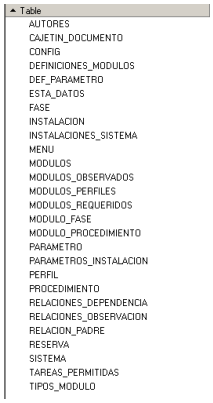
Este proceso puede tardar algunos minutos.

2. Arranque del servidor.

Una vez publicado y desplegado la aplicación “ptwanda-web” se pasa a arrancar el servidor.

Para ello se pulsa sobre el icono  , o botón secundario del ratón sobre el servidor creado y se selecciona la opción “Start”.

Al arrancar el servidor de aplicaciones se podrá observar que se crean las tablas correspondientes sobre el DataSource “jdbc/Plataforma, utilizando el mapeo de tablas de hibernate:



3. Corrección del error la tabla “CONFIG”

Al arrancar el servidor de aplicaciones por primera vez se observará que se produce el siguiente error:

```
ERROR DE CONFIGURACIÓN, REVISE EL VALOR DE LAS PROPIEDADES URL_ESCRITORIO Y URL_PFIRMA
java.lang.NullPointerException
        at es.juntadeandalucia.plataforma.util.Resources.setPropiedades(Resources.java:58)
        at es.juntadeandalucia.plataforma.modulos.ConfigServiceImpl.obtenerProperties(ConfigServiceImpl.java:65)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:585)
        at org.springframework.util.MethodInvoker.invoke(MethodInvoker.java:276)
        at org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean$MethodInvokingJob.executeInternal(MethodInvokingJobDetailFactoryBean.ja
        at org.springframework.scheduling.quartz.QuartzJobBean.execute(QuartzJobBean.java:86)
        at org.quartz.core.JobRunShell.run(JobRunShell.java:203)
        at org.quartz.simpl.SimpleThreadPool$WorkerThread.run(SimpleThreadPool.java:520)
```

Este error es debido a que la tabla “CONFIG” no contiene las propiedades correspondientes, para el correcto funcionamiento de la aplicación “plataforma-sdk”.

Para el solucionar este problema, es necesario ejecutar el script de BBDD “**05.Datos_PTW_TPOWNER.sql**” que se acompaña con la distribución del producto.

Una vez ejecutado el script de BBDD, se vuelve a arrancar el servidor.

Esta vez se ha de arrancar correctamente; sobre la vista “Console” debe aparecer al arrancar el siguiente mensaje:

```
[2009-01-19 11:14:02,843] trewa.util.Log  INFO:  (trewa.bd.trapi.trapiui.tpo.dao.TrPerfilUsuarioDAO - obtenerPerfilUsuario(TpoPK, ClausulaWhere, ClausulaOrd
[2009-01-19 11:14:02,843] trewa.util.Log  INFO:  (trewa.bd.trapi.trapiui.tpo.dao.TrPerfilUsuarioDAO - obtenerPerfilUsuario(TpoPK, ClausulaWhere, ClausulaOrd
[2009-01-19 11:14:03,609] trewa.util.Log  INFO:  (trewa.bd.trapi.trapiui.tpo.dao.TrPerfilUsuarioDAO - obtenerPerfilUsuario(TpoPK, ClausulaWhere, ClausulaOrd
[2009-01-19 11:14:03,609] trewa.util.Log  INFO:  (trewa.bd.trapi.trapiui.tpo.dao.TrPerfilUsuarioDAO - obtenerPerfilUsuario(TpoPK, ClausulaWhere, ClausulaOrd
[2009-01-19 11:14:03,968] es.juntadeandalucia.plataforma.modulos.ModulosInstalacionPredefinidosServiceImpl  INFO: Sincronización con tramitador realizada.
```

5.5 Acceso a la Aplicación

Una vez arrancado el servidor de aplicaciones, se procede al acceso a la aplicación.

Para ello se abre un navegador, y según los la configuración que hayamos realizado sobre el servidor, en nuestro caso se accede mediante la Url:

http://localhost:8080/ptwanda-web

6 GUÍA DE DESARROLLO DE MÓDULOS FUNCIONALES

6.1 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 w@ndA.

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 módulos funcionales.

Estas pautas son de obligado cumplimiento para lograr la construcción de un módulo compatible con PTw@ndA 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 PTw@ndA.

6.2 Tipos de módulos funcionales

Por su naturaleza funcional, existen dos tipos de componentes funcionales o módulos en la Plataforma de Tramitación w@ndA, 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 siguiente imagen se tiene un ejemplo de escritorio, en el podemos ver los distintos módulos genéricos que existen, como son: información del usuario, datos del expediente, datos de la fase, transiciones posibles… etc.

Debido a su forma de visualización y objeto, existen 8 tipos de *módulos* para la Plataforma de Tramitación:

- **Portlet**: Módulos cuya funcionalidad es accesible a través de una sección del escritorio de tramitación. *Ej: Expedientes relacionados, Árbol de evolución, …*
- **Externo**: Son módulos cuya funcionalidad es accesible desde cualquier menú definido en la aplicación. *Ej: Alta de expedientes de Recursos Administrativos, Consulta de comunidades andaluzas inscritas, …*
- **Utilidades**. Módulos invocados desde el propio escritorio de tramitación que dan soporte y ayuda a la tramitación de los expedientes. *Ej: Datos del expediente, Traslado de expedientes, …*
- **Procedimiento**: Desarrollo de las tareas, acciones, condiciones y variables que forman parte de la implementación del modelado de un procedimiento concreto.
- **Administración:** Módulos cuya funcionalidad es accesible desde la herramienta de administración, con objeto de realizar tareas de administración, parametrización o mantenimiento.
- **Consulta**: Módulos cuya funcionalidad es añadir acciones que sean de interés para los trámites directamente sobre el listado de expedientes que se muestran tras la realización de una búsqueda de expedientes.
- **Web Service:** Son módulos cuya funcionalidad es incluir el servicio web implementado en el archivo de descripción de web service que define el catálogo de servicios de la Plataforma de Tramitación.
- **Utilidades masivas:** Módulos invocados desde el módulo de búsqueda de expedientes cuando se accede al detalle de un conjunto de expedientes.

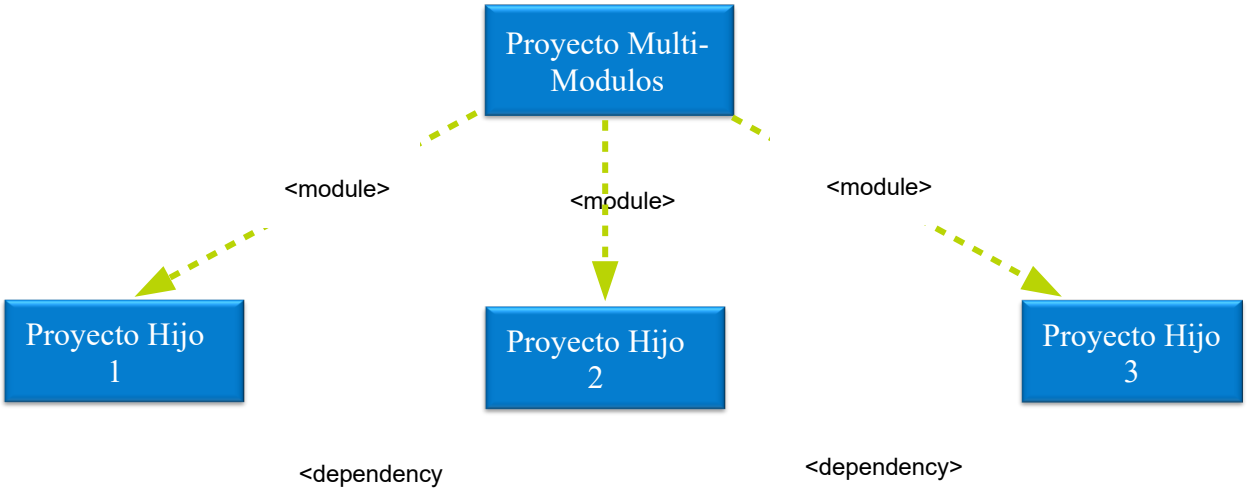
6.3 Desarrollo del módulo funcional

Para desarrollar el nuevo módulo funcional en PTw@ndA será necesario desarrollar y configurar una serie de componentes que formarán el desarrollo completo del módulo.

Apache Maven proporciona una estructura multi-modulos para el desarrollo que es muy recomendable utilizar cuando se vayan a implementar varios módulos.

Un proyecto multi-modulos es un tipo particular de proyecto, no produce ningún artefacto final y está compuesto de otros proyectos conocidos con el nombre **módulos**. Cuando se ejecuta un comando en el proyecto, lo ejecutará en todos los proyectos hijos. Maven es capaz, a través de su componente de reactor, descubrir el orden correcto de ejecución y detectar dependencias circulares. De esta forma obtenemos un fichero padre donde podremos declarar dependencias que serán comunes a todos los hijos, como por ejemplo, el core de plataforma.

La estructura de nuestro proyecto deberá ser la siguiente:



Los módulos hijos podrán tener dependencias entre ellos, siempre y cuando no se cree una dependencia circular.

El primer paso será crear nuestro proyecto multi-modulos. En este punto puede optarse por crear un proyecto maven desde linea de comandos o crear “a mano” un pom.xml que identificará a nuestro proyecto multi-modulos. Como comentamos anteriormente, el proyecto multi-modulos no generará ningún artefacto, por lo cuál no tendrá una estructura de proyecto como tal, simplemente quedará instanciado con la creación del pom.xml.

En el directorio donde hemos creado nuestro workspace con Eclipse, creamos un fichero pom.xml con el siguiente contenido:

```
<project    xmlns="http://maven.apache.org/POM/4.0.0"    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>
<groupId>es.juntadeandalucia.ptwanda</groupId>
<artifactId>modulosSubAdm</artifactId>
<packaging>pom</packaging>

<version>1.0</version>
<name>nombre_padre</name>
<url>http://maven.apache.org</url>

<build>
    <plugins>
       <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <configuration>
             <encoding>ISO-8859-1</encoding>
             <source>1.6</source>
             <target>1.6</target>
             <showDeprecation>false</showDeprecation>
             <showWarnings>false</showWarnings>
          </configuration>
       </plugin>
    </plugins>

</build></project>
```

Para la creación de un proyecto multi-modulos es necesario indicar tipo pom en el packaging.

Especificar el nivel de compilación para todos los proyectos hijos.

A continuación, creamos los distintos proyectos hijos. En este punto se crearán tantos proyectos hijos como módulos funcionales queramos desarrollar.

Los proyectos se crearán desde línea de comandos, lanzando el comando desde el directorio donde se encuentre el pom.xml del proyecto multi-modular ó bien desde la ruta donde se encuentre nuestro workspace si no se hace uso de dicha estructura.

Al crear el proyecto hijo, Maven actualizará el pom.xml del proyecto multi-modular, insertando la referencia al módulo hijo creado.

El comando a lanzar es:

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes
-DgroupId=chap.scae.ptwanda -DartifactId=SubvencionesAdm -Dversion=1.0
```

Los parámetros groupId, artifactId y version podrán tener el valor que se deseen. En este caso, se ha optado por los que se visualizan.

Al lanzar el comando, Maven descargará los plugins necesarios en el repositorio local. Si no encuentra estos plugins en el repositorio de CHAP, puede configurarse en el settings.xml de Apache Maven que se conecte al repositorio central de Maven. Para ello añadimos en la etiqueta <mirror> lo siguiente:

```
<mirror>
  <id>archiva.default</id>
  <url>https://ws024.juntadeandalucia.es/maven/repository/internal</url>
  <name>Repositorio Maven SCAE/CJAP</name>
  <mirrorOf>*</mirrorOf>
</mirror> -->
<mirror>
  <id>central</id>
  <url>http://repo1.maven.org/maven2</url>
  <name>Maven Repository Switchboard</name>
  <mirrorOf>*</mirrorOf>
</mirror>
```

Cuando ejecutemos el comando nos preguntará que opción de creación queremos utilizar. Seleccionamos '15' y pulsamos Intro.

```
C:\DesarrolloModulos\workspace>mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=cjap.scae.ptwanda -DartifactId=SubvencionesADM -Dversion=1.0
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] ------------------------------------------------------------------------
[INFO] Building Plataforma de tramitaci?n
[INFO]    task-segment: [archetype:generate] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] Preparing archetype:generate
[INFO] No goals needed for project - skipping
[INFO] Setting property: classpath.resource.loader.class => 'org.codehaus.plexus.velocity.ContextClassLoaderResourceLoader'.
[INFO] Setting property: velocimacro.messages.on => 'false'.
[INFO] Setting property: resource.loader => 'classpath'.
[INFO] Setting property: resource.manager.logwhenFound => 'false'.
[INFO] [archetype:generate {execution: default-cli}]
[INFO] Generating project in Interactive mode
[INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.maven.archetypes:maven-archetype-quickstart:1.0)
Choose archetype:
1: internal -> appfuse-basic-jsf (AppFuse archetype for creating a web application with Hibernate, Spring and JSF)
2: internal -> appfuse-basic-spring (AppFuse archetype for creating a web application with Hibernate, Spring and Spring MUC)
3: internal -> appfuse-basic-struts (AppFuse archetype for creating a web application with Hibernate, Spring and Struts 2)
4: internal -> appfuse-basic-tapestry (AppFuse archetype for creating a web application with Hibernate, Spring and Tapestry 4)
5: internal -> appfuse-core (AppFuse archetype for creating a jar application with Hibernate and Spring and XFire)
6: internal -> appfuse-modular-jsf (AppFuse archetype for creating a modular application with Hibernate, Spring and JSF)
7: internal -> appfuse-modular-spring (AppFuse archetype for creating a modular application with Hibernate, Spring and Spring MUC)
8: internal -> appfuse-modular-struts (AppFuse archetype for creating a modular application with Hibernate, Spring and Struts 2)
9: internal -> appfuse-modular-tapestry (AppFuse archetype for creating a modular application with Hibernate, Spring and Tapestry 4)
10: internal -> maven-archetype-j2ee-simple (A simple J2EE Java application)
11: internal -> maven-archetype-marmalade-mojo (A Maven plugin development project using marmalade)
12: internal -> maven-archetype-mojo (A Maven Java plugin development project)
13: internal -> maven-archetype-portlet (A simple portlet application)
14: internal -> maven-archetype-profiles ()
15: internal -> maven-archetype-quickstart ()
16: internal -> maven-archetype-site-simple (A simple site generation project)
17: internal -> maven-archetype-site (A more complex site project)
18: internal -> maven-archetype-webapp (A simple Java web application)
19: internal -> jini-service-archetype (Archetype for Jini service project creation)
20: internal -> softeu-archetype-seam (JSF+Facelets+Seam Archetype)
21: internal -> softeu-archetype-seam-simple (JSF+Facelets+Seam (no persistence) Archetype)
22: internal -> softeu-archetype-jsf (JSF+Facelets Archetype)
23: internal -> jpa-maven-archetype (JPA application)
24: internal -> spring-osgi-bundle-archetype (Spring-OSGi archetype)
25: internal -> confluence-plugin-archetype (Atlassian Confluence plugin archetype)
26: internal -> jira-plugin-archetype (Atlassian JIRA plugin archetype)
27: internal -> maven-archetype-har (Hibernate Archive)
28: internal -> maven-archetype-sar (JBoss Service Archive)
29: internal -> wicket-archetype-quickstart (A simple Apache Wicket project)
30: internal -> scala-archetype-simple (A simple scala project)
31: internal -> lift-archetype-blank (A blank/empty liftweb project)
32: internal -> lift-archetype-basic (The basic (liftweb) project)
33: internal -> cocoon-22-archetype-block-plain ([http://cocoon.apache.org/2.2/maven-plugins/])
34: internal -> cocoon-22-archetype-block ([http://cocoon.apache.org/2.2/maven-plugins/])
35: internal -> cocoon-22-archetype-webapp ([http://cocoon.apache.org/2.2/maven-plugins/])
36: internal -> myfaces-archetype-helloworld (A simple archetype using MyFaces)
37: internal -> myfaces-archetype-helloworld-facelets (A simple archetype using MyFaces and facelets)
38: internal -> myfaces-archetype-trinidad (A simple archetype using Myfaces and Trinidad)
39: internal -> myfaces-archetype-jsfcomponents (A simple archetype for create custom JSF components using MyFaces)
40: internal -> gmaven-archetype-basic (Groovy basic archetype)
41: internal -> gmaven-archetype-mojo (Groovy mojo archetype)
Choose a number:  (1/2/3/4/5/6/7/8/9/10/11/12/13/14/15/16/17/18/19/20/21/22/23/24/25/26/27/28/29/30/31/32/33/34/35/36/37/38/39/40/41) 15: :
```

Nos pedirá confirmación. Escribimos 'Y', pulsamos Intro y el proyecto se habrá creado correctamente.

```
[INFO] artifact org.apache.maven.archetypes:maven-archetype-quickstart: checking for updates from central
Confirm properties configuration:
groupId: cjap.scae.ptwanda
artifactId: SubvencionesADM
version: 1.0
package: cjap.scae.ptwanda
 Y: : Y
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating OldArchetype: maven-archetype-quickstart:RELEASE
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: cjap.scae.ptwanda
[INFO] Parameter: packageName, Value: cjap.scae.ptwanda
[INFO] Parameter: package, Value: cjap.scae.ptwanda
[INFO] Parameter: artifactId, Value: SubvencionesADM
[INFO] Parameter: basedir, Value: C:\DesarrolloModulos\workspace
[INFO] Parameter: version, Value: 1.0
[INFO] ********************* End of debug info from resources from generated POM **********************
[INFO] OldArchetype created in dir: C:\DesarrolloModulos\workspace\SubvencionesADM
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 minutes 44 seconds
[INFO] Finished at: Thu May 27 18:50:48 CEST 2010
[INFO] Final Memory: 10M/18M
[INFO] ------------------------------------------------------------------------
C:\DesarrolloModulos\workspace>
```

En el caso de la estructura multi-modular se habrá modificado el pom.xml incorporando el nuevo módulo creado:

```
<project    xmlns="http://maven.apache.org/POM/4.0.0"    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>es.juntadeandalucia.ptwanda</groupId>
  <artifactId>modulosSubAdm</artifactId>
  <packaging>pom</packaging>
  <version>1.0</version>
  <name>nombre_padre</name>
  <url>http://maven.apache.org</url>

  <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
               <encoding>ISO-8859-1</encoding>
               <source>1.6</source>
               <target>1.6</target>
               <showDeprecation>false</showDeprecation>
               <showWarnings>false</showWarnings>
            </configuration>
         </plugin>
      </plugins>
  </build>

  <modules>
    <module>SubvencionesADM</module>
  </modules>
</project>
```

Nuevo módulo creado

El módulo creado tiene una estructura por defecto, que es la que implementa el plugin utilizado en la generación del proyecto. Esta estructura habrá que adaptarla para que coincida con la estructura general de un módulo vertical para la plataforma, ya que a la hora de generar el empaquetado zip final, Maven localiza los ficheros a incluir en el zip basándose en esta estructura definida.

Todo módulo debe estar formado por los siguientes componentes:

- Ficheros de configuración
- Clases Java
- Recursos JSP
- Hojas de estilo e imágenes
- Fichero pom.xml
- Fichero modulo-vertical-pt.xml
- Fichero despliegue.xml

A continuación se especificarán las normas y consejos a seguir para cada uno de los componentes citados:

6.3.1 Ficheros de configuración

Para implementar un nuevo módulo funcional, es necesario introducir ficheros de configuración que permitan adjuntar el módulo a PTw@ndA, y en los que se definan los recursos incorporados y su comportamiento.

Concretamente, es necesario definir por cada módulo dos ficheros correspondientes a la configuración de la navegación y presentación del módulo (Struts 2), y otro con la configuración de los componentes de negocio y acceso a datos (Spring).

A continuación se indica la estructura y características de cada uno de estos dos ficheros:

- Fichero de configuración Struts:

  Este fichero incluye las nuevas definiciones de acciones (actions) que se invocarán desde la capa de presentación (páginas JSP), y las relaciones y redirecciones entre ellas.

  Es necesario seguir las siguientes pautas para el correcto desarrollo y despliegue del módulo:

  - La nomenclatura debe ser struts-<nombre-modulo>.xml
  - El fichero de configuración para el alta debe tener el namespace: *modulos/<nombre-modulo>*
  - El fichero de configuración para las tareas debe tener el namespace: *agenda/tareas*
  - No puede declararse una acción (action) cuyo nombre ya exista en PTw@ndA.

  A continuación se muestra un ejemplo de fichero de configuración de Struts:

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

<struts>

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

               <action name="accion1" method="metodoAccion1"
                      class="claseAction">
                      <result name="success">/modulos/moduloNuevo/bbb.jsp</result>
                      <result name="otraAccion">/modulos/moduloNuevo/ccc.jsp</result>
               </action>

               ...
```

```
</struts>
```

- Fichero de configuración Spring:

Este fichero incluye las definiciones de beans de negocio y de acceso a datos, y las referencias o inyecciones de otros componentes ya existentes en Ptw@ndA.

Es necesario seguir las siguientes pautas para el correcto desarrollo y despliegue del módulo:

- Solo existirá un único fichero de configuración por módulo.
- No pueden utilizarse como id de beans incorporados los ya existentes en PTw@ndA. (Dentro de la ruta /WEB-INF/clases/ se encuentran los ficheros de configuración de Spring donde podrán consultarse los ids de los beans ya existentes en la plataforma).

A continuación se muestra un ejemplo de fichero de configuración de Spring. Es importante destacar que la cabecera de los ficheros de configuración debe ser siempre la indicada en el ejemplo siguiente:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="
                     http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                     http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

        <bean      id="Servicio"      class="es.juntadeandalucia.plataforma.aaa.ServicioNuevo"
        scope="prototype" parent="PTWandaService">
               <property name="servicioAInyectar">
                     <ref bean="referenciaServicioAInyectar"/>
               </property>
        </bean>
        ...
```

6.3.2 Clases JAVA

Dependiendo del tipo de módulo que se quiera desarrollar y de su objetivo, se requerirá el desarrollo de clases Java que implementen la funcionalidad deseada, implementen variables, nuevos servicios, etc…

Es recomendable que se siga como convección para la nomenclatura de paquetes la ruta

es.juntadeandalucia.plataforma.modulo. <NOMBRE_MODULO>

Dependiendo del tipo de módulo es posible que se incorporen los siguientes tipos de clases Java:

- **Portlet**: Clases Action, Servicios, Clases DAO, Clases DTO.
- **Externo**: Clases Action, Servicios, Clases DAO, Clases DTO.
- **Utilidades:** Clases Action, Servicios, Clases DAO, Clases DTO.

- **Procedimiento**: Clases Condiciones, Clases Acciones, Clases Variables, Clases Action, Servicios, Clases DAO, Clases DTO.

A partir de la ruta anteriormente especificada, y del objeto de la clase Java, se proponen los siguientes paquetes:

PAQUETE	DESCRIPCIÓN
AltaExpediente	ficheros relacionados con el proceso de alta de un expediente
Tareas	ficheros relacionados con las tareas de un expediente
Utilidades	ficheros relacionados con las utilidades
Condiciones	ficheros relacionados con las condiciones
Acciones	ficheros relacionados con las acciones
Variables	ficheros relacionados con las variables
XmlExpediente	ficheros relacionados con la estructura XML de otros datos de expedientes
Dao	ficheros relacionados con el acceso a BBDD
Dto	tipos propios de datos
Config	ficheros de configuración
Properties	ficheros properties

6.3.3 Servicios

PTw@ndA ofrece una amplia capa de servicios de negocio que permiten interaccionar con el motor de tramitación Trew@, gestionar la configuración y visualización de módulos, noticias, notas del expediente, …

En el caso que la implementación del módulo funcional requiera la creación de un nuevo servicio, es necesario que se especifique la herencia del servicio base de PTw@ndA: `PTWandaServiceImpl`. Para ello es necesario que se especifique en la definición de la clase Java:

```
/**
 * <p>
 * Servicio nuevo destinado a ...
 * </p>
 *
 * @author everis
 * @version 1.0
 */
```

```
public class ServicioNuevoImpl extends PTWandaServiceImpl implements IServicioNuevo {
```

A la hora de definir este servicio dentro del fichero de configuración `applicationContext.xml`, habrá de definirse expresamente el padre del nuevo servicio:

```
<bean id="servicioNuevo" class="es.juntadeandalucia.plataforma.ServicioNuevoImpl"
parent="PTWandaService">

...
</bean>
```

Si el Servicio desarrollado requiere interaccionar con el motor de tramitación Trew@, es conveniente que la herencia se realice sobre el Servicio `ConfiguracionTramitacionServiceImpl`, en el que se definen métodos que permiten interacturar y gestionar la utilización de la API de acceso a Trew@.

6.3.4 Recursos JSP

Los ficheros JSP serán las páginas web que se mostrarán en el módulo, y sobre las que interactuará el usuario. A partir de aquí se ubicarán en los siguientes bloques:

BLOQUE	DESCRIPCIÓN
altaExpediente	jsp relacionadas con el alta de un expediente
tareas	jsp relacionadas con las tareas de un expediente
utilidades	jsp relacionadas con las utilidades
decorators	jsp para la decoración

6.3.5 Hojas de estilo e imágenes

Puede ser posible que el desarrollo del módulo funcional requiera la incorporación de hojas de estilo y/o imágenes.

Las hojas de estilo e imágenes se ubican al mismo nivel que los recursos web.

6.3.6 Fichero pom.xml

El fichero Project Object Model (pom.xml) describe el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos.

Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la **compilación del código** y su **empaquetado** en diferentes formatos.

El contenido del fichero pom.xml se divide en tres bloques principales:

1. **Identificación del producto** generado por el código fuente que acompaña al documento pom.xml. Se hace mediante cuatro parámetros.
    a. **groupId**: Identificador del proyecto o módulo funcional. Se identifica con la estructura jerárquica de directorios que supone la localización de la librería en el repositorio que la aloje. Idealmente, se corresponde con la estructura de carpetas interna que tiene la librería, aunque no es una práctica obligatoria.
    b. **artifactId**: Identificador del módulo vertical.
    c. **name**: Texto con la descripción del módulo vertical.
    d. **version**: versión del producto.
2. **Dependencias:** Librerías o recursos externos requeridos para la generación del módulo funcional.
3. **Repositorios** donde se encuentran alojadas las citadas librerías. Deberá declararse el repositorio perteneciente al Servicio de Coordinación de Administración Electrónica: https://ws074.juntadeandalucia.es/archiva

La estructura del fichero pom.xml debe ser la siguiente:

```
<project     xmlns="http://maven.apache.org/POM/4.0.0"     mlns:xsi=http://www.w3.org/2001/XMLSchema-instance       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>modulosSubAdm</artifactId>
    <groupId>es.juntadeandalucia.ptwanda</groupId>
    <version>1.0</version>
  </parent>
  <groupId>es.juntadeandalucia.ptwanda</groupId>
  <artifactId>SubvencionesADM</artifactId>
  <packaging>jar</packaging>
  <version>${project.parent.version}</version>
  <name>SubvencionesADM</name>
  <url>http://maven.apache.org</url>

  <dependencies>
     <dependency>
        <groupId>es.juntadeandalucia.ptwanda</groupId>
        <artifactId>ptwanda-core</artifactId>
        <version>2.4.3</version>
        <scope>provided</scope>
     </dependency>
  </dependencies>

  <build>
    <plugins>

      <!-- NO SE INCLUYEN EN EL JAR NI LOS RECURSOS WEB NI LOS XML DE CONFIGURACIÓN -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <configuration>
          <excludes>
            <exclude>webapp/**</exclude>
            <exclude>despliegue.xml</exclude>
```

El pom.xml del proyecto hijo tiene una

referencia al proyecto multi-modulos.

Declarar una dependencia a otro proyecto hijo. Resaltar la importancia de declarar esta dependencia de tipo PROVIDED. De esta forma se indica a Maven que utilice el módulo hermano SÓLO para la compilación y no lo incluya en el zip de este módulo.

```
            <exclude>conf/**</exclude>
            <exclude>lib/**</exclude>
         </excludes>
       </configuration>
     </plugin>

     <!-- CON ESTE PLUGIN SE CONSTRUYE EL ZIP DEL MODULO -->
     <plugin>
        <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
           <descriptors>
              <descriptor>modulo-vertical-pt.xml</descriptor>
           </descriptors>
        </configuration>
        <executions>  <!-- FUERZO EL ASSEMBLY A LA FASE PACKAGE DEL PROYECTO -->
           <execution>
              <id>make-assembly</id>
              <phase>package</phase>
              <goals>
                 <goal>single</goal>
              </goals>
           </execution>
        </executions>
     </plugin>

     <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
           <encoding>ISO-8859-1</encoding>
           <source>1.6</source>
           <target>1.6</target>
           <showDeprecation>false</showDeprecation>
           <showWarnings>false</showWarnings>
        </configuration>
     </plugin>

   </plugins>
 </build>

</project>
```

Para el desarrollo de todo módulo vertical de la plataforma es necesario especificar la depedencia al core de plataforma residente en el repositorio de CHAP. En el pom.xml de ejemplo se observa cómo declarar esta dependencia.

El plugin **maven-jar-plugin** es utilizado para generar el jar con las clases compiladas y los archivos de properties, que será añadido en el directorio /lib del empaquetado zip final. Para generarlo se excluyen los directorios conf, lib, webapp y el fichero despliegue.xml.

Para la construcción del zip del módulo se utiliza el plugin **maven-assembly-plugin**, indicándole la localización del fichero de configuración *modulo-vertical-pt.xml* y forzando el assembly en la etapa “package”. Una vez ejecutado desde la línea de comando la instrucción **mvn package**, se generará el zip del módulo.

A la hora de especificar dependencias hay que tener especial cuidado que las librerías indicadas no sean incompatibles con las librerías propias de la plataforma de tramitación. Para ello puede observarse el pom.xml de la plataforma de tramitación para saber las dependencias que tiene.

6.3.7 Fichero modulo-vertical-pt.xml

La configuración para la realización del assembly viene definida en el fichero modulo-vertical-pt.xml. Este fichero permanece invariable para cualquier módulo vertical, independientemente del tipo de módulo a generar. El contenido del fichero se muestra a continuación:

```
<assembly>
   <id>modulo-vertical-pt</id>
   <formats>
      <format>zip</format>
   </formats>
   <includeBaseDirectory>false</includeBaseDirectory>

  <dependencySets>
        <dependencySet>
           <outputDirectory>/lib</outputDirectory>
           <includes>
              <include>*:jar</include>
           </includes>
        </dependencySet>
   </dependencySets>


   <fileSets>
      <fileSet>
         <directory>${project.basedir}/src/main/resources</directory>
         <outputDirectory>/</outputDirectory>
         <includes>
            <include>webapp/**</include>
            <include>despliegue.xml</include>
            <include>conf/**</include>
            <include>lib/**</include>
         </includes>
      </fileSet>
      <fileSet>
         <directory>${project.basedir}/src/main/webapp</directory>
         <outputDirectory>webapp</outputDirectory>
         <includes>
            <include>**</include>
         </includes>
      </fileSet>
   </fileSets>

</assembly>
```

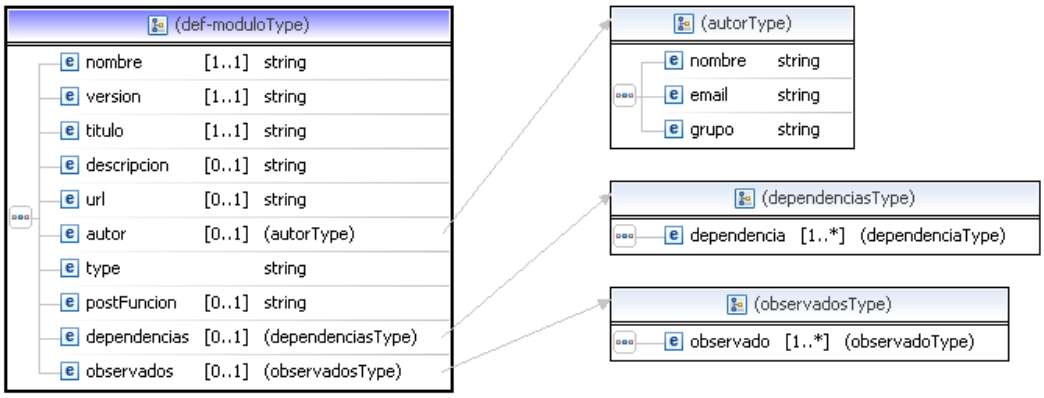
La etiqueta fileSet especifíca el contenido a incluir en el zip del módulo funcional.

6.3.8 Fichero despliegue.xml

Es obligatorio definir un fichero denominado `despliegue.xml`, en el cual se detallan aspectos descriptivos e informativos del módulo, actuando como fichero descriptor del módulo funcional. Este fichero se estructurará con los siguientes elementos:

El nombre del módulo especificado en la etiqueta <nombre> del archivo despliegue.xml será utilizado para desplegar los correspondientes archivos dentro del war de la plataforma. Por ejemplo, si el nombre de nuestro módulo es subadm, los recursos web se ubicarán en /modulos/subadm.

Resaltar la importancia de ausencia de espacios en blancos y otros caracteres especiales en la composición del nombre del módulo, ya que será utilizado para la construcción de rutas.



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.
- **descripcion**: texto descriptivo del módulo.
- **url**: dirección correspondiente a la página inicial del módulo.
- **icono-on:** ruta del icono que se visualiza al posicionar el cursor sobre el acceso al módulo, en el caso de tratarse de un módulo de tipo utilidad.
- **icono-off:** ruta del icono que se visualiza, en el caso de tratarse de un módulo de tipo utilidad.
- **icono-menu-principal:** ruta del icono que se visualiza en el contenedor, en el caso de tratarse de un módulo de tipo externo.
- **autor**: datos del autor del módulo.
- **type**: tecnología con la que se ha desarrollado el módulo (puede tomar los valores STRUTS-2 o NONE).

- **tipoInstalacion**: forma de visualización del módulo (puede tomar los valores PORTLET, EXTERNO, UTILIDADES, NONE, WS, ADMINISTRACIÓN, CONSULTA, UTILIDADMASIVA).
- **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="icono-menu-principal" 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-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:enumeration value="WS"/>
                      <xs:enumeration value="ADMINISTRACION"/>
                     <xs:enumeration value="CONSULTA"/>
                      <xs:enumeration value="UTILIDADMASIVA"/>
                   </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>
```

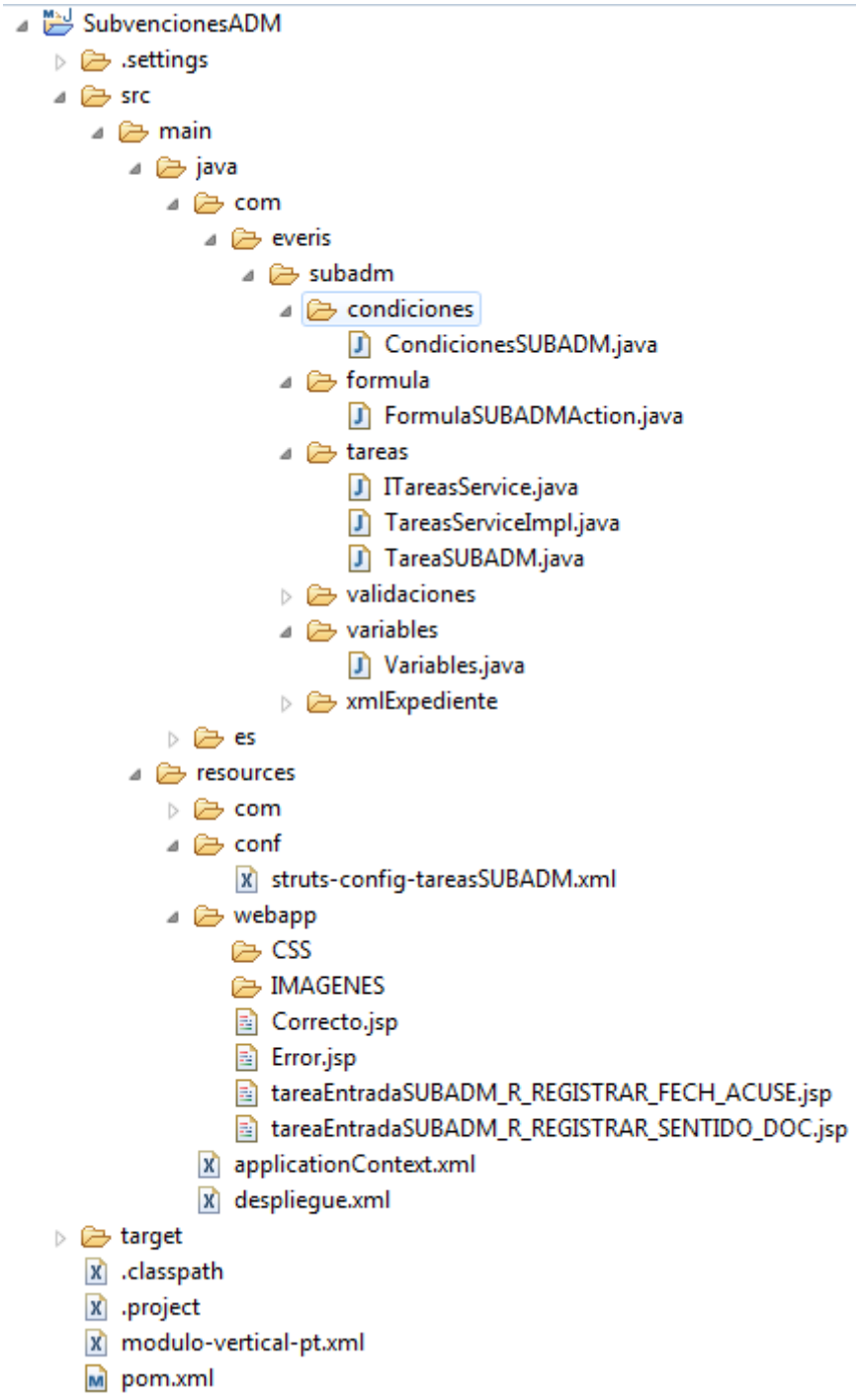
```
                              <xs:element name="modulo" minOccurs="1" maxOccurs="1">
                                 <xs:complexType>
                                    <xs:sequence>
                                       <xs:element name="nombre" type="xs:string" />
                                    </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 "*SubvencionesADM*":

```
<?xml version="1.0" encoding="UTF-8" ?>
<def-modulo>
   <nombre>subadm</nombre>
   <version>1.0</version>
   <titulo>Subvenciones Administrativas</titulo>
   <descripcion>Módulo reutilizable de subvenciones administrativas</descripcion>
   <autor>
      <nombre>juntadeandalucia</nombre>
      <email>desarrollo@juntadeandalucia.es</email>
      <grupo>desarrollo</grupo>
   </autor>
   <type>STRUTS-2</type>
   <tipoInstalacion>NONE</tipoInstalacion>
   <postFuncion></postFuncion>
   <dependencias>
      <dependencia>
         <modulo>
            <nombre>comun</nombre>
         </modulo>
         <version>1.0</version>
     </dependencia>
   </dependencias>
   <observados>
      <observado>
         <nombre>capa_tareas_asoc</nombre>
      </observado>
      <observado>
         <nombre>capa_tramita</nombre>
      </observado>
   </observados>
```

```
</def-modulo>
```

La estructura final del módulo, en formato Maven previo a su compilación, debe ser la siguiente:



6.4 Generación del empaquetado del módulo funcional

6.4.1 Comandos para generación del empaquetado zip final

Para ejecutar los comandos maven desde la línea de comandos debemos situarnos al mismo nivel donde se ubica el archivo pom.xml del proyecto maven del módulo vertical a generar.

Los comandos a ejecutar y en este orden son:

- **mvn clean**

  Borra los compilados generados en compilaciones anteriores.

- **mvn compile**

  Compila todos los fuentes .java haciendo uso de las librerías especificadas en las dependencias.

- **mvn package**

  Construye el artefacto final. Con este comando, en el directorio target, se genera el .jar con los compilados del módulo vertical y el empaquetado zip final.

- **mvn install**

  Instala el artefacto final en el repositorio local (directorio .m2/repository/). Este comando realiza previamente un mvn package. Normalmente este nivel no es necesario a menos que tengamos otros proyectos que dependan del arquetipo generado.

Los comandos anteriores pueden lanzarse en modo offline. Para ello se añade **–o** al final de cada comando. Con esto indicamos a Maven que no se conecte a los repositorios para descargar las librerías. Esta opción resulta de gran utilidad cuando en el repositorio local (directorio .m2/repository/) se encuentran instaladas todas las librerías especificadas en las dependencias del pom.xml.

Para contemplar el resultado de la compilación realizada desde línea de comandos en Eclipse pueden ejecutarse 2 comandos adicionales después de la ejecución de mvn compile:

- **mvn eclipse:clean**

  Borra la compilación previa visualizada desde eclipse.

- **mvn eclipse:eclipse –Dwtpversion=1.5**

  Se realiza la compilación para eclipse, creando los ficheros necesarios para que eclipse interprete el módulo creado como un proyecto web.

7 INTEGRACIÓN MÓDULO VERTICAL EN PLATAFORMA

7.1 Introducción

Una vez configurado el entorno de desarrollo para eclipse y realizado el desarrollo de los módulos funcionales faltaría por configurar cómo va a detectar la plataforma al arrancar que hay diversos módulos funcionales que debe desplegar en el war que ha publicado y el cuál WildFly arrancará. Para ello se realizarán 4 pasos adicionales que nos permitirán desplegar la plataforma con los módulos funcionales.

7.2 Modificación del pom.xml

Cuando WildFly arranque la aplicación, en el war desplegado, deberá contener cada uno de los ficheros de los que se compone un módulo. Para ello utilizamos el plugin de Maven para indicarle los ficheros del módulo que queremos desplegar en el war, así como la ruta exacta donde habrá que copiarlos.

Utilizamos el plugin de maven “maven-resources-plugin” para especificar los ficheros y las rutas. Para facilitar el trabajo de los desarrolladores crearemos un pom.xml padre a todos los módulos que se estén desarrollando, de forma que las directivas se aplicarán de manera independiente a cada uno de los módulos hijos.

El pom.xml padre deberá tener la siguiente estructura:

```
<?xml version="1.0" encoding="UTF-8"?>
<project    xmlns="http://maven.apache.org/POM/4.0.0"    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0    http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>es.juntadeandalucia.ptwanda</groupId>
  <artifactId>DemoVerticalPTWanda</artifactId>
  <packaging>pom</packaging>
  <version>1.0</version>
  <name>Plataforma de tramitacin</name>
  <url>http://maven.apache.org</url>
  <modules>
    <module>ModuloHijo-1</module>
       <!-- OTROS MODULOS HIJOS
    <module>ModuloHijo-2</module>
    <module>ModuloHijo-3</module>
       -->
  </modules>

 <dependencies>
    <dependency>
       <groupId>es.juntadeandalucia.ptwanda</groupId>
       <artifactId>ptwanda-core</artifactId>
       <version>2.3.0</version>
       <scope>provided</scope>
    </dependency>
 </dependencies>

 <build>
    <plugins>
       <!-- NO SE INCLUYEN EN EL JAR NI LOS RECURSOS WEB NI LOS XML DE CONFIGURACION  -->
       <plugin>
          <groupId>org.apache.maven.plugins</groupId>
```

```
          <artifactId>maven-jar-plugin</artifactId>
          <configuration>
             <excludes>
                <exclude>webapp/**</exclude>
                   <exclude>despliegue.xml</exclude>
                   <exclude>conf/**</exclude>
                   <exclude>lib/**</exclude>
                </excludes>
          </configuration>
       </plugin>

        <!-- TODOS LOS PLUGINS CONFIGURADOS HASTA EL FINAL SOLO SON VALIDOS PARA EN EL ENTORNO DE
DESARROLLO -->
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-resources-plugin </artifactId>
          <executions>
             <execution>
                <id>copy-resources-web</id>
                <phase>generate-sources</phase>
                <goals>
                   <goal>copy-resources</goal>
                </goals>
              <configuration>
                    <outputDirectory>${env.PTWANDA_HOME}/WebContent/modulos/${project.artifactId}</
outputDirectory>
                 <encoding>UTF-8</encoding>
                 <includeEmptyDirs>true</includeEmptyDirs>
                 <resources>
                    <resource>
                       <directory>${basedir}/src/main/resources/webapp</directory>
                    </resource>
                 </resources>
               </configuration>
             </execution>
           </executions>
         </plugin>
         <plugin>
           <groupId>org.apache.maven.plugins</groupId>
           <artifactId>maven-resources-plugin </artifactId>
           <executions>
              <execution>
                 <id>copy-resources-struts</id>
                 <phase>generate-sources</phase>
                 <goals>
                    <goal>copy-resources</goal>
                 </goals>
                 <configuration>
                  <outputDirectory>${env.PTWANDA_HOME}/build/classes/modulos/${project.artifactId}</
outputDirectory>
                    <encoding>UTF-8</encoding>
                    <resources>
                       <resource>
                          <directory>${basedir}/src/main/resources/conf</directory>
                       </resource>
                    </resources>
                 </configuration>
              </execution>
            </executions>
          </plugin>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-resources-plugin </artifactId>
            <executions>
              <execution>
                 <id>copy-resources-despliegue</id>
                 <phase>generate-sources</phase>
```

```
                <goals>
                <goal>copy-resources</goal>
              </goals>
           <configuration>
<outputDirectory>${env.PTWANDA_HOME}/build/classes/es/juntadeandalucia/plataforma/
modulospredefinidos</outputDirectory>
              <encoding>UTF-8</encoding>
              <resources>
                 <resource>
                    <directory>${basedir}/src/main/resources</directory>
                    <includes>
                       <include>despliegue*.xml</include>
                    </includes>
                 </resource>
              </resources>
           </configuration>
        </execution>
     </executions>
  </plugin>
  <!-- FIN DE LISTADO DE PLUGINS SOLO VALIDOS PARA EL ENTORNO DE DESARROLLO -->
 </plugins>
</build>


</project>
```

El primer plugin `maven-resources-plugin` es referente a los recursos web, es decir, los ficheros almacenados en el directorio webapp del módulo, tal como se indica en la etiqueta `<directory>${basedir}/src/main/resources/webapp</directory>`. Este plugin cogerá el contenido de la carpeta webapp y lo copiará dentro del proyecto plataforma, concretamente en la ruta especificada en la etiqueta:

```
<outputDirectory>${env.PTWANDA_HOME}/WebContent/modulos/${project.artifactId}</outputDirectory>
```

Se presupone la existencia de una variable de entorno denominada PTWANDA_HOME que contendrá el nombre del proyecto web con el que se importó el war de plataforma en eclipse.

En el segundo bloque se especifica que el archivo de struts del módulo lo copie en la ruta /classes/modulos/`${project.artifactId}`.

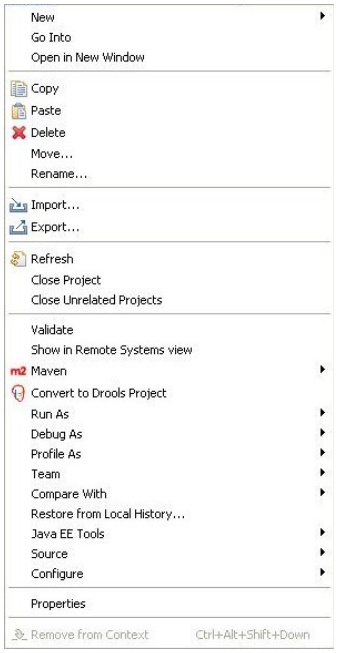
Por último, en el tercer bloque se hace referencia al fichero despliegue.xml. Como podemos comprobar este archivo se copiará en la carpeta modulospredefinidos. Con esto se consigue que el módulo se almacene en base de datos, en la tabla DEFINICIONES_MODULOS. Es importante destacar que **hay que renombrar el fichero con un nombre único.** En este caso lo hemos renombrado como subadm.xml. Esto es debido a que al copiar varios ficheros despliegues.xml de diferentes módulos verticales se estarían machacando continuamente.

A la hora de generar el empaquetado zip final el archivo despliegue deberá renombrarse de nuevo con el nombre despliegue.xml. El renombrado sólo es para el desarrollo, no para el empaquetado.

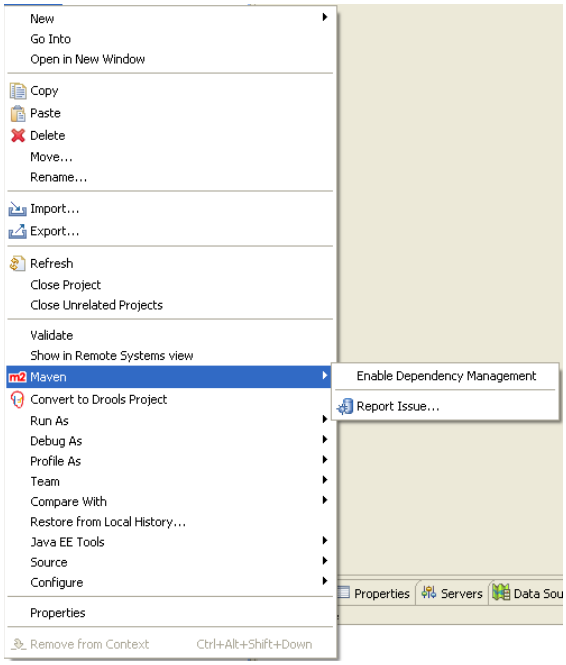
7.3 Activación del plugin Maven 2 en el módulo funcional

Para que los plugins especificados en el pom.xml sean ejecutados es necesario activar el plugin que se instaló en el punto 5.6.1.

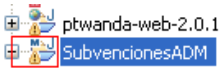
Haciendo click con el botón derecho del ratón sobre el proyecto del módulo funcional se cargan las siguientes opciones:



Pulsamos sobre la opción Maven, desplegándose las siguientes opciones:



Pulsamos sobre la opción *Enable Dependency Management* para activar el plugin. Si la activación fue correcta visualizaremos una pequeña m sobre el nombre del proyecto:



7.4 Modificación del fichero org.eclipse.wst.common.component

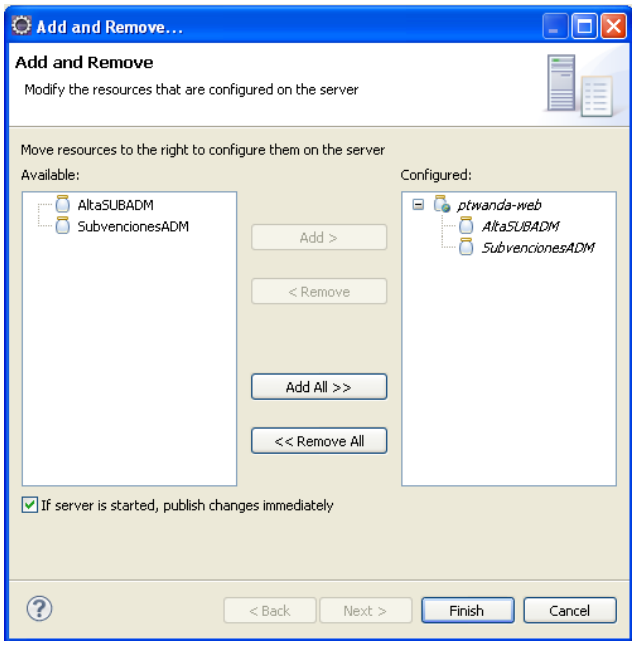
En el punto 7.2 se modificó el pom.xml del módulo funcional para desplegar los recursos web, el archivo de struts y el fichero despliegue.xml, sin embargo, no se mencionó nada de las clases java, applicationContext.xml y demás ficheros como properties, hbm.xml, ect…

Estos ficheros restantes se desplegarán en el war de plataforma incluidos en un jar que genera sólo el servidor de aplicaciones a la hora de publicar el proyecto que va a arrancar. Para ello es necesario editar el fichero **org.eclipse.wst.common.component**, que se encuentra dentro del directorio .settings del proyecto con el que se importó el war de la plataforma:

```
<?xml version="1.0" encoding="UTF-8"?>
<project-modules id="moduleCoreId" project-version="1.5.0">
    <wb-module deploy-name="ptwanda-web">
        <wb-resource deploy-path="/" source-path="/WebContent"/>
        <wb-resource deploy-path="/WEB-INF/classes" source-path="/src"/>
        <property name="context-root" value="ptwanda-web-"/>
        <property name="java-output-path" value="/ptwanda-web/build/classes"/>
        <dependent-module archiveName="SubvencionesADM.jar" deploy-path="/WEB-INF/lib" handle="module:/resource/SubvencionesADM/SubvencionesADM">
            <dependency-type>uses</dependency-type>
        </dependent-module>
    </wb-module>
</project-modules>
```

El jar que se formará tendrá el nombre SubvencionesADM.jar, tal como se indica en la propiedad *archiveName*. El nombre SubvencionesADM que aparece en la propiedad *handle* hace referencia al nombre del proyecto del módulo vertical en el eclipse.

De esta forma a la hora de agregar el proyecto plataforma al servidor WildFly aparecerá una dependencia al módulo agregado:



8 PROGRAMACIÓN DE UN TRÁMITE

8.1 Introducción

En el siguiente apartado se detallan todos los pasos a seguir para realizar la programación de un trámite, que conlleva el alta de expediente, tareas, acciones y condiciones necesarias para el funcionamiento del procedimiento previamente modelado y cargado en Trew@.

8.2 Desarrollo de 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:

IAltaExpedientesService.java

```
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 Pataformapermiso);
   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.xml que contenga los servicios creados.

```
<?xml version='1.0' encoding='utf-8'?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="
            http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

    <bean id="PTWandaService"
        class="es.juntadeandalucia.plataforma.PTWanda.PTWandaServiceImpl"
        scope="prototype">
        <property name="logService">
            <ref bean="logService" />
        </property>
    </bean>

    <bean id="ConfiguracionTramitacionService"
        class="es.juntadeandalucia.plataforma.configuracionTramitacion.ConfiguracionTramitacionServiceImpl"
        scope="prototype" parent="PTWandaService">
        <property name="confService">
            <ref bean="confSistemaService" />
        </property>
    </bean>
```

En este fichero se define el identificador del servicio, mediante la etiqueta “bean id”, así como 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.

Es importante nombrar estos ficheros con el nombre “*applicationContext.xml*”, para que el framework Spring los inyecte (cargue) automáticamente al estar definido de tal forma en el fichero “web.xml”, de la aplicación Plataforma de Tramitación:

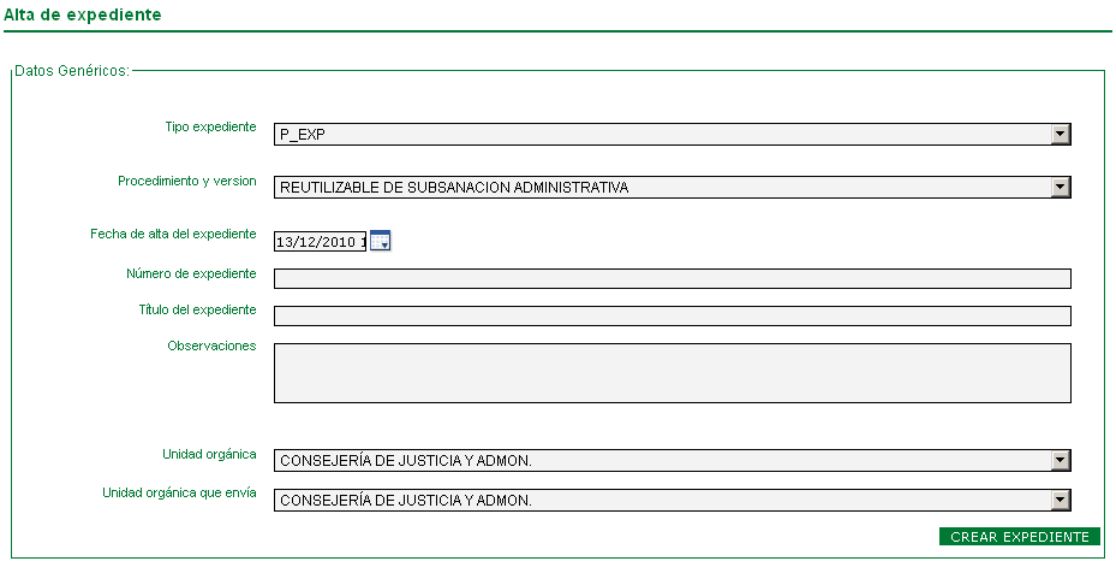
```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
                classpath*:/applicationContext.xml,
        </param-value>

</context-param>
```

8.3 Desarrollo de un alta de 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.

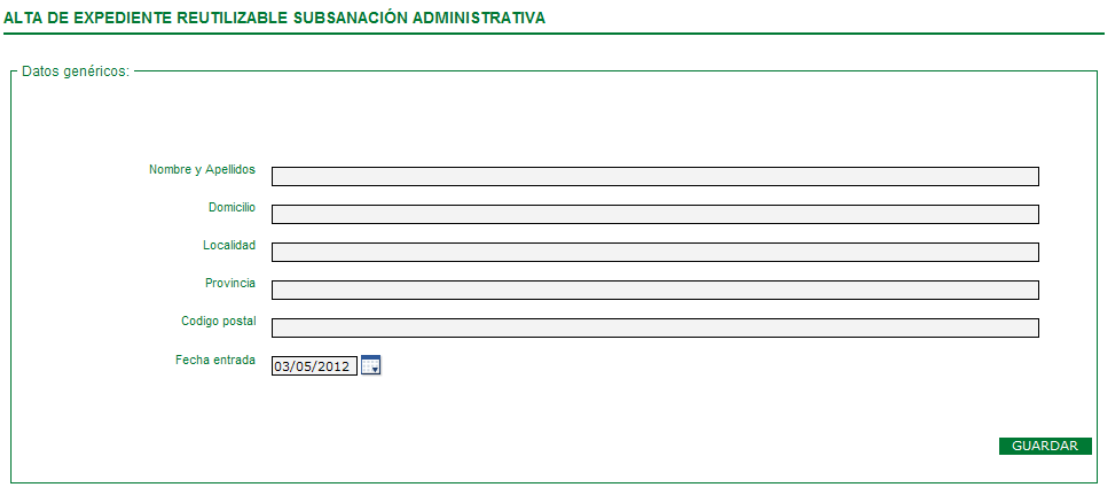
El proceso de alta de expediente se divide en dos partes fundamentales:

- 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 particulares del procedimiento seleccionado.



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

- 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.

```
# altaSUBADM.jsp

<s:form enctype="multipart/form-data"  theme="simple" name="moduloAltaExpEspecifica">

.....
.....
<s:submit value="Guardar" action="guardaAltaSUBADM"/>
```

A partir de la versión 2.0 de PTw@ndA es posible utilizar formul@ como generador de formularios de forma que no sea necesario desarrollar los formularios de captura de datos. Ver apartado 9.10 para la utilización de Formul@.

- strutsXXX.xml: 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.

```
# struts-alta-expediente.xml
<action name="guardaAltaSUBADM" method="guardaAltaSUBADM"
                                        class="AltaSUBADM">
        <result name="localizacion" type="redirect">
            /busqueda/irABuscadorGenerico.action?refExpedienteCreado=${numExpCreado}
            &amp;refNumExpediente=${numero}
        </result>
        <result name="input">altaSUBADM.jsp</result>
        <result name="error">/inicio/error.jsp</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.

```
# struts.xml

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

Indicar que los cambios en este fichero struts.xml, del proyecto "plataforma-sdk", sólo son necesarios realizarlos en un entorno de Desarrollo. Estos cambios no se reflejan posteriormente en módulo .zip a crear. Plataforma de Tramitación en tiempo de Despliegue en los entornos de Producción, cuando se incluya el nuevo .zip del módulo realizará estos cambios en lel fichero "struts.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@, tanto para recuperar el identificador del procedimiento que va a dar de alta, como salvaguardar el mismo.

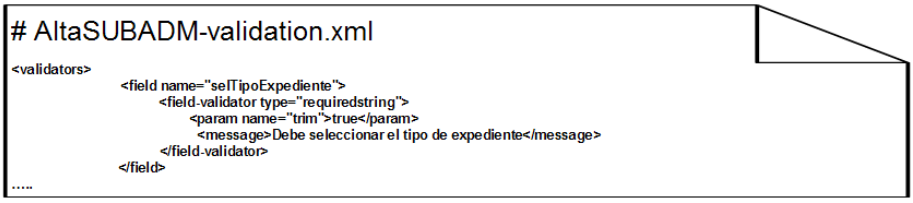
AltaSUBADM.java

```
public String guardaAltaSUBADM()
  {
     boolean datosCorrectos = false;
     List nuevoExpediente = null;
     user = (UsuarioWeb)session.get("usuario_en_sesion");
     configurarServicio(user, altaExpedientesService);
     OtrosDatosExpedientes datosExp = new OtrosDatosExpedientes();
     datosCorrectos = comprobarDatos();
     if(datosCorrectos)
     {
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        String dia = sdf.format(new Date());
        DateFormat date = DateFormat.getTimeInstance(3);
        String hora = date.format(new Date());
        String unidad = "";
        datosExp.setNombreSolicitante(nombreSolicitante);
        datosExp.setDomicilioSolicitante(domicilioSolicitante);
        datosExp.setLocalidadSolicitante(localidadSolicitante);
        datosExp.setProvinciaSolicitante(provinciaSolicitante);
        datosExp.setCpostalSolicitante(cpostalSolicitante);
        datosExp.setFechaEntradaReg(fechaEntradaReg);
        altaExpedientesService.setOtrosDatos(datosExp.toString());
        String resultado = altaExpedientesService.generarExpediente(selTipoExpediente, selProcVersion, dia, titulo, observaciones, null, null, user,
Resources.getPropiedad("buscador_servidorSolr"));
        if(!resultado.equals("success"))
        {
           return "error";
        }
        configurarServicio(user, consultaExpedienteService);
        try
        {
           nuevoExpediente = consultaExpedienteService.obtenerExpedientes(altaExpedientesService.getIdExpedienteCreado().toString(), null, null);
        }
        catch(BusinessException e)
        {
           return "error";
        }
        if(nuevoExpediente == null)
        {
           return "error";
        } else
        {
           numero = ((IExpediente)nuevoExpediente.get(0)).getRefExpediente();
           setRefExpediente(((IExpediente)nuevoExpediente.get(0)).getRefExpediente());
           setNumExpCreado(numero);
           setRefExpedienteCreado(numero);
           return "localizacion";
        }
     } else
     {
        return "input";
     }
  }
```

- applicationContext.xml: en este fichero se definen los identificadores de las clases Java (acciones), así como todos los servicios empleados por dichas clases. En este fichero se definen las clases Java que gestionarán las altas de los diferentes tipos de procedimientos y las acciones necesarias en las tareas, así como la definición de las clases Java que implementan los servicios desarrollados por y para el proyecto vertical.

```
<bean id="AltaSUBADM" class="com.everis.AltaSUBADM.altaExpediente.AltaSUBADM" scope="prototype">
    <property name="altaExpedientesService">
    <ref bean="altaExpedientesService"/>
    </property>
    <property name="consultaExpedienteService">
    <ref bean="consultaExpedienteService"/>
    </property>
    <property name="logService">
    <ref bean="logService"/>
    </property>
    <property name="tramitacionService">
    <ref bean="tramitacionService"/>
    </property>
    <property name="indexacionService">
    <ref bean="indexacionService"/>
    </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.



8.4 Desarrollo de Tareas

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

- Tareas de incorporar documentos.
- Tareas de generación de documentos.
- Tareas Web o de adquisición de datos.
- Tareas tipo Otros.

8.4.1 Tareas tipo Incorporar documento

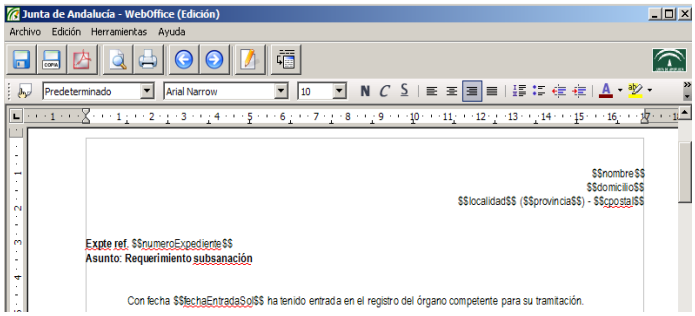
Para las tareas tipo incorporar no es necesario ningún desarrollo en la Plataforma de Tramitación. Una vez modelada la tarea en Trew@, automáticamente en el módulo funcional de "*Documentos y Tareas a realizar*" se obtendrá la tarea para incorporar documentos.

8.4.2 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.

Desde la versión 2.3.0 de PTw@ndA se incluye un editor de texto alternativo a WebOffice que permite la edición del documento de una forma más eficiente y usable para el usuario tramitador, pudiéndose configurar el nuevo editor desde la herramienta de administración de la plataforma, mediante el parámetro de configuración EDITOR_TEXTOS_DEFECTO.




```
# Variables.java

public class VariablesCJAP extends trewa.ext.TrAccesoUI {

public String obtCifEmpresa(Integer id)throws TrException
{
   String res = "";
   try{
       TrAPIUI apiui = getApiUI();
       String xml_cad = apiui.obtenerOtrosDatos(new TpoPK(id), "E");
       OtrosDatosExpedientes datosExp = null;
       datosExp = new OtrosDatosExpedientes(xml_cad);
       res = datosExp.getCifEmpresa();
       if(res == null)
           res = "";

       return res;
  } catch (Exception e) {
      return "";
   }
 }
-----
```

8.4.3 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:

```
# struts-tareasSUBADM.xml

...
<!-- Fase: Validacion de la solicitud -->
<action name="tareaEntradaSUBADM_R_REGISTRAR_SENTIDO_DOC" method="inicio"class="tareasSUBADM_Action">
                <result name="success">/modulos/SUBADM/tareaEntradaSUBADM_R_REGISTRAR_SENTIDO_DOC.jsp</result>
                        <interceptor-ref name="basicStack" />
</action>

<action name="guardarDatosValSol" method="guardarDatosValSol"class="tareasSUBADM_Action">
                <result name="success">/modulos/SUBADM/Correcto.jsp</result>
                <result name="error">/modulos/SUBADM/Error.jsp</result>
                <result name="recarga">/modulos/SUBADM/tareaEntradaSUBADM_R_REGISTRAR_SENTIDO_DOC.jsp</result>
                        <interceptor-ref name="basicStack" />
</action>

....
```

Recordar: este fichero de struts-xxxxx.xml se ha de incluir en el fichero “struts.xml”. Indicar, al mismo tiempo, que los cambios en este fichero struts.xml, del proyecto “plataforma-sdk”, solo es necesario realizarlo en un entorno de desarrollo. Estos cambios no se reflejan posteriormente en módulo .zip a crear. Plataforma de Tramitación en tiempo de despliegue en los entornos de producción, cuando se incluya el nuevo .zip del módulo realizará estos cambios en lel fichero “struts.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.

 # tareaEntradaSUBADM_R_REGISTRAR_SENTIDO_DOC.jsp

```
<!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="Verificacion de la documentacion" /></title>

<!-- HOJAS DE ESTILOS -->
<link rel="stylesheet" type="text/css"                  href="../../instalaciones/css/hojaEstiloAplicacion.css" />
<s:head theme="ajax"/>
</head>
<body>
                <s:div id="capa_ejecutarTarea">
                                <center>
                                                <div id="contenedorCabecera">
                                                                <div id="cabeceraPaginaAplicacion"></div>
                                                                <div id="rellenoCabeceraPagina"></div>
                                                </div>
                                                <s:actionerror />
                                                <s:fielderror  />
                                                </center>

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

      <center>
        <s:form  theme="simple" name="formulario">
<s:div cssStyle="width:90%">
                        <s:div cssClass="contenedorCampos">
                                <br>
<s:text name="tit">Verificacion de la documentacion</s:text>
<s:div cssStyle="width:100%;text-align:left;margin-left:46%">
<s:if test="${solicitudValida == 1}">
<input type="radio" name="documentacionValida" checked="checked" value="Documentacion valida"> Documentacion valida <br>
<input type="radio" name="documentacionValida" value="Documentacion no valida">Documentacion no valida <br>
</s:if>
<s:else>
<input type="radio" name="documentacionValida" value="Solicitud valida">Documentacion valida <br>
<input type="radio" name="documentacionValida" checked="checked" value="Documentacion no valida"> Documentacion no valida <br>
</s:else>
</s:div>
</s:div>
</s:div>
<br>
<s:div id="AE_botones" theme="simple" cssStyle="width:20%;text-align:center;">
<table>
                        <tr>
                                        <td><s:submit value="Guardar" action="guardarDatosValSol" /></td>
                                        <td><s:submit value="Cancelar" onclick="self.close()"/></td>
                        </tr>
</table>
</s:div>
</s:form>
</center>
</s:div>

<s:div cssStyle="width:100%;color:red;font-weight:bold;" cssClass="mensajesError">
                        <s:property value="mensajeError"/>
</s:div>
</body>
</html>
```

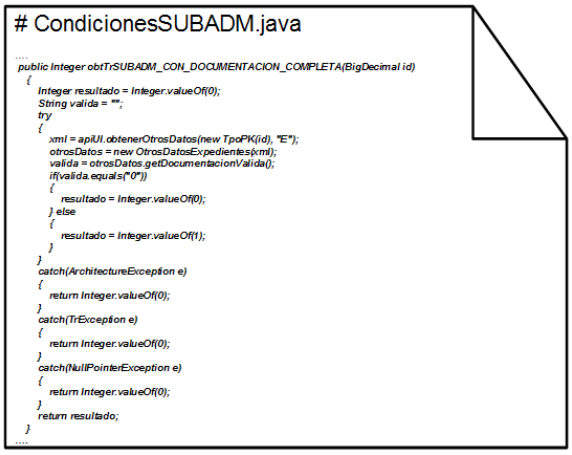
A partir de la versión 2.0 de PTw@ndA es posible utilizar formul@ como generador de formularios de tareas de manipulación de datos, de forma que no sea necesario desarrollar los formularios de captura de datos. Ver apartado 9.8 para la utilización de Formul@.

8.5 Desarrollo de 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 TREW@, 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 transición asociada.

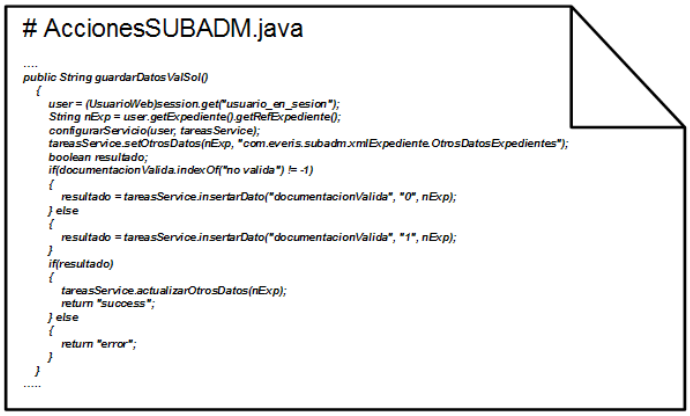


8.6 Desarrollo de acciones

Las acciones son eventos lanzados cuando ocurre un hecho indicado en TREW@ 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.



8.7 Desarrollo de módulos tipo utilidades

8.7.1 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.

8.7.2 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>
        <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>
        <autor>
            <nombre>ute</nombre>
            <email>ute@utes.com</email>
            <grupo>staff</grupo>
         </autor>
         <type>STRUTS-2</type>
```

```
        <tipoInstalacion>UTILIDADES</tipoInstalacion>
        <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:

%Proyecto_Vertical%\WEB-INF\classes\es\juntadeandalucia\plataforma\modulospredefinidos

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

- 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-2.

  - Un fichero de Struts-2: este 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

struts-utilidadCJAP.xml

```
<struts>
   <constant name="struts.objectFactory" value="spring" />
   <constant name="struts.devMode" value="true" />
<package name="datosExpedienteReas" namespace="/modulos/datosExpedienteReas"
   extends="struts-default">
<global-results>
<result name="error">/administracion/error.jsp</result>
</global-results>
<!-- UTILIDAD : MODIFICAR DATOS EXPEDIENTE -->
<action name="datosExpedienteReas" method="datosExpedienteReas"
class="datosExpedienteReasAction">
<result name="ins">/modulos/datosExpedienteReas/datosExpedienteReasINS.jsp</result>
<result name="cer">/modulos/datosExpedienteReas/datosExpedienteReasCER.jsp</result>
<result name="ext">/modulos/datosExpedienteReas/datosExpedienteReasEXT.jsp</result>
<result name="can">/modulos/datosExpedienteReas/datosExpedienteReasCAN.jsp</result>
<result name="mod">/modulos/datosExpedienteReas/datosExpedienteReasMOD.jsp</result>
<result name="cao">/modulos/datosExpedienteReas/datosExpedienteReasCAO.jsp</result>
<result name="ren">/modulos/datosExpedienteReas/datosExpedienteReasREN.jsp</result>

<result name="error">/modulos/datosExpedienteReas/Error.jsp</result>
<interceptor-ref name="basicStack" />
</action>

<!-- INS -->
<action name="guardarDatosReasIns" method="guardarDatosReasINS"
class="datosExpedienteReasAction">
<result name="success">/modulos/datosExpedienteReas/Correcto.jsp</result>
<result name="error">/modulos/datosExpedienteReas/Error.jsp</result>
<interceptor-ref name="basicStack" />
</action>
```

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

  Indicar que los cambios en este fichero struts.xml, del proyecto "plataforma-sdk", sólo es necesario realizarlo en un entorno de desarrollo. Estos cambios no se reflejan posteriormente en módulo .zip a crear. Plataforma de Tramitación en tiempo de despliegue en los entornos de producción, cuando se incluya el nuevo .zip del módulo realizará estos cambios en lel fichero "struts.xml".

- Programación de 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:

DatosExpedienteCJAPAction.java

```
public DatosExpedienteCJAPAction(){
}
public String datosExpedienteCJAP(){
     int version;
     ResourceBundle propiedades = ResourceBundle.getBundle("cjap");
     UsuarioWeb user = (UsuarioWeb) session.get("usuario_en_sesion");
     IExpediente exp = user.getExpediente();

     session.put("expediente",exp.getRefExpediente());
     try {
          //Obtenemos el id del procedimiento del expediente desde el que se
          //carga la utilidad
          version = Integer.parseInt(exp.getProcedimiento().getRefProcedimiento());

     .....
```

- Programar 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 el JSP.
- Añadir al fichero applicationContext.xml que utilice Plataforma de Tramitación, un nuevo "Bean" definiendo el identificador de la clase Java, así como todos los servicios que ésta use.

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.

9 BUENAS PRÁCTICAS

9.1 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 PTw@ndA.
- **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 PTw@ndA.
- **listaPerfiles**: lista de objetos de tipo `Perfil` 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*

Se desaconseja la transmisión de valores entre fases de una misma petición en el ámbito de la sesión dado que en entornos de despliegue de alta disponibilidad, los valores almacenados se replican en todos los nodos con los costes de computación y comunicación asociados. Para estas necesidades es preferible el uso del ámbito “request.setAttribute”.

9.2 Recarga selectiva de portlets

El escritorio de tramitación puede configurarse para que no realice una recarga completa de todos los portlets que lo componen cuando se regrese a él después de la realización de una tarea, generación de un documento, realización de una utilidad, etcétera.

En el caso de querer realizar una recarga completa de los portlets debemos definir el correspondiente submit del siguiente modo:

```
<input type="submit" value="cerrar"
```

```
   onclick="javascript:window.opener.location.reload();window.close();" />
```

De esta manera se especifica que cierre la ventana actual y recargue la ventana padre completa del escritorio de tramitación.

En el caso de querer recargar un portlet en concreto habrá que definirlo de la siguiente manera:

```
<s:submit value="Cerrar" name="btnCerrar" id="btnCerrar"
   onclick="javascript:opener.recargaPadre([['capa_doc_asoc',
   '../modulos/docsAsociadosExpediente/listarDocumentos.action']]);self.close();"/>
```

Mediante la función javascript recargaPadre indicamos el portlet a recargar. Esta función recibe 1 parámetro:

- Un array con 2 parámetros: el nombre del módulo a recargar (campo NOMBRE de la tabla DEFINICIONES_MODULOS) y la ruta con el action que carga el módulo (campo URL de la tabla DEFINICIONES_MODULOS).

Posteriormente se cierra la ventana hija con la instrucción self.close().

9.3 Invocación de los servicios de indexación

La Plataforma de Tramitación implementa un servicio exclusivo para la indexación de la información especificada en el fichero schema.xml. Para realizar el mapeo de los diferentes campos a poder indexar se ha creado un objeto que mapea dichos campos.

El objeto **ExpedienteTrewa,** localizado en el paquete **es.juntadeandalucia.plataforma.expediente**, contiene toda la información necesaria que será utilizada por el servicio de indexación. La información contenida en el objeto de Trew@ TrExpediente será almacenada íntegramente como un atributo en este objeto.

```java
package es.juntadeandalucia.plataforma.expediente;

/**
 * <p>
 * Implementaci&oacute;n de un expediente del sistema utilizando el <i>API</i>
 * de Trew@ para acceder a la informaci&oacute;n relativa al mismo.
 * </p>
 *
 * @author everis
 * @version 1.0
 */
public class ExpedienteTrewa implements IExpediente, Serializable {



    /**
     *
     */
    private static final long serialVersionUID = -3713078992878032216L;

    private TrExpediente expediente;

    public ExpedienteTrewa() {

    }

    public ExpedienteTrewa(TrExpediente expediente, ISistema sistema, IUsuario usuario, String idServicio) {
        this.expediente = expediente;
        this.sistema = sistema;
        this.usuario = usuario;
        this.idServicio = idServicio;
    }
```

El servicio encargado de recibir la información e indexarla en el motor de indexación Solr se denomina **IIndexacionService,** el cuál implementa 3 métodos para la comunicación con el motor de indexación:

- **crearExpediente:** método que recibe un objeto IExpediente e indexa su contenido en función de los campos que se han definido en el schema de indexación de Solr (schema.xml).
- **actualizarExpediente:** método que recibe un objeto IExpediente y actualiza el contenido de la información en el núcleo de Solr. En caso de no existir el objeto en el núcleo, se creará un índice nuevo que contendrá la información correspondiente.
- **eliminarExpediente:** método que recibe una cadena con la referencia del expediente y elimina del núcleo de Solr toda la información asociada a él.

La interfaz de este servicio es la siguiente:

```java
package es.juntadeandalucia.plataforma.service.busqueda;

/**
 * <p>
 * Interfaz que define las operaciones p&uacute;blicas disponibles para la
 * indexaci&oacute;n de contenidos en el motor de b&uacute;squeda de PTw@ndA.
 * </p>
 *
 * @author everis
 * @version 1.0
 */
public interface IIndexacionService extends IPublicService, IPTWandaService {

    /**
     * Método que extrae la información del expediente para indexarlo.
     * @param expediente Expediente a indexar
     * @throws ArchitectureException Excepción de arquitectura
     * @throws BusinessException Excepción de negocio
     */
    public void crearExpediente(IExpediente expediente) throws ArchitectureException, BusinessException;


    /**
     * Actualiza la información de un expediente en el motor de indexación.
     * @param expediente Expediente a actualizar.
     * @throws ArchitectureException Excepción de arquitectura
     * @throws BusinessException Excepción de Negocio
     */
    public void actualizarExpediente(IExpediente expediente) throws ArchitectureException, BusinessException;


    /**
     * Método que elimina un expediente del motor de indexación
     * @param numExpediente Identificador del expediente a eliminar
     * @throws ArchitectureException Expepción de arquitectura
     */
    public void eliminarExpediente(String numExpediente) throws ArchitectureException;

}
```

9.4 Indexación desde un cliente Solr.

Una de las ventajas de la indexación es la posibilidad de utilizar un cliente externo a la plataforma, con el objetivo de indexar en el servidor Solr la información obtenida.

Para tal efecto Plataforma proporciona un cliente Solr habilitando una API para realizar la indexación. Esta API se compone de los siguientes métodos:

```java
/**
 * Interface que publica los métodos necesarios para realizar el envío de un
 * mensaje a la cola JMS desde un cliente externo a PTWANDA.
 *
 * @author everis
 *
 */
public interface IIndexacionSolr {

    /**
     * Método que envia un nuevo mensaje con la informacion indexable.
     *
     * @param url
     *            del servicio de PTWANDA
     * @param informacionIndexable
     *            información a indexar
     * @param userSolr
     *            the userSolr
     * @param passwordSolr
     *            the passwordSolr
     *
     * @return resultado de la operacion
     */
    boolean enviarMensajeParaIndexar(String urlService,
            Map<String, String> informacionIndexable, String userSolr,
            String passwordSolr);

    /**
     * Método que envia un mensaje para eliminar el índice que se corresponde
     * con el expediente pasado como parámetro.
     *
     * @param url
     *            del servicio de PTWANDA
     * @param numeroExpedientes
     *            numero del expediente a eliminar del servidor solr
     * @param userSolr
     *            the userSolr
     * @param passwordSolr
     *            the passwordSolr
     *
     * @return resultado de la operacion
     */
    boolean enviarMensajeEliminarIndice(String urlService,
            String numeroExpedientes, String userSolr, String passwordSolr);

}
```

Los parámetros a especificar para utilizar los métodos de la API son:

- **numeroExpediente**: número del expediente a eliminar.
- **urlService**: URL donde se encuentra publicado el servicio web de plataforma. La url tendría el siguiente formato:
  - http://ip:puerto/contexto/WebServices/PlataformaWS/plataforma-tramitacion.wsdl.
- **informacionIndexable:** Mapa con la información a indexar. Los campos a indexar deben existir en el esquema de indexación de plataforma (fichero schema.xml).
- **usuarioSolr:** usuario habilitado para realizar peticiones HTTP al componente Solr.
- **passwordSolr:** password del usuario habilitado para realizar peticiones HTTP al componente Solr.

Los campos que actualmente están definidos en el esquema de indexación de Plataforma son:

- **id** --> columna T_NUM_EXP de la tabla TR_EXPEDIENTES del esquema de Trew@.
- **numeroExp** --> columna T_NUM_EXP de la tabla TR_EXPEDIENTES del esquema de Trew@.
- **tituloExp** --> columna T_TITULO_EXP de la tabla TR_EXPEDIENTES del esquema de Trew@.
- **observacionesExp** --> columna T_OBSERVACIONES de la tabla TR_EXPEDIENTES del esquema de Trew@.
- **unidadOrg** --> entre las siglas UO debe estar el valor de la columna UORG_X_UORG de la tabla TR_EXPEDIENTES del esquema de Trew@.
- **fechaAltaExp** --> columna F_CREACION de la tabla TR_EXPEDIENTES del esquema de Trew@.
- **fase** --> columna X_FASE de la tabla TR_FASES del esquema de Trew@.
- **refFaseExp** --> columna X_FASE de la tabla TR_FASES del esquema de Trew@.
- **procedimientoId** --> columna TIEV_X_TIEV de la tabla TR_DEFPROCS_GR del esquema de Trew@.
- **sistema** --> columna C_SISTEMA de la tabla GN_SISTEMAS del esquema de Trew@.
- **refExp** --> columna X_EXPE de la tabla TR_EXPEDIENTES del esquema de Trew@.
- **faseExp** --> columna D_DESCRIPCION de la tabla TR_FASES del esquema de Trew@.
- **archivadoExp** --> columna F_ARCHIVO de la tabla TR_EXPEDIENTES del esquema de Trew@.
- **procedimientoExp** --> columna D_DESCRIPCION de la tabla TR_TIPOS_EVOLUCIONES del esquema de Trew@.
- **interesados** -> Nombre, apellidos e identificador de los interesados del expediente.

Ejemplo:

```
Map<String, String> informacionIndexable = new HashMap<String, String> ();

informacionIndexable.put("id", "Expediente 17 Marzo 1.2");
informacionIndexable.put("numeroExp", "Expediente 17 Marzo 1.2");
informacionIndexable.put("tituloExp", "Expediente 17 Marzo 1.2");
informacionIndexable.put("observacionesExp", "Expediente 17 Marzo 1.2");
informacionIndexable.put("unidadOrg", "UO1UO");
informacionIndexable.put("fechaAltaExp", "2011-01-19T00:00:00Z");
informacionIndexable.put("fase", "5");
informacionIndexable.put("refFaseExp", "5");
informacionIndexable.put("procedimientoId", "1");
informacionIndexable.put("sistema", "PT_WANDA");
```

informacionIndexable.put("refExp", "772");

informacionIndexable.put("faseExp", "VERIFICACION_DOCUMENTACION");

informacionIndexable.put("archivadoExp", "false");

informacionIndexable.put("procedimientoExp", "REUTILIZABLE DE SUBSANACION ADMINISTRATIVA");

informacionIndexable.put("exp_0_interesado", "Pedro Perez López 00000000T");

Para poder hacer uso del cliente Solr de PTw@ndA será necesario incluir en el lib de la aplicación externa las siguientes librerías:

- **ptwanda-clienteSolr-.jar**

En el caso de desarrollar la aplicación utilizando Maven habrá que añadir la siguiente dependencia en el pom.xml:

```
<dependency>
   <groupId>es.juntadeandalucia.plataforma</groupId>
   <artifactId>ptwanda-clienteSolr</artifactId>
   <version></version>
</dependency>
```

A continuación se expone un ejemplo de cómo realizar la invocación al servicio de indexación a través de la API:

```
Map<String, String> informacionIndexable = new HashMap<String, String> ();

informacionIndexable.put("id", "Expediente 17 Marzo 1.2");
informacionIndexable.put("numeroExp", "Expediente 17 Marzo 1.2");
informacionIndexable.put("tituloExp", "Expediente 17 Marzo 1.2");
informacionIndexable.put("observacionesExp", "Expediente 17 Marzo 1.2");
informacionIndexable.put("unidadOrg", "UO1UO");
informacionIndexable.put("fechaAltaExp", "2011-01-19T00:00:00Z");
informacionIndexable.put("fase", "5");
informacionIndexable.put("refFaseExp", "5");
informacionIndexable.put("procedimientoId", "1");
informacionIndexable.put("sistema", "PT_WANDA");
informacionIndexable.put("refExp", "772");
informacionIndexable.put("faseExp", "FASE: VERIFICACION_DOCUMENTACION");
informacionIndexable.put("archivadoExp", "false");
informacionIndexable.put("procedimientoExp", "REUTILIZABLE DE SUBSANACION ADMINISTRATIVA");
informacionIndexable.put("exp_0_interesado", "Pedro Perez López 00000000T");
informacionIndexable.put("exp_1_interesado", "Ana Martín Sánchez 11111111H");



IIndexacionSolr cliente = new IndexacionSolr();
```

```
cliente.enviarMensajeParaIndexar("http://7.128.80.14:8080/ptwanda-web/WebServices/PlataformaWS/
plataforma-tramitacion.wsdl.",informacionIndexable,"usuario","password");
```

9.5 Parámetros disponibles para el cálculo de variables

A la hora de invocarse desde PTw@ndA a las tareas de generación de documentos, el sistema recupera los siguientes parámetros:

- P_EXP: Referencia del expediente
- P_USR: Código del Usuario Web
- P_FASE: Identificador de la fase
- P_REF_DOC: Referencia del documento
- P_PROC: Identificador del procedimiento
- P_TIPO_DOC: Referencia del tipo de documento

Estos parámetros están disponibles para su utilización a la hora de calcular una determinada variable. Para ello deberán definirse en Trew@ los parámetros de entrada de cada variable con el nombre exacto especificado anterioremente.

9.6 Uso de interceptores proporcionados por PTw@ndA

A partir de la versión 2.1 de PTw@ndA es posible hacer uso de una serie de interceptores que se han implementado y que facilitan las labores de desarrollo así como hacen más seguro el módulo funcional que se está desarrollando.. Para aclarar el funcionamiento de un interceptor, se hará una breve definición de los mismos.

Un Interceptor en el framework Struts es un método que se ejecuta antes y después de la acción que se está ejecutando. Con lo que se puede modificar el flujo de la acción para que se ejecute un método de una forma limpia y transparente al resto del flujo.

A continuación se enumeran los distintos interceptores así como se explica cómo debe usarlo en su implementación el desarrollador:

9.6.1 Interceptor de autenticación de usuarios

La funcionalidad aportada por el interceptor, es la de la validación de un usuario que está intentando ejecutar una acción perteneciente a un módulo, comprobando que dicho usuario posee el perfil necesario para tener acceso a dicho módulo. Con el uso de este interceptor evitamos que un usuario que no tiene visibilidad sobre un módulo pueda acceder a la funcionalidad aportada por dicho módulo escribiendo la url de dicho método directamente en el navegador.

Para usar el interceptor, tenemos que modificar el fichero de configuración de struts de la siguiente forma:

En primer lugar tenemos que hacer que el paquete de acciones que estamos definiendo extienda del paquete en el que está definido el interceptor, es decir tenemos que incluir en el atributo “extends” de la etiqueta “package”. Ejemplo:

<package name=”moduloA” namespace=”modulos/moduloA” extends="AccessValidator">

Una vez realizado este paso, sólo nos queda incluir el interceptor en cada una de las acciones en las que queramos que se ejecute, para ello tenemos que incluir la etiqueta <interceptor-ref name="AccessValidationInterceptor"/> en el action correspondiente. Ejemplo:

```
<action name="inicio" class=" moduloAAction" method="inicio">
        <result name="success">/modulos/ moduloA /inicio.jsp</result>
        <interceptor-ref name="AccessValidationInterceptor"/>
</action>
```

Es Importante destacar que para que el interceptor funcione de forma correcta existe una restricción, ya que el nombre del namespace debe contener el nombre del modulo al final de la cadena, ya que a partir de ahí se obtiene el nombre del módulo del que se comprobarán los permisos.

9.6.2 Interceptor para incluir “migas de pan”

El uso de este interceptor permite añadir una miga de pan al módulo, las migas de pan o “breadcrumb” aportan a un usuario un rastro de por dónde ha navegado, ya que es conveniente que el usuario sepa en todo momento dónde se encuentra, cómo ha llegado hasta allí y tener la posibilidad de volver a una posición anterior de su camino en el momento que él lo desee.

Para hacer uso de él, al igual que en el caso anterior es necesario configurar el fichero de struts del módulo:

Para empezar el paquete del módulo debe extender del paquete en el que está definido el interceptor, es decir tenemos que incluirlo en el atributo “extends” de la etiqueta “package”. Ejemplo:

<package name=”moduloA” namespace=”modulos/moduloA” extends="migas">

A continuación, tenemos que incluir el interceptor en las acciones que queremos que se incluyan como miga, por lo que tenemos que incluir la etiqueta <interceptor-ref name="miga"/> en el action correspondiente. Ejemplo:

```
<action name="inicio" class=" moduloAAction" method="inicio">
        <result name="success">/modulos/ moduloA /inicio.jsp</result>
        <interceptor-ref name="miga"/>
</action>
```

9.6.3 Interceptor para evitar ataques XSS

Gracias al uso de este interceptor podemos evitar los ataques XSS (Cross Site Scripting) a nuestra aplicación, estos ataques abarcan cualquier ataque que permita ejecutar código de "scripting", como VBScript o JavaScript, en el contexto de otro sitio web. El problema se soluciona validando correctamente los datos de entrada que son insertados en cualquier formulario de la aplicación. Esta vulnerabilidad puede estar presente de forma directa (también llamada persistente) o indirecta (también llamada reflejada).

Para su uso el fichero de configuración de struts del módulo debe cumplir ciertas restricciones:

El paquete de definición del módulo tiene que heredar del paquete en el cual está definido el interceptor, es decir tenemos que incluirlo en el atributo “extends” de la etiqueta “package”. Ejemplo:

```
<package name=”moduloA” namespace=”modulos/moduloA” extends="XSSValidatorInterceptor">
```

Tras este paso, hay que añadir una etiqueta “interceptor-ref” en las acciones que gestionen las acciones de guardado de datos de formulario. Ejemplo:

```
<action name="inicio" class=" moduloAAction" method="inicio">
        <result name="success">/modulos/ moduloA /inicio.jsp</result>
        <interceptor-ref name="XSSValidator"/>
</action>
```

9.6.4 Interceptor para evitar ataques CSRF

El uso de este interceptor permite a una aplicación prevenir ataques CSRF. Un ataque CSRF fuerza al navegador web validado de una víctima a enviar una petición a una aplicación web vulnerable, la cual entonces realiza la acción elegida a través de la víctima. Al contrario que en los ataques XSS, vistos en el apartado anterior, los cuales explotan la confianza que un usuario tiene en un sitio en particular, el cross site request forgery explota la confianza que un sitio tiene en un usuario en particular. Para evitar este tipo de ataques se hace uso de tokens, existe un token para cada usuario logado en el sistema, y cuando se lance una petición a la aplicación se comprueba que el token recibido coincide con el token del usuario logado, en caso contrario no se ejecuta la petición.

Para hacer uso del interceptor hay que realizar la siguiente configuración:

En primer lugar tenemos que hacer que el paquete de acciones que estamos definiendo extienda del paquete en el que está definido el interceptor, es decir tenemos que incluir en el atributo “extends” de la etiqueta “package”. Ejemplo:

```
<package name=”moduloA” namespace=”modulos/moduloA” extends="TokenValidator">
```

Una vez realizado este paso, sólo nos queda incluir el interceptor en cada una de las acciones en las que queramos que se ejecute, para ello tenemos que incluir la etiqueta <interceptor-ref name="TokensValidationInterceptor"/> en el action correspondiente. Ejemplo:

```
<action name="inicio" class=" moduloAAction" method="inicio">
        <result name="success">/modulos/ moduloA /inicio.jsp</result>
        <interceptor-ref name="TokensValidationInterceptor"/>
</action>
```

9.6.5 Uso de múltiples interceptores

Cabe destacar que el uso de los interceptores no es excluyente, por lo que podemos usar varios de ellos en la misma acción. Para ello simplemente tenemos que hacer extender de tantos paquetes de interceptores como sea necesario:

```
<package name=”moduloA” namespace=”modulos/moduloA” extends="migas,XSSValidatorInterceptor,AccessValidator">
```

Y luego hay que añadir a cada action tantos interceptores como queramos que se ejecuten previamente a la ejecución de la acción que se haya producido.

```
<action name="inicio" class=" moduloAAction" method="inicio">
        <result name="success">/modulos/ moduloA /inicio.jsp</result>
        <interceptor-ref name="miga"/>
        <interceptor-ref name="XSSValidator"/>
```

</action>

9.7 Uso de Formul@ como generador de formularios

A partir de la versión 2.0 de PTw@ndA es posible utilizar formul@ como generador de formularios de tareas de manipulación de datos, de forma que no sea necesario desarrollar los formularios de captura de datos. Para ello será necesario realizar las siguientes acciones:

I.Nomenclatura de la tarea de manipulación de datos:

Para la utilización de formul@ como generador de formularios es indispensable definir en el modelado del procedimiento el nombre de la tarea que hará uso del formulario creado en formula con la siguiente nomenclatura:

FORMULA:CODIGO_FORMULARIO:ACTION_INICIO:ACTION_FIN

Al detectar esta nomenclatura en la tarea, PTw@ndA interpreta que la tarea de manipulación de datos no está definida mediante una JSP o action de struts, sino que se ha creado mediante el motor de formularios formul@.

II.Action de inicio:

Para su invocación, el usuario deberá crear el código java que implementa el Action definido en el nombre de la tarea: “ACTION_INICIO”, este action suele ser utilizado para la precarga de variables, y debe extender de la clase FormulaAction.

```
public class FormulaPruebaAction extends FormulaAction
```

Además deberá implementarse el método abstracto `public abstract Map <String,String> establecerVariables (List<VariableFormularioDTO> variables).`

En este método se recibe una lista con las variables del formulario, y se cargarán con el valor deseado, devolviéndose en el mapa de retorno del método.

Una vez realizada la implementación del “ACTION_INICIO”, éste debe redireccionar al fichero “formula.jsp” (propio de PTw@ndA) que se encarga de mostrar el formulario creado en formul@.

III.Action de fin:

Tras la realización de la acción oportuna después de la manipulación del formulario, debe crearse otro action en el fichero de struts, con el nombre “ACTION_FIN” en el que se enlaza con el método que

quiera ejecutarse tras el uso del formulario. Para ello debe extenderse en la clase que implemente el Action de fin de la clase FormulaAction, implementándose el método abstracto `public abstract void guardaVariables (Map <String,String> variables).`

En este método se recibe un mapa con los nombres y valores de las variables, de forma que puedan almacenarse o tratarse de forma específica, implementándolo el desarrollador.

9.8 Uso de Proces@ para tareas de manipulación de datos

A partir de la versión 2.1 de PTw@ndA es posible utilizar la herramienta Proces@ a la hora de implementar las diferentes tareas de manipulación de datos. Esta herramienta nos permite que sistemas cliente, en este caso PTw@ndA, puedan acceder y ejecutar scripts groovy con la finalidad de cargar datos iniciales del formulario de una BBDD propia. Proces@ está integrado a su vez con la herramienta Formul@, por lo cual tenemos acceso a toda la funcionalidad del motor de formularios trabajando con el cliente de proces@.

Los pasos necesarios a realizar para la utilización de procesa son:

I.Nomenclatura de la tarea de manipulación de datos:

Para la utilización de proces@ como generador de formularios es indispensable definir en el modelado del procedimiento el nombre de la tarea que hará uso del formulario creado en formula y del código de procesa asociado al formulario anterior, con la siguiente nomenclatura:

:PROC:NAMESPACE_STRUTS:CODIGO_PROCESA

Al detectar esta nomenclatura en la tarea, PTw@ndA interpreta que la tarea de manipulación de datos no está definida mediante una JSP o action de struts, sino que se ha creado mediante el motor de formularios formul@, teniendo un código de procesa asociado, el definido por el parámetro CODIGO_PROCESA.

El parámetro NAMESPACE_STRUTS contiene el nombre del namespace donde se definen los actions de inicio y de fin asociados al formulario. Estos action tendrán siempre el mismo nombre.

II.Definición de actions de inicio y fin:

Cada tarea que queramos realizar con procesa deberá tener asociado un package de struts con un namespace único en toda la aplicación. Concretamente el nombre que se le dé al namespace deberá coincidir con el nombre indicado en el modelado de la tarea.

```
<package name="procesaSUBADM" namespace="/procesaSUBADM" extends="jsf-default">

    <action name="inicioProcesa" method="inicio" class="procesaSUBADMAction">
        <result name="success" type="jsf">/procesa/procesa.jsp</result>
        <interceptor-ref name="jsfStack"/>
    </action>

    <action name="finProcesa" method="finProcesa" class="procesaSUBADMAction">
        <result name="success" type="redirect-action">
            <param name="actionName">inicioProcesa</param>
            <param name="parse">true</param>
            <param name="namespace">/procesaSUBADM</param>
            <param name="guardado">true</param>
        </result>
        <interceptor-ref name="basicStack" />
    </action>
</package>
```

En la imagen vemos un ejemplo de definición de package de struts. Los atributos que deben de variar de una tarea a otra serán el nombre y el namespace del package, así como la clase de ambos actions. El resto de valores como son el nombre de los actions de inicio y fin deben mantener la misma nomenclatura expuesta en la imagen.

III.Definición del bean de la tarea:

Dado que la implementación de Proces@ está basada en JSF será necesario especificar 2 beans asociados al action que implemente los métodos establecerVariables y guardarVariables: un bean para Spring y otro bean para JSF. En el caso del bean de spring indicaremos en el correspondiente applicationContext.xml la siguiente sentencia:

```
<bean id="procesaSUBADMAction" class="com.everis.subadm.procesa.ProcesaSUBADMAction" scope="prototype" parent="procesaTareaAction">
</bean>
```

Es importante que se indique el padre del action que en este caso será el bean **procesaTareaAction.**

Para el bean JSF será necesario crear un fichero faces-config.xml. Este fichero debe ubicarse en un directorio META_INF dentro del jar que contenga la implementación del action. Dentro del xml definiremos el bean de jsf:

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN" "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>

    <managed-bean>
        <description>Subvenciones ADM</description>
        <managed-bean-name>procesaSUBADMAction</managed-bean-name>
        <managed-bean-class>com.everis.subadm.procesa.ProcesaSUBADMAction</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
</faces-config>
```

IV.Implementación de métodos establecerVariables y guardarVariables:

El último paso de la integración es la creación del action que implemente los métodos establecerVariables y guardarVariables. El método establecerVariables se invocará durante la ejecución del action de inicio. De

este modo se podrá recuperar el valor de cada variable previo a su visualización. El método guardarVariables será invocado una vez pulsado el botón de guardar y terminar asociado al formulario en cuestión.

Un ejemplo de action sería el siguiente:

```
package com.everis.subadm.procesa;

import java.io.Serializable;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import es.juntadeandalucia.motorformularios.cliente.vo.VariableFormularioDTO;
import es.juntadeandalucia.plataforma.actions.procesa.IProcesaOperacionesVariables;
import es.juntadeandalucia.plataforma.actions.procesa.ProcesaTareaAction;


public class ProcesaSUBADMAction extends ProcesaTareaAction implements IProcesaOperacionesVariables,Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1232400060359288347L;

    public ProcesaSUBADMAction () {
    }

    public  Map <String,String> establecerVariables (List<VariableFormularioDTO> variables)
    {
        //IMPLEMENTAR
        return mapaVariables;
    }


    public void guardaVariables (Map <String,String> variables){
        //IMPLEMENTAR
    }

}
```

El action deberá extender de **ProcesaTareaAction** e implementar la interface **IProcesaOperacionesVariables**, la cual contiene el prototipo de ambas funciones a implementar en la clase.

9.9 Uso de Proces@ en altas específicas de expedientes

A partir de la versión 2.1 de PTw@ndA se incorpora la posibilidad de dar de alta datos específicos de un expediente en el propio módulo de alta de expediente que implementa la plataforma por defecto.

Una vez realizado el alta genérica se habilitará un listado de formularios de formula para su edición o detalle. Cada formulario puede ser implementado de la misma manera que se ha expuesto en el punto 9.9, con una serie de modificaciones que vemos a continuación:

I.Nomenclatura del formulario:

Una vez pulsado el enlace edición o detalle de cada formulario, la plataforma debe ser capaz de invocar al action de inicio del formulario seleccionado, conociendo de antemano el nombre del namespace donde se ubica el action de inicio. La información necesaria, al igual que en las tareas de manipulación de datos, son el namespace de struts y el código de procesa asociado al formulario del motor de formularios.

Esta información se configura desde la herramienta de administración de PTw@ndA. Desde el apartado *Mantenimiento de datos del expediente* se podrán asociar formularios a los procedimientos vigentes del motor de tramitación, indicando por cada uno de ellos el namespace y el código de procesa.

II.Definición de actions de inicio y fin:

Cada formulario deberá tener asociado un package de struts con un namespace único en toda la aplicación. Concretamente el nombre que se le dé al namespace deberá coincidir con el nombre indicado en la zona de administración.

```
<package name="procesa" namespace="/procesaDatosExpediente" extends="migas,TokenValidator,jsf-default,struts-default">

    <action name="inicioProcesa" method="inicio" class="gestionDatosFormulaAction">
        <result name="success" type="jsf">/procesaAlta/procesaAlta.jsp</result>
        <result name="externo" type="redirect-action">
            <param name="actionName">inicioProcesaAltaExterno</param>
            <param name="parse">true</param>
            <param name="namespace">/procesaAlta</param>
        </result>
        <interceptor-ref name="jsfStack"/>
    </action>

    <action name="finProcesa" method="finProcesa" class="gestionDatosFormulaAction">
        <result name="success" type="redirect-action">
            <param name="actionName">inicioProcesa</param>
            <param name="parse">true</param>
            <param name="namespace">/procesaDatosExpediente</param>
            <param name="guardado">true</param>
        </result>
        <interceptor-ref name="jsfStack" />
    </action>

</package>
```

En la imagen vemos un ejemplo de definición de package de struts. Los atributos que deben de variar de un formulario a otro serán el nombre y el namespace del package, así como la clase de ambos actions. El resto de valores como son el nombre de los actions de inicio y fin deben mantener la misma nomenclatura expuesta en la imagen.

III.Definición del bean de la tarea:

Dado que la implementación de Proces@ está basada en JSF será necesario especificar 2 beans asociados al action que implemente los métodos establecerVariables y guardarVariables: un bean para Spring y otro bean para JSF. En el caso del bean de spring indicaremos en el correspondiente applicationContext.xml la siguiente sentencia:

```
<bean id="gestionDatosFormulaAction" class="com.everis.plataforma.gestionDatos.action.GestionDatosFormulaAction" scope="prototype" parent="procesaAltaAction">
</bean>
```

Es importante que se indique el padre del action que en este caso será el bean **procesaAltaAction.**

Para el bean JSF será necesario crear un fichero faces-config.xml. Este fichero debe ubicarse en un directorio META_INF dentro del jar que contenga la implementación del action. Dentro del xml definiremos el bean de jsf:

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN" "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>

    <managed-bean>
        <description>Gestión de datos del expediente</description>
        <managed-bean-name>gestionDatosFormulaAction</managed-bean-name>
        <managed-bean-class>com.everis.plataforma.gestionDatos.action.GestionDatosFormulaAction</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>

</faces-config>
```

IV.Implementación de métodos establecerVariables y guardarVariables:

El último paso de la integración es la creación del action que implemente los métodos establecerVariables y guardarVariables. El método establecerVariables se invocará durante la ejecución del action de inicio. De este modo se podrá recuperar el valor de cada variable previo a su visualización. El método guardarVariables será invocado una vez pulsado el botón de guardar y terminar asociado al formulario en cuestión.

```java
package com.everis.plataforma.gestionDatos.action;

import java.io.Serializable;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import es.juntadeandalucia.motorformularios.cliente.vo.VariableFormularioDTO;
import es.juntadeandalucia.plataforma.actions.procesa.IProcesaOperacionesVariables;
import es.juntadeandalucia.plataforma.actions.procesa.ProcesaAltaAction;

/**
 * Clase que implementa los métodos establecerVariables y guardarVariables, que
 * serán invocados al cargar el formulario y al guardarlo, respectivamente.
 *
 * @author everis
 *
 */
public class GestionDatosFormulaAction extends ProcesaAltaAction implements IProcesaOperacionesVariables, Serializable {

    /** The Constant serialVersionUID. */
    private static final long serialVersionUID = 6535574841718381197L;

    /**
     * Instantiates a new gestion datos formula action.
     */
    public GestionDatosFormulaAction(){
    }

    /**
     * Método que carga las variables del formulario de formula asociado al
     * expediente.
     *
     * @param variables
     *            variables del formulario
     * @return mapa con los pares casilla-valor
     */
    public Map<String, String> establecerVariables(List<VariableFormularioDTO> variables) {
        Map<String, String> mapaVariables = new LinkedHashMap<String, String>();


        return mapaVariables;
    }

    /**
     * Método que guarda los valores introducidos en el formulario de formula.
     *
     * @param variables
     *            variables del formulario
     */
    public void guardaVariables(Map<String, String> variables) {


    }

}
```

El action deberá extender de **ProcesaAltaAction** e implementar la interface **IProcesaOperacionesVariables**, la cual contiene el prototipo de ambas funciones a implementar en la clase.

9.10.....Uso de numerador

La Plataforma de Tramitación permite la implementación y configuración de un componente numerador que se encargue de generar de forma automática el número de expediente al darlo de alta.

Si queremos realizar la implementación de un numerador específico, este deberá implementar la interfaz **INumerador**, con sus dos métodos correspondientes. El método obtenerNumeroExpediente deberá devolver el número del expediente generado.

```java
package es.juntadeandalucia.plataforma.numeradorPT;
```

```
public class NumeradorPT implements INumerador {

        public String obtenerNumeroExpediente(String refProcedimiento) throws BusinessException {

                ....

        }

        public String obtenerNumeroCIWA(String refInteresado) throws BusinessException {

                ...

        }

}
```

Para favorecer la adecuación al formato de identificador de Expediente Electrónico exigido por la NTI, http://www.boe.es/diario_boe/txt.php?id=BOE-A-2011-13170 , Anexo I del PDF de la disposición, es necesario generar un número de expediente con longitud de 30 caracteres.

El número de expediente del numerador no será el número final del expediente en Trew@, sino una parte de él. El resto del identificador asignado al expediente “ES + DIRCOMUN” vendrá establecido por Trew@ automáticamente.

En Trew@ debemos crear un nuevo componente de tipo numerador especificando nombre, descripción y organismo. En los datos del componente creamos dos atributos, PARAM_PAQUETE y PARAM_CLASE, cuyos valores serán el paquete y nombre de la clase del numerador.

Por último, creamos un parámetro de configuración en plataforma denominado nombre_componente_numera cuyo valor sea el nombre que hemos asignado al componente numerador en Trew@.

9.11.....Módulo de tipo web Service

A partir de la versión 2.1.1 de PTw@ndA se incorpora la posibilidad de dar de alta módulos de tipo web service. Mediante este nuevo tipo de módulo funcional se podrá desplegar un catálogo de servicios web desarrollados con Spring.

Para el desarrollo de un nuevo módulo de tipo Web Service y que este se despliegue correctamente dentro del catálogo de servicios de la Plataforma de Tramitación, será necesario crear y/o modificar los siguientes ficheros:

En el fichero **despliegue.xml** debemos especificar que el tipo de instalación es WS como se muestra en el siguiente ejemplo:

```
<def-modulo>
    <nombre>moduloEjemploWS</nombre>
    <version>1.0</version>
    <titulo>Servicio web </titulo>
    <descripcion>Servicio web</descripcion>
    <type>STRUTS-2</type>
```

```
        <tipoInstalacion>WS</tipoInstalacion>
    </def-modulo>
```

El fichero de configuración de Spring deberá ser renombrado como **applicationContext-servlet.xml.** Este fichero deberá contener la definición del bean que generará el fichero descriptor del servicio web (wsdl). Es importante destacar que el servicio web se publicará siempre a partir de la ruta /WebServices/… A continuación se muestra un ejemplo de definición del vean:

```
<!-- Generacion automatica del WSDL -->
<bean id="webservice-entrega" class="org.springframework.ws.wsdl.wsdl11.DynamicWsdl11Definition">
  <property name="builder">
    <bean class="org.springframework.ws.wsdl.wsdl11.builder.XsdBasedSoap11Wsdl4jDefinitionBuilder">
      <property name="schema" value="/modulos/pruebaWS/xsd/entrega.xsd"/>
      <property name="portTypeName" value="ENTREGATramitacion"/>
      <property name="locationUri" value="http://localhost:8080/ptwanda-web-2.1.0/WebServices/entregaWS"/>
    </bean>
  </property>
</bean>
```

El fichero xsd a partir del cual se generará el wsdl correspondiente deberá ubicarse dentro de la carpeta webapp del módulo.

9.12.....Módulo de tipo administración

A partir de la versión 2.1.1 de PTw@ndA se incorpora la posibilidad de dar de alta módulos de tipo administración. Para el desarrollo de un nuevo módulo de tipo administración y que este sea visible desde la herramienta de administración de la Plataforma de Tramitación, será necesario crear y/o modificar el fichero despliegue.xml indicando el tipo adecuado.

```
    <def-modulo>
        <nombre>moduloEjemploAdministracion</nombre>
        <version>1.0</version>
        <titulo>Servicio web </titulo>
        <descripcion>Administracion</descripcion>
        <type>STRUTS-2</type>
        <tipoInstalacion>ADMINISTRACION</tipoInstalacion>

    </def-modulo>
```

Una vez instalado el módulo será necesario configurar la entrada de menú en la administración de PTw@nda, como se explica detalladamente en el apartado *4.1 Modificación de menús* del manual de administración.

9.13.....Módulo de tipo utilidad masiva

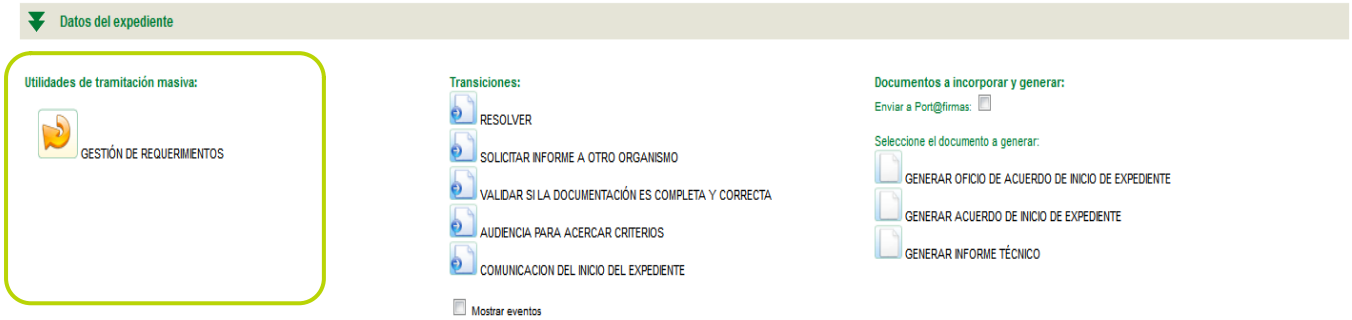
A partir de la versión 2.2.0 de PTw@ndA se incorpora la posibilidad de dar de alta módulos de tipo utilidad masiva. Para el desarrollo de un nuevo módulo de tipo utilidad masiva, será necesario crear y/o modificar el fichero despliegue.xml indicando el tipo adecuado:

```
    <def-modulo>
        <nombre>moduloEjemploUtilidadMasiva</nombre>
        <version>1.0</version>
        <titulo>Utilidad masiva de requerimientos</titulo>
        <descripcion>Utilidad masiva de requerimientos</descripcion>
```

```
        <type>STRUTS-2</type>
        <tipoInstalacion>UTILIDADMASIVA</tipoInstalacion>
    </def-modulo>
```

Los módulos de tipo utilidad masiva tienen como objetivo realizar operaciones sobre un conjunto de expedientes. El acceso a este tipo de módulo estará localizado en la zona de detalle del expediente, en el módulo funcional de búsqueda de expedientes.

Una vez realizada una búsqueda por procedimiento y fase, se podrán seleccionar todos los expedientes para realizar operaciones masivas, habilitándose las utilidades masivas configuradas para el procedimiento y la fase seleccionadas (una vez instalado el módulo de tipo utilidad masiva será necesario realizar la configuración de visibilidad del mismo).



La plataforma invocará el action de inicio del módulo de tipo utilidad masiva pasando las referencias de los expedientes seleccionados sobre los cuales se realizará la operación masiva. A continuación se muestra un ejemplo de un action de inicio de un módulo de utilidad masiva, donde se aprecia cómo recuperar las referencias de los expedientes:

```
package es.juntadeandalucia.ptwanda;

import java.util.Map;
import org.apache.struts2.interceptor.ServletRequestAware;
import org.apache.struts2.interceptor.SessionAware;

public class UtilidadMasivaAction implements SessionAware {


    /** Variable que contiene los identificadores de expedientes seleccionados*/
    private String listaExpedientesSel;

    /** The session. */
    private Map session;


    public String inicio() {
        listaExpedientesSel = (String) session.get("listaExpedientesSel");
        return "success";
    }


    public String getListaExpedientesSel() {
        return listaExpedientesSel;
    }


    public void setListaExpedientesSel(String listaExpedientesSel) {
        this.listaExpedientesSel = listaExpedientesSel;
    }

    @Override
    public void setSession(Map session) {
        // TODO Auto-generated method stub
        this.session = session;
    }
}
```

9.14.....Uso de firmantes en variables

A la hora de implementar variables para una plantilla puede resultar útil recuperar el nombre del firmante asociado a la hora de generar el documento en cuestión. Esta información se encuentra almacenada en la plataforma de tramitación.

A continuación se muestra una porción del código que recupera el nombre del firmante principal de un documento:

```
cw = new ClausulaWhere();
cw.addExpresion(TrDocumentoExpediente.CAMPO_REFTIPODOC, OperadorWhere.OP_IGUAL, P_TIPO_DOC.toString());
cw.addExpresion(TrDocumentoExpediente.CAMPO_ESTADO, OperadorWhere.OP_IGUAL, "R");
docs = getApiUI().obtenerDocumentosExpediente(new TpoPK(P_EXP), false, cw, null);
```

```
if (docs != null && docs.length == 1) {
        firmantes = gestionFirmaService.obtenerFirmantesDocumentoPT(docs[0].getREFDOCEXP().toString());
            if (firmantes != null && firmantes.size() > 0) {
              usuario = gestionUsuarioService.obtenerUsuario(firmantes.get(0).getUsuarioFirma());
.....
..... ....
            } else {
                ....
            }
        ....
}
```

9.15..... Configuración servidor de correo

En versión 2.1.1 se incorpora la funcionalidad de envío de correos electrónicos para notificar avisos generados durante la tramitación del expediente. Para poder enviar correos se hace necesario la configuración de un servidor de correos. Este servidor será configurado a nivel de servidor de aplicaciones. Para ello será necesario crear un nuevo mbean en el fichero mail-service.xml, ubicado en el directorio deploy.

En este fichero indicaremos los diferentes atributos y parámetros del servidor de correos, como el usuario, la contraseña o dirección ip.

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

  <!-- ===================================================================== -->
  <!-- Mail Connection Factory                                               -->
  <!-- ===================================================================== -->

  <mbean code="org.jboss.mail.MailService"
         name="jboss:service=Mail">
    <attribute name="JNDIName">java:/Mail</attribute>
    <attribute name="User">nobody</attribute>
    <attribute name="Password">password</attribute>
    <attribute name="Configuration">
      <!-- A test configuration -->
      <configuration>
        <!-- Change to your mail server prototocol -->
        <property name="mail.store.protocol" value="pop3"/>
        <property name="mail.transport.protocol" value="smtp"/>

        <!-- Change to the user who will receive mail  -->
        <property name="mail.user" value="nobody"/>

        <!-- Change to the mail server  -->
        <property name="mail.pop3.host" value="pop3.nosuchhost.nosuchdomain.com"/>

        <!-- Change to the SMTP gateway server -->
        <property name="mail.smtp.host" value="smtp.nosuchhost.nosuchdomain.com"/>

        <!-- The mail server port -->
        <property name="mail.smtp.port" value="25"/>

        <!-- Change to the address mail will be from  -->
        <property name="mail.from" value="nobody@nosuchhost.nosuchdomain.com"/>

        <!-- Enable debugging output from the javamail classes -->
        <property name="mail.debug" value="false"/>
      </configuration>
    </attribute>
    <depends>jboss:service=Naming</depends>
  </mbean>

</server>
```

Por último, creamos un parámetro de configuración en plataforma denominado JNDI_SERVIDOR_CORREO cuyo valor sea el nombre que hemos asignado al atributo JNDIName del fichero mail-service.xml.

9.16.....Descartar y eliminar tareas

En la Plataforma de Tramitación existe la opción de descartar y eliminar las tareas que han sido previamente creadas. Desde la versión 2.1.1 se proporciona un mecanismo que permite eliminar los datos relativos a la tarea que se quiere descartar o eliminar, dejando el proceso de tramitación como se encontraba antes de la realización de la tarea.

Este mecanismo se puede utilizar para eliminar tanto tareas web, como tareas de procesa o formula. A continuación se detalla el funcionamiento para cada uno de los casos.

9.16.1 Eliminar tarea web

El mecanismo de borrado de tareas invocará el action con nombre: “eliminar” concatenado con el nombre de la tarea en Trew@. Para ello definimos el siguiente action en el struts de tareas:

```
<action name="eliminarNOMBRETAREA" method="eliminar" class="Clase">
    <result name="success">/modulos/tareasAsociadasExpediente/vacia.jsp</result>
    <result name="error">/modulos/tareasAsociadasExpediente/errorEliminarTarea.jsp</result>
    <interceptor-ref name="basicStack" />
</action>
```

Implementamos el método que se encargue de borrar los datos en su clase correspondiente. Obligatoriamente el proceso de borrado de datos debe realizarse dentro de un bloque try/catch para que en caso de que se produzca alguna excepción durante el borrado, se capture la excepción y se mantenga la tarea en Trew@ para evitar inconsistencia de datos.

```
public String eliminar(){
    try{
            ...
    catch (Exception e) {
        e.printStackTrace();
        return "error";
    }
      return "success";
}
```

9.16.2 Eliminar tarea de procesa

Para el caso de las tareas de procesa el action que se invoca para eliminar la tarea es: eliminarDatosProcesa, por lo que debemos añadir en el struts de tareas lo siguiente:

```
<action name="eliminarDatosProcesa" method="eliminarDatos" class="Clase">
    <result name="success">/modulos/tareasAsociadasExpediente/vacia.jsp</result>
    <result name="error">/modulos/tareasAsociadasExpediente/errorEliminarTarea.jsp</result>
    <interceptor-ref name="basicStack" />
</action>
```

Del mismo modo que para las tareas web, implementamos el método que se encargará de borrar los datos, encerrando el código dentro de un bloque try/catch.

```
public String eliminarDatos ( String nombreTarea, String identificadorTarea) {
    try{
        ….
```

```
    }catch (Exception e) {
        e.printStackTrace();
        return "error";
    }
    return "success";
```

9.16.3 Eliminar tarea de formula

Si la tarea es de formula el action que se invoca es: “eliminar” concatenado con el nombre del action de inicio que se haya configurado en trewa. El action debe incluirse en el struts de tareas

```
<action name="eliminarACTION_INICIO" method="eliminarDatos" class="Clase">
    <result name="success">/modulos/tareasAsociadasExpediente/vacia.jsp</result>
    <result name="error">/modulos/tareasAsociadasExpediente/errorEliminarTarea.jsp</result>
    <interceptor-ref name="basicStack" />
</action>
```

Igual que para las tareas web y las de procesa implementamos el método que se encargará de borrar los datos, encerrando el código dentro de un bloque try/catch.

```
public String eliminarDatos () {
    try{
        ...
    }catch (Exception e) {
        e.printStackTrace();
        return "error";
    }
    return "success";
}
```

9.17 Obtención de los datos de consulta de los servicios SCSP

El módulo SCSP proporciona un servicio para la obtención de los datos de consulta de los servicios SCSP para un interesado en cuestión. El servicio devuelve un listado con cada petición realizada a cada certificado solicitado.

Un ejemplo de invocación al servicio sería el siguiente:

.....

```
WebApplicationContext wac = ContextLoader.getCurrentWebApplicationContext();

ICoreScspService coreService = (ICoreScspService) wac.getBean("coreScspServiceImpl");

List<PeticionType> listaPeticiones = coreService. obtenerDatosConsultasSCSPInteresado( refExpediente, refInteresado);

……
```

9.18..... Esquema de indexación dinámico

Desde la versión 2.1.1 de la Plataforma de Tramitación se permite la posibilidad de indexar campos específicos en el esquema de indexación de Solr, manteniendo siempre por defecto el esquema proporcionado por la plataforma.

El esquema de indexación se ha modificado para contemplar la indexación de campos definidos de forma dinámica. Se han incluído los siguientes campos:

```
<dynamicField name="*_i" type="int" indexed="true" multiValued="true" stored="true" />
<dynamicField name="*_s" type="string" indexed="true" multiValued="true" stored="true" />
<dynamicField name="*_l" type="long" indexed="true" multiValued="true" stored="true" />
<dynamicField name="*_t" type="text" indexed="true"  multiValued="true" stored="true" />
<dynamicField name="*_es" type="text_es" indexed="true"  multiValued="true" stored="true" />
<dynamicField name="*_b" type="boolean" indexed="true" multiValued="true" stored="true" />
<dynamicField name="*_f" type="float" indexed="true" multiValued="true" stored="true" />
<dynamicField name="*_d" type="double" indexed="true" multiValued="true" stored="true" />
<dynamicField name="*_dt" type="date" indexed="true" multiValued="true" stored="true" />
```

Para hacer uso de la funcionalidad de indexación dinámica es necesario la creación de una clase que será la encargada de establecer los nuevos campos a indexar junto con sus valores asociados. Esta clase deberá implementar la interface *IExtractorInformacion*.

Una vez creada la clase será necesario configurar el parámetro de configuración **IMPLEMENTACION_EXTRACTOR_SOLR**. Este parámetro debe contener la ruta completa de la clase creada, incluido el package, por ejemplo, 'es.juntadeandalucia.plataforma.busqueda.ClaseIndexador'.

Es importante destacar que la nueva clase no deberá contener los campos de indexación por defecto ya que la plataforma seguirá gestionando el esquema inicial con independencia de tener una clase extractor específica.

A continuación se muestra un ejemplo de una clase de prueba que realiza la indexación de campos específicos:

```
package es.juntadeandalucia.plataforma.busqueda;

import java.util.HashMap;
import java.util.Map;
import es.juntadeandalucia.plataforma.comunes.excepciones.ArchitectureException;
import es.juntadeandalucia.plataforma.comunes.excepciones.BusinessException;
import es.juntadeandalucia.plataforma.service.busqueda.IExtractorInformacion;
import es.juntadeandalucia.plataforma.service.expediente.IExpediente;

public class ClaseIndexador implements IExtractorInformacion {
```

```
    @Override
    public Map<String, Object> extraerInformacion(IExpediente expediente,
            String path) throws BusinessException, ArchitectureException {

        Map<String, Object> mapaIndexador = new HashMap< String, Object>();

        mapaIndexador.put("interesados_s", "00000000T");
        mapaIndexador.put("precio_d", "1045.45");

        return mapaIndexador;
    }

}
```

9.19 Recargas AJAX en Internet Explorer

A partir de la versión 2.2.0 de PTw@ndA, se ha detectado un bug del navegador Internet Explorer relacionado con las recargas AJAX realizadas haciendo uso de Struts2.

La etiqueta siguiente declara un formulario oculto cuya respuesta es inyectada en un div con identificador “divFasesGenericos”:

```
<s:form      theme="ajax"      action="/busqueda/actualizarFases.action"      cssStyle="display:none"
id="formAjaxFases" method="post">
        <s:hidden id="procedimientoGenerico" name="procedimiento"/>
        <s:submit id="submitActualizarFasesGenerico" cssStyle="display:none" theme="ajax"
            targets="divFasesGenerico" loadingText="Cargando Fases..."/>
</s:form>
```

Al ejecutar el submit asociado al formulario en Internet Explorer, el resultado del formulario no es inyectado en el div indicado en el atributo **targets**, sino que se carga el contenido en una página nueva.

Para realizar la inyección en el div es necesario declarar el atributo **type** de la etiqueta **s:submit**, estableciendo su valor a **button**.

```
<s:form      theme="ajax"      action="/busqueda/actualizarFases.action"      cssStyle="display:none"
id="formAjaxFases" method="post">
        <s:hidden id="procedimientoGenerico" name="procedimiento"/>
        <s:submit type="button" id="submitActualizarFasesGenerico" cssStyle="display:none"
            theme="ajax" targets="divFasesGenerico" loadingText="Cargando Fases..."/>
</s:form>
```

9.20 Gestión de transaccionalidad para el API de Trew@

La gestión del API de Trew@ está centralizada en el servicio IConfigurableService, servicio padre de todos los servicios de tramitación que hacen uso del api para operar con Trew@.

El comportamiento por defecto de la plataforma es operar con el api, estableciendo el valor de la varianle **autocommit** a true. Este comportamiento implica que después de cualquier operación de insert/update/delete realizada por el api, automáticamente ejecutará un commit sobre la base de datos, guardando los últimos cambios.

Desde la versión 2.2.0 de PTw@ndA es posible modificar el comportamiento del api, pudiendo establecer la variable autocommit con un valor false, delegándose la invocación a los métodos commit() y rollback() del api, pudiéndose deshacer los cambios realizados en el motor de tramitación en caso de fallo, manteniéndose en todo momento un estado consistente de la base de datos.

Se ha incluido en el servicio padre de la plataforma los siguientes métodos:

- **iniciarOperacionTransaccion():**

  Este método cierra la conexión de la api, en el caso que estuviera instanciada, y establece la variable autocommit a false. A la hora de instanciarse el api, nos preguntaremos por el valor de la variable autocommit, que por defecto será true.

  Es necesario invocar este método por cada servicio que se quiera utilizar.

- **confirmarOperacionTransaccion():**

  Realiza un commit para confirmar la operación realizada. Si el commit es ejecutado con éxito, se cierra la conexión del api y se reestablece el valor de autocommit a true, quedándose con el comportamiento por defecto.

  Del mismo modo que en el método anterior, será necesario invocar este método por cada servicio implicado en la transacción.

- **deshacerOperacionTransaccion():**

  Realiza un rollback para deshacer la operación. Si el rollback se ejecuta con éxito, se cierra la instancia del api y se restablece el valor de autocommit a true.

  Del mismo modo que en el método anterior, será necesario invocar este método por cada servicio implicado en la transacción.

Con estas modificaciones un posible ejemplo de transaccionalidad sería el siguiente, en el que vemos que se deben de aplicar los nuevos métodos del servicio padre tanto en los servicios propios de PTw@ndA como en los servicios de los módulos verticales.

```
try {

    // Iniciamos transaccion por cada servicio implicado
    //en la transacción
    tareasService.iniciarOperacionTransaccion();
    docService.iniciarOperacionTransaccion();
    tareaS.iniciarOperacionTransaccion();

    // Generamos documento
    IExpediente expedienteActual = (IExpediente) user
            .getExpediente();
    String refdoc = (String) session.get("referencia");
    List<IMensaje> listaAvisosD = new LinkedList<IMensaje>();
    IDocumento doc = null;

    // Mapa de parametros
    Map<String, String> parametros = new HashMap<String, String>();
    String refExpFase = user.getFaseActual().getRefFaseActual();
    String proc = user.getFaseActual().getRefDefProc();

    doc = docService.generarDocumento(expedienteActual, refdoc,
            parametros, user.getUsuario(), proc, "", listaAvisosD,
            refExpFase);

    // Finalizamos tarea
    Map<String, Object> mapaCriterios = new HashMap<String, Object>();
    mapaCriterios.put(TareaServiceImpl.BUSQUEDA_TAREA_EXP, "44966");

    tareaS.finalizarTareaExpedienteOptimo("44966", "M",
            user.getUsuario());

    //Accion propia del servicio del módulo vertical
    resultado = tareasService.insertarDato("fechaARNSubsana",
            fechaARNSubsana, nExp);

    // Confirmamos la operación para todos
    //los servicios
    tareasService.confirmarOperacionTransaccion();
    docService.confirmarOperacionTransaccion();
    tareaS.confirmarOperacionTransaccion();

} catch (Exception e) {
    // Deshacemos operación para todos los servicios
    tareasService.deshacerOperacionTransaccion();
    docService.deshacerOperacionTransaccion();
    tareaS.deshacerOperacionTransaccion();
}
```

9.20.1 Gestión Multi-Trew@

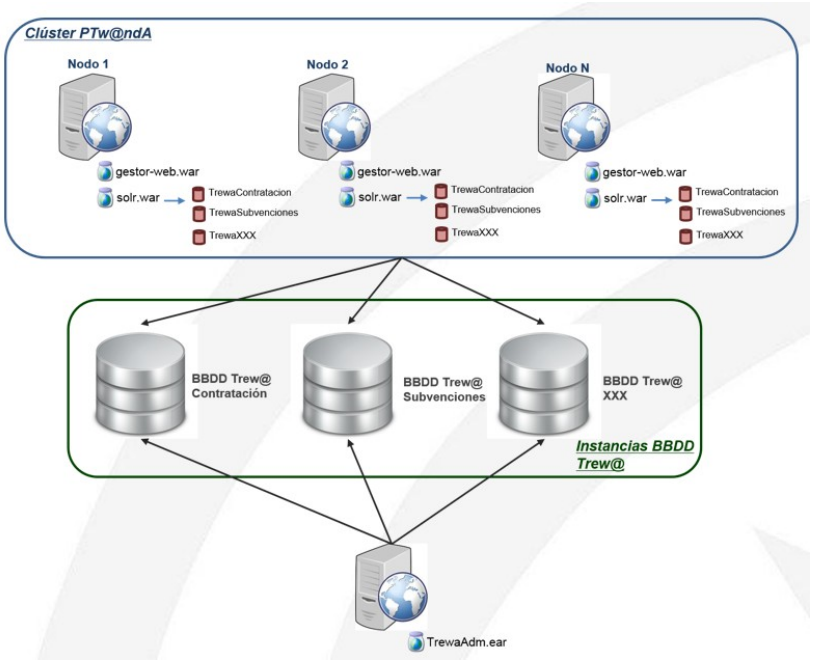
Desde la versión 2.4.0, se evolucionó la arquitectura de la Plataforma de Tramitación de forma que es posible la comunicación con más de una instancia del motor de tramitación Trew@. Cada instancia del motor de tramitación se traduce en un nombre JNDI único declarado en los ficheros de configuración del servidor de aplicaciones WildFly.

Esto supone importantes beneficios para los usuarios de la plataforma de tramitación ya que es posible desde la misma aplicación la creación, gestión y tramitación de expedientes que pertenezcan a distintos

procedimientos que, a su vez, se encuentran creados en distintas instancias de Trew@, pudiéndose superar el ámbito de negocio específico de los procedimientos tramitados por la plataforma.

Mediante la evolución a una arquitectura Multi-Trew@ se garantiza una mejor gestión de la carga de usuarios del gestor, al desacoplarse los procedimientos, derivándose a instancias de bases de datos diferentes en función de la familia de procedimientos en cuestión. Esto proporciona un aumento en la mantenibilidad del sistema, ya que se logra separar la información de negocio de las distintas familias de procedimiento, facilitando la gestión y actualización de la información.

A continuación, se muestra un diagrama con la infraestructura tecnológica de la funcionalidad Multi-Trew@:



Un punto a tener en cuenta es la gestión de la instancia de Trew@ a la que se hace referencia en tiempo de ejecución de un módulo funcional. Se muestra un ejemplo, a continuación, de cómo instanciar la adecuada API de Trew@ de manera correcta.

Una posibilidad es obtener el usuario en sesión y de él obtener tanto el JNDI como el Sistema; y de esta forma obtener la API de Trew@ para hacer uso de ella:

```
String jndi = user.getJndiTrewa();
String sistema = user.getSistema().getCodigo();
TrAPIUI apiUI = conexionesService.getObtenerApiUI(jndi, sistema);
```

9.21. Uso de variables en campo asunto de envío a Port@firmas

En la versión 2.3.0 de PTw@ndA se incorpora un nuevo parámetro de configuración denominado ASUNTO_PFIRMA, mediante el cuál se puede establecer el asunto enviado a Port@firmas cuando se realiza una petición de firma de documento.

El valor configurado al parámetro podrá contener definición de variables que serán sustituidas en el momento de mostrar al usuario tramitador el formulario de envío de documento a Port@firmas, debiéndose definir las variables con el formato $$Nombre_de_la_variable$$.

Cada variable definida en el texto configurado para el campo asunto deberá darse de alta en Trew@, asociando a cada variable una clase Java y un método de implementación. Será necesario definir la clase y el método de forma que dicho método devuelva la cadena de caracteres que sustituirá a esa variable.

La plataforma de tramitación w@ndA proporciona la implementación por defecto de un conjunto de variables que pone a disposición del administrador de la plataforma para su inclusión en el texto configurado para el campo asunto. A continuación se enumeran las variables implementadas, así como los datos necesarios de cada una para su alta en el motor de tramitación Trew@:

1. Número del expediente

- Nombre de la variable: EXPEDIENTE_PFIRMA
- Descripción de la variable: Número de expediente al cual pertenece el documento enviado a firmar
- Paquete: es.juntadeandalucia.plataforma.actions.VariablesPortafirmas
- Función: obtExpediente
- Parámetros:

Nombre	Descripción	Tipo	Tamaño	Orden
P_EXP	Número del expediente	Numérico	10	1

2. Fase del expediente

- Nombre de la variable: FASE_PFIRMA
- Descripción de la variable: Fase del expediente en la que se realiza el envío del documento
- Paquete: es.juntadeandalucia.plataforma.actions.VariablesPortafirmas
- Función: obtFase
- Parámetros:

Nombre	Descripción	Tipo	Tamaño	Orden
P_EXP	Número del expediente	Numérico	10	1
P_FASE	Identificador de la fase en la que se envía el documento	Numérico	10	2

3. Documento

- Nombre de la variable: DOCUMENTO_PFIRMA
- Descripción de la variable: Nombre del documento enviado a firmar
- Paquete: es.juntadeandalucia.plataforma.actions.VariablesPortafirmas

- Función: obtDocumento
- Parámetros:

Nombre	Descripción	Tipo	Tamaño	Orden
P_EXP	Número del expediente	Numérico	10	1
P_REF_DOC	Identificador del documento permitido	Numérico	10	2

4. Firmantes

- Nombre de la variable: FIRMANTES_PFIRMA
- Descripción de la variable: Firmantes asociados al documento
- Paquete: es.juntadeandalucia.plataforma.actions.VariablesPortafirmas
- Función: obtFirmantes
- Parámetros:

Nombre	Descripción	Tipo	Tamaño	Orden
P_EXP	Número del expediente	Numérico	10	1
P_REF_DOC	Identificador del documento permitido	Numérico	10	2

5. Interesados

- Nombre de la variable: INTERESADOS_PFIRMA
- Descripción de la variable: Interesados asociados al documento / expediente
- Paquete: es.juntadeandalucia.plataforma.actions.VariablesPortafirmas
- Función: obtInteresados
- Parámetros:

Nombre	Descripción	Tipo	Tamaño	Orden
P_EXP	Número del expediente	Numérico	10	1
P_REF_DOC	Identificador del documento permitido	Numérico	10	2

A continuación se muestra un ejemplo de texto configurable para el parámetro ASUNTO_PFIRMA:

“El documento de nombre $$DOCUMENTO_PFIRMA$$ del expediente $$EXPEDIENTE_PFIRMA$$ enviado a firmar en la fase $$FASE_PFIRMA$$ tiene como interesado/s a $$INTERESADOS_PFIRMA$$ y como firmante/s a $$FIRMANTES_PFIRMA$$.”.

10 ANEXO I: Adaptación de módulos anteriores a la versión 2.0 de plataforma

Cabe destacar que a partir de la versión 2.0 de PTw@ndA, se ha efectuado una migración de versión de struts2, pasando de la versión 2.0.9 a la 2.0.14.

Los módulos desarrollados sobre la versión 2.0.9 de Struts2 (PTw@ndA 1.0.3 o inferior) podían utilizar expresiones y lenguaje EL, mientras que a partir de la versión 2.010 de Struts2 (PTw@ndA 2.0.0 o superior) se usan OGNL en las JSP.

Por lo tanto deberá cambiarse los ficheros JSP de los módulos realizados anteriormente, de forma que los tags de Struts se adecuen a esta nueva nomenclatura.

Puede consultarse información al respecto en el siguiente enlace: http://issues.apache.org/struts/browse/WW-2107

Adicionalmente, es necesario que los Servicios desarrollados para los módulos funcionales, hereden directamente de un Servicio de PTw@ndA: `PTw@ndAServiceImpl` o `ConfiguracionTramitacionServiceImpl`.

Estos pasos están descritos en el apartado ServiciosServicios".

11 ANEXO II: Internacionalización de PTw@ndA

Entre los soportes que proporciona el framework Struts encontramos la internacionalización (i18n) a través de ficheros de recursos y Java Locales. Se pueden definir strings a visualizar en tu fichero de recursos, y luego ser usados en tus JSPs.

Para ello proporciona la opción de identificar un idioma por defecto para cada Action, o para cada package, así mismo como para cada aplicación. Al igual que ocurre con cualquier otro ámbito, la aplicación cambiará el idioma de su interfaz a partir de los especificado en la configuración del navegador.

En aquellos paquetes donde se quiera hacer uso de la internacionalización bastará con crear un archivo (package.properties) en el directorio donde se encuentren las clases de la aplicación (ej: d:\com\empresa\aplicacion) donde se contengan las claves y valores con el formato clave.subclave=texto de los textos que pertenezcan al idioma principal. Ejemplo:

```
...
titulo.aplicacion= Plataforma de Tramitación w@ndA
aplicacion.header=Bienvenido a la Plataforma de Tramitación w@ndA
...
```

Para cada idioma alternativo se creará un archivo nuevo que se llame igual pero que termine en "_xx.properties" siendo xx el código ISO de idioma. Ejemplo:

Para un cliente de habla alemana sería **package _de.properties**
Para un cliente de habla francesa sería **package _fr.properties**
Para un cliente de habla italiana sería **package _it.properties**
Para un cliente de habla española sería **package _es.properties**

Posteriormente en las JSP se hará referencia a las claves definidas en el properties package, tomando aquellos valores en función del idioma establecido por el usuario en el navegador.

12 REFERENCIAS

Referencia	Documento
[1]	Manual de Instalación de PTw@ndA
[2]	Manual de Administración de PTw@ndA
[3]	Norma Técnica de Interoperabilidad de Documento Electrónico https://www.boe.es/boe/dias/2011/07/30/pdfs/BOE-A-2011-13169.pdf
[4]	Norma Técnica de Interoperabilidad de Expediente Electrónico https://www.boe.es/boe/dias/2011/07/30/pdfs/BOE-A-2011-13170.pdf