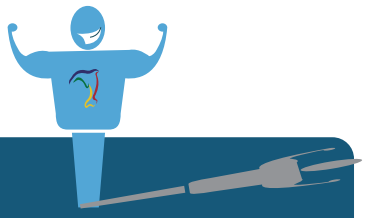
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

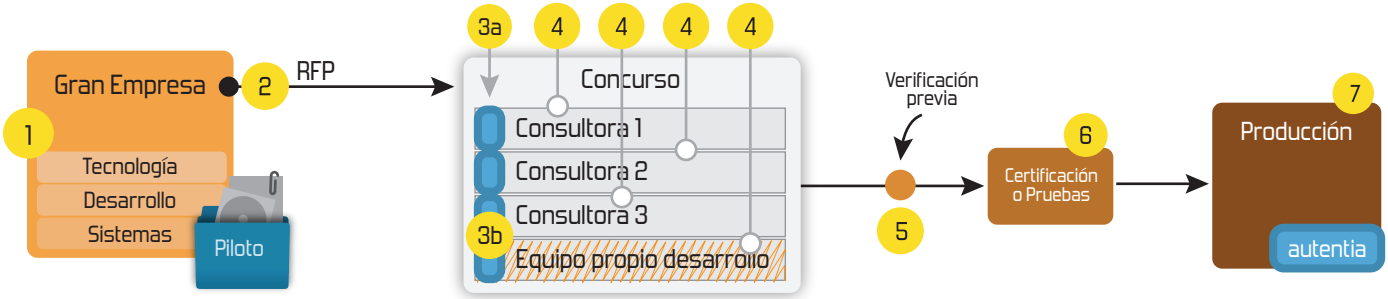
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Ultimas Noticias

» Autentia en JavaHispano
» Accesibilidad en entornos Web
» Liberada TNTConcept 0.16.1
» Cuarta charla Autentia + Agile Spain: Introducción a Scrum
» Historia de la Informática. Capítulo 40 - 1953
» ¡Adictos Renovado!
» Una historia de guerra Ágil: SCRUM Y XP DESDE LAS TRINCHERAS, Cómo hacemos Scrum
» Comentarios sobre Wikinomics de Don Tapscott
» Gestión de Repositorios Maven
» Valoración de tutoriales
» Empezamos nueva aventura: Un libro ...

+Noticias Destacadas

» Autentia en JavaHispano
» Accesibilidad en entornos Web
» Liberada TNTConcept 0.16.1
» Cuarta charla Autentia + Agile Spain: Introducción a Scrum
» Nueva sección de libros y El modelo Google ...
» Comparador de sueldos en la profesión Informática
» Empezamos nueva aventura: Un libro ...
» Si se pregunta ¿Qué ofrece este Web?
» Grupo XING
» +7,5 Millones de visualizaciones de nuestros Tutoriales

+Comentarios Cómic

+Enlaces

Últimos tutoriales

2009-04-15
Iniciación a OSWorkflow con Spring

2009-04-14
Tests de Selenium con librerías de componentes JSF: Apache Tomahawk.

2009-04-13
JTAPI. El API de Telefonía para Java

2009-04-13
Registro de Web Services con Apache jUDDI. Configuración y ejemplo

2009-04-13
Cómo hacer UML con Eclipse y el plugin UML2

2009-04-09
Spring WS: Servicios Web a través del correo electrónico

2009-04-02
Creación de cursos con Moodle

2009-03-31
Integrar Liferay Portal 5.2.1 con Pentaho BI 2.0.0 sobre MySQL 5.1

2009-03-31
Spring WS: Construcción de Clientes de Servicios Web con Spring

2009-03-30
Administración de sitios Moodle

2009-03-29
Empaquetamiento de aplicaciones de escritorio (standalone) con Maven

2009-03-27
Primeros pasos con Moodle

2009-03-26
Introducción a JSF Java

2009-03-25
A1 Website Analyzer

Tutorial desarrollado por

Borja Lázaro de Rafael

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática

Puedes encontrarme en Autentia

Somos expertos en Java/J2EE

Descargar este documento en formato PDF: osworkflowSpring.pdf

Fecha de creación del tutorial: 2009-04-15

Iniciación a OSWorkflow con Spring

Índice de contenido

Introducción.

En prácticamente todos los proyectos siempre nos encontramos con alguna entidad que pasa por una serie de estados, y dependiendo del estado en que se encuentre será posible realizar una serie de acciones u otras. Normalmente esto lo modelamos con una máquina de estados, obteniendo una representación del workflow para dicha entidad. Una vez tenemos la máquina de estados es común que sea el programador el que tenga que ir preguntando en qué estado se encuentra, controlando de esta forma las acciones que ofrece, y siendo también el responsable de cambiar de estado; lo que puede provocar en casos olvidados no ofreciendo todas las acciones o no actualizando el estado de la entidad al ejecutar alguna acción.

Esta responsabilidad la podemos delegar en los motores de workflow, que básicamente son los encargados de controlar las transiciones y cambios de estado de una máquina de estados. De esta forma, una vez tengamos definida nuestra máquina de estados sería conveniente delegar la responsabilidad del workflow a uno de estos motores sin necesidad de tener que ir controlando todo desde nuestra aplicación. En este tutorial vamos a presentar uno de estos motores OSWorkflow y su integración con Spring.

Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portatil Samsung R70 (Intel(R) Core(TM)2 Duo 2,5Ghz, 2046 MB RAM, 232 Gb HD)
- Sistema Operativo: Windows Vista Home Premium
- Máquina Virtual Java: JDK 1.5.0_14 de Sun Microsystems (http://java.sun.com/javase/downloads/index_jdk5.jsp)
- IDE Eclipse 3.3 (Europa) (http://www.eclipse.org/downloads/)

Dependencias.

Para utilizar OSWorkflow y Spring en nuestro proyecto debemos tener registradas todas las dependencias necesarias; ya sabéis que crear un proyecto de Maven nos ayuda a gestionar todas estas dependencias de una forma sencilla; por lo que simplemente configurando el fichero "pom.xml" ya estaremos listos para empezar a trabajar.

view plain print ?

```xml
<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>workflow</groupId>
    <artifactId>workflow</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>workflow</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <!-- junit -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.4</version>
            <scope>test</scope>
        </dependency>

        <!-- librer&iacute;as para osworkflow -->
        <dependency>
            <groupId>opensymphony</groupId>
            <artifactId>osworkflow</artifactId>
            <version>2.8.0</version>
        </dependency>
        <dependency>
            <groupId>opensymphony</groupId>
            <artifactId>oscore</artifactId>
            <version>2.2.5</version>
        </dependency>
        <dependency>
            <groupId>opensymphony</groupId>
            <artifactId>propertyset</artifactId>
            <version>1.4</version>
        </dependency>


        <!-- anotaciones de Hibernate -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-annotations</artifactId>
            <version>3.4.0.GA</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-commons-annotations</artifactId>
            <version>3.3.0.ga</version>
        </dependency>

        <!-- springframework -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>2.5.6</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>2.5.6</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-hibernate3</artifactId>
            <version>2.0.8</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.5.2</version>
        </dependency>

        <!-- driver de mysql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.6</version>
        </dependency>

        <!-- anotaciones de java -->
        <dependency>
            <groupId>javax.annotation</groupId>
            <artifactId>jsr250-api</artifactId>
            <version>1.0</version>
        </dependency>
    </dependencies>
    <build>
        <finalName>workflow</finalName>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.5</source>
                    <target>1.5</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

OSWorkflow.

Como hemos adelantado, OSWorkflow es un motor de workflow en el que podemos delegar la responsabilidad de todo el workflow. OSWorkflow nos permite definir los distintos pasos por los que pasar, restricciones para la ejecución de acciones, acciones con alternativas del estado al que pasar dependiendo del resultado de la acción, acciones que bifurcan o unen el flujo, funciones que se ejecutan antes o después de realizar una acción, etc.

Para ver una parte de las posibilidades que nos ofrece nos vamos a basar en un ejemplo de envío de pedidos con la siguiente máquina de estados, que simplifica la gestión de pedidos en un almacén.

2009-03-24
Cómo ver el correo de Gmail sin conexión a Internet

2009-03-20
JasperReports Maven Plugin

2009-03-16
Creación de contenidos SCORM: eXe

2009-03-15
Spring WS: Creación de Servicios Web con Spring

2009-03-13
Instalación Alfresco (Labs)

2009-02-26
Maven JXR Plugin: publica el código fuente en el site

2009-03-15
Generación de XML Schema (XSD) y DTD a partir de documentos XML

2009-03-04
Persistencia con Spring

2009-02-26
Vistas materializadas

2009-02-03
Instalación de MySQL 5.1 en Windows

2009-03-03
Instalación de Java Virtual Machine

2009-03-03
Primeros Pasos con Liferay 5.2.1

2009-02-27
Edicion de video MPEG2

2009-02-26
Introducción teórica a XPath

2009-02-26
Integración Selenium / Maven 2 / Surefire / Cargo / Tomcat 6

2009-02-24
Selenium Remote Control

2009-02-22
Integración de Groovy, JRuby y BeanShell con Spring 2

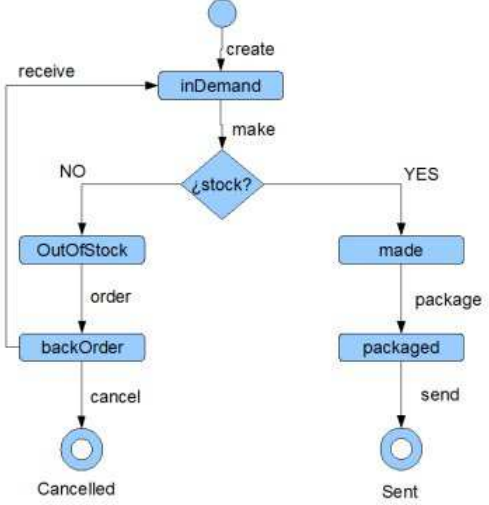
2009-02-18
Instalación de Pentaho BI Suite Community Edition 1.7.0

2009-02-18
Replicar Web PHP en máquina local

2009-02-16
Selenium Core : El motor de Selenium.

2009-02-16
Integración de JasperReports con PHP

2009-02-09
EJB 3.0 y pruebas unitarias con Maven, JUnit 4 y Embedded JBoss sobre Java 6



OSWorkflow nos permite definir el flujo mediante un fichero XML, donde los elementos principales son:

- pasos (<step>): Son los distintos estados posibles del flujo. En nuestro ejemplo lo son "inDemand", "made", "outOfStock", etc.
- acciones (<action>): Serían las transiciones entre un estado y otro. En nuestro ejemplo lo son "make", "order", "package", etc. Los elementos <action> se definen dentro de los elementos <step>.
- estado (atributos status y old-status): OSWorkflow nos permite definir el estado en que se encuentra una instancia del workflow dentro de un paso. Vendría a ser como un subestado dentro de los estados principales. Es un string al que le podemos asignar el valor que nos convenga, normalmente suele ser suficiente con :
  - "Underway": Suele indicar que el workflow se encuentra en dicho paso y está disponible para seguir su flujo.
  - "Queued": Suele indicar que el workflow se encuentra en dicho paso y está esperando algún evento para poder seguir su flujo.
  - "Finished": Suele indicar que el paso ya ha sido abandonado.

  Nos puede servir para restringir las acciones a realizar dentro de un paso dependiendo del subestado de dicho paso. Estos atributos se definen en los resultados de las acciones.
- resultados (<result> y <unconditional-result>): Se definen dentro de las acciones, siendo obligatorio definir un elemento <uncoditional-result>. En el resultado se indicará el subestado en el que se queda el paso en el que se ejecuta la acción (mediante el atributo "old-status") y el estado del paso al que se dirige el flujo (mediante el atributo "status"). Para indicar a que paso se salta al ejecutar la acción se define por medio del atributo "step". El hecho de tener la posibilidad de elementos <result> y <unconditional-result> nos sirve para incluir condiones que dirijan el flujo. Estas condiciones se definirán dentro de los elementos <result>. En caso de no cumplirse ninguna, el resultado de la acción será el definido por el elemento <unconditional-result>

Además de estos elementos, en la definición del workflow se pueden definir otros adicionales, como permisos, condiciones, bifurcaciones y uniones, etc. algunos de los cuales iremos viendo en el ejemplo. Para más información podéis ver la documentación de OSWorkflow.

Sabiendo los elementos de definición del workflow creamos un fichero XML que define la máquina de estados del ejemplo.

2009-02-09
Web Service Security

2009-02-09
Manual Avanzado de Firebug

2009-01-29
Ejemplo con Mockito

2009-01-29
Uso de Mock objects en pruebas con Mockito

2009-01-29
StrutsTestCase

2009-01-28
Eventos en Hibernate (parte III)

2009-01-28
Eventos en Hibernate (parte II)

2009-01-27
Eventos en Hibernate (parte I)

2009-01-25
Aprendiendo XMLSchema a través de ejemplos

2009-01-20
Pruebas Software con Junit 4 y Eclipse

2009-01-19
Executor : Un programa para ejecutarlos a todos.

2009-01-18
Soap Monitor: Monitorización de mensajes SOAP en Axis2

2009-01-16
Restaurar una Base de Datos en SQL Server o como cambiar el propietario de los objetos de la base de datos

2009-01-14
Solución a NoClassDefFoundError: SWTResourceUtil

Últimas ofertas de empleo

2009-03-26
Comercial - Ventas - ALMERIA.

2009-03-12
Comercial - Ventas - VALENCIA.

2009-03-12
Comercial - Ventas - SEVILLA.

2009-02-21
Otras - Estética/Peluquería - MADRID.

2009-02-13
T. Información - Otros no catalogados - MADRID.

view plain print ?

```
<!DOCTYPE workflow PUBLIC "-//OpenSymphony Group//DTD OSWorkflow 2.6//EN" "http://www.opensymphony.com/osworkflow/workfl
<workflow>
    <initial-actions>
      <!-- Es la acción inicial que se ejecuta al crear una instancia del workflow -->
        <action id="100" name="create">
            <results>
                <unconditional-result old-status="Finished"
                    status="Underway" step="1" />
            </results>
        </action>
    </initial-actions>
    <steps>
        <step id="1" name="inDemand">
            <actions>
                <action id="1" name="make">
                    <results>
                        <result old-status="Finished" status="Underway"
                            step="5">
                            <!-- conjunto de condiciones que si se cumplen este resultado será el resultado al ejecutar
                            <conditions type="AND">
                                <condition type="class">
                                    <arg name="class.name">
                                        com.autentia.workflow.conditions.CheckStock
                                    </arg>
                                </condition>
                            </conditions>
                        </result>
                        <!-- El "unconditional-result" será el resultado si no se cumple la condición del resultado defi
                        <unconditional-result old-status="Finished"
                            status="Underway" step="2" />
                    </results>
                </action>
            </actions>
        </step>
        <step id="2" name="outOfStock">
            <actions>
                <action id="2" name="order">
                    <results>
                        <unconditional-result old-status="Finished"
                            status="Underway" step="3" />
                    </results>
                </action>
            </actions>
        </step>
        <step id="3" name="backOrder">
            <actions>
                <action id="3" name="receive">
                    <results>
                        <unconditional-result old-status="Finished"
                            status="Underway" step="1" />
                    </results>
                </action>
                <action id="4" name="cancel">
                  <!-- Ejemplo para incluir una condición que deberá cumplirse para tener permisos de ejecución de la ac
                    <restrict-to>
                        <conditions type="AND">
                            <condition type="class">
                                <arg name="class.name">
                                    com.opensymphony.workflow.util.StatusCondition
                                </arg>
                                <arg name="status">Underway</arg>
                            </condition>
                        </conditions>
                    </restrict-to>
                    <results>
                        <unconditional-result old-status="Finished"
                            status="Finished" step="4" />
                    </results>
                </action>
            </actions>
        </step>
        <step id="4" name="cancelled"></step>
        <step id="5" name="made">
            <actions>
                <action id="5" name="package">
                    <results>
                        <unconditional-result old-status="Finished"
                            status="Underway" step="6" />
                    </results>
                </action>
            </actions>
        </step>
        <step id="6" name="packaged">
            <actions>
                <action id="6" name="send">
                    <results>
                        <unconditional-result old-status="Finished"
                            status="Finished" step="7" />
                    </results>
                </action>
            </actions>
        </step>
        <step id="7" name="sent"></step>
    </steps>
</workflow>
```

En este fichero de ejemplo se puede ver como se han definido todos los pasos por los que puede pasar el flujo y las acciones disponibles a realizar en cada uno de ellos, tal y como están definidos en la máquina de estados.

Al principio del fichero se deben definir las posibles acciones iniciales. Estas acciones iniciales son las que crean una nueva instancia del workflow. En nuestro caso de ejemplo tenemos que la acción inicial es "create", en la que se define como resultado de la acción que la instacia del workflow va al paso con id=1 (estado "inDemand").

```
view plain  print  ?

.....
<initial-actions>
  <!-- Es la acción inicial que se ejecuta al crear una instancia del workflow -->
    <action id="100" name="create">
        <results>
            <unconditional-result old-status="Finished"
                status="Underway"  step="1" />
        </results>
    </action>
</initial-actions>
.....
```

En el primer paso (step con id=1) se puede ver como hay dos posibles resultados para la acción "make" dependiendo de la comprobación del stock; para lo cual nos hemos tenido que definir una condición que devuelve "true" o "false". Podemos ver como esta condición va definida en un elemento <conditions> con un atributo "type" con valor "AND", como podemos suponer dentro del elemento <conditions> podríamos definir múltiples condiciones, que de cumplirse todas, al tener el operador "AND", este resultado sería el resultado de ejecutar la acción.

```
view plain  print  ?

.....
<conditions type="AND">
    <condition type="class">
        <arg name="class.name">
            com.autentia.workflow.conditions.CheckStock
        </arg>
    </condition>
</conditions>
.....
```

Definir condiciones en OSWorkflow es bastante sencillo, sólo hay que crear una clase que implemente el interfaz "com.opensymphony.workflow.Condition" implementando el método "public boolean passesCondition(Map transientVars, Map args, PropertySet propertySet)". Donde los parámetros son:

- transientVars: Conjunto de variables no persistentes donde podemos encontrar;
  - store: La instancia del WorkflowStore.
  - descriptor: El descritor del workflow, donde podemos acceder a su definición.
  - currentSteps: Un array con los pasos en los que se encuentra la instancia del workflow.
  - entry: La representación del flujo recorrido, donde podremos acceder a los pasos actuales de la instancia del workflow y a los pasos que se han recorrido.
  - context: El contexto de la instancia del gestor de workflows.
  - actionId: El identificador de la acción que se está ejecutando.
  - configuration: La configuración del gestor de workflow.
  - createdStep: El nuevo paso que se ha creado al ejecuta la acción.
- args: Argumentos que pueden ser pasados desde la definición en el xml con elementos <arg name="argumento">valor</arg>. (Se ve con un ejemplo de Roles un poco más abajo)
- propertySet: Conjunto de propiedades que se desea sean persistentes. No se explica en este tutorial. (Utiliza la librería PropertySet).

En nuestro caso hemos definido la condición "CheckStock" que devuelve true si el identificador de la instancia del workflow es impar:

```
view plain  print  ?

public class CheckStock implements Condition {

    public boolean passesCondition(final Map transientVars, final Map args, final PropertySet propertySet)
            throws WorkflowException {
        final WorkflowEntry entry = (WorkflowEntry)transientVars.get( "entry");
        if (entry.getId() % 2 == 0) {
            return false;
        }
        return true;
    }

}
```

A modo de ejemplo, en la acción con id=4, se ha creado una restricción de ejecución. Esto sirve para que sólamente se pueda ejecutar dicha acción si se cumplen las condiciones definidas en la restricción. En nuestro caso de ejemplo sólo se permite ejecutar dicha acción si el paso se encuentra en estado "Underway".

Por poner otro ejemplo de uso, se podría definir una condición que sólo permitiese la ejecución de la acción si el usuario pertenece a unos posibles roles.

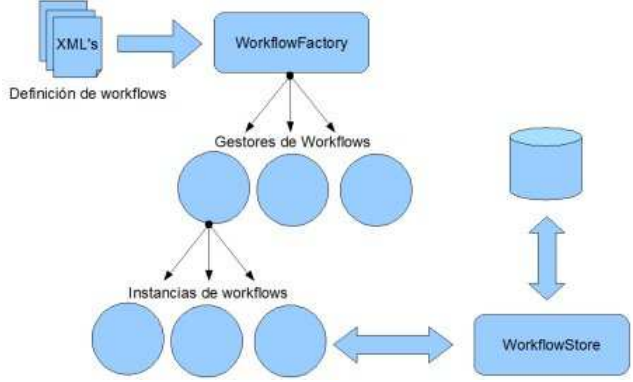
```
view plain  print  ?

.....
<!--- restricción de ejemplo -->
<restrict-to>
   <conditions type="AND">
       <condition type="class">
           <arg name="class.name">
               com.autentia.workflow.conditions.UserInRole
           </arg>
           <arg name="roles">Rol1, Rol2, Rol3, etc.</arg>
       </condition>
   </conditions>
</restrict-to>
.....
```

Ya tenemos definido nuestro workflow (máquina de estados) en el fichero "commissioning.xml", que colocaremos en el classpath de nuestro proyecto para que sea accesible, pero faltaría definir la configuración propia de OSWorkflow:

- WorkflowStore: Responsable de persistir las distintas instancias de los posibles workflows. Por ejemplo en un sistema de pedidos, cada pedido en particular, tendrá su propia instancia de workflow (mantiene el estado en el que se encuentra el pedido, acciones que puede ejecutar, etc.) aunque todas ellas son del mismo tipo.
- WorkflowFactory: Responsable de crear los distintos tipos de workflows (gestores de workflows) partiendo de su definición.

Nota: Lo que hemos llamado "gestor de workflow" será una instancia de una clase que implemente el interfaz "com.opensymphony.workflow.Workflow", donde se definen los servicios para trabajar con las instancias concretas de los workflows.

Vamos a aclarar lo que es un WorkflowStore, WorkflowsFactory, gestor de workflows, instancias de workflow y tipos de workflow un poco más. Si tenemos ficheros XML's que definen workflows de forma lógica; con el WrokflowFactory crearemos un gestor de workflows por cada tipo. Con el "gestor" que se ha creado podrémos crear instancias concretas de ese workflow, y a través del gestor iremos realizando las distintas acciones sobre cada instancia. Por último, tenemos el WorkflowStore, que es el responsable de la persistencia de las instancias concretas de workflow que hemos creado con el "gestor de workflows". En el gráfico siguiente vemos una representación de este funcionamiento:



Al integrar OSWorkflow con Spring, esta configuración la haremos directamente en los fichero de configuración de Spring. Si no estamos integrando con Spring, hay que definirse el fichero de configuración "osworkflow.xml" que también debería estar en el classpath. Un ejemplo de este fichero sería:

```
view plain  print  ?
<osworkflow>
    <persistence class="com.opensymphony.workflow.spi.memory.MemoryWorkflowStore" />

    <factory class="com.opensymphony.workflow.loader.XMLWorkflowFactory" >
        <property key="resource" value="workflows.xml" />
    </factory>
</osworkflow>
```

Se puede ver que guarda las distintas instancias de los workflows en memoria, no utiliza ninguna base de datos. En cuanto al WorkflowFactory, al haber definido el workflow en XML (podría estar en una BD), utilizamos la clase "XMLWorkflowFactory" que necesita una propiedad (el fichero "workflows.xml") que le indica donde están definidos los distintos workflows. El fichero "workflows.xml", también lo ubicaremos en nuestro classpath, y tendrá una referencia por cada workflow; en nuestro caso:

```
view plain  print  ?
<workflows>
    <workflow name="commissioning" type="resource" location="commissioning.xml"/>
</workflows>
```

Integrando OSWorkflow con Spring

Una de las ventajas que tenemos con OSWorkflow es que lo podemos integrar con Spring añadiendo las referencias necesarias en el "applicationContext.xml". Siendo en este fichero donde definiremos la configuración que antes se hacía en el fichero "osworkflow.xml". El "applicationContext.xml" será:

view plain print ?

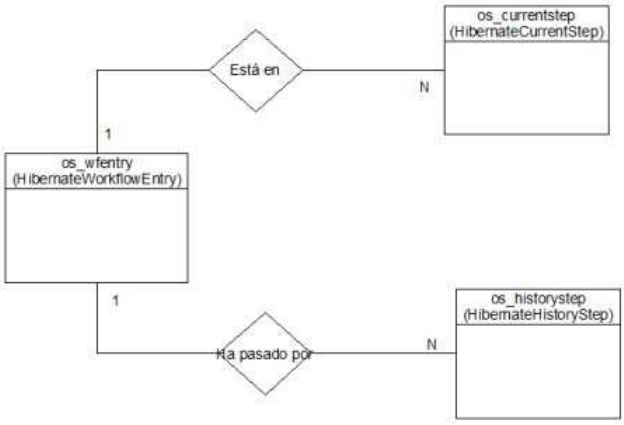
```xml
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"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

    <context:annotation-config />
    <context:component-scan base-package="com.autentia.workflow" />

    <!-- ............................ -->
    <!-- Configuracion del datasource -->
    <!-- ............................ -->
    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url"
            value="jdbc:mysql://localhost:3306/osworkflow" />
        <property name="username" value="osworkflow" />
        <property name="password" value="osworkflow" />
    </bean>

    <!-- ........................... -->
    <!-- Configuracion de hibernate -->
    <!-- ........................... -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean" >
        <property name="dataSource" ref="dataSource" />
        <property name="configurationClass"
            value="org.hibernate.cfg.AnnotationConfiguration" />
        <property name="configLocation">
            <value>classpath:hibernate.cfg.xml</value>
        </property>
    </bean>
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager" >
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>

    <!-- ..................................................... -->
    <!-- Definicion de los beans necesarios para el workflow     -->
    <!-- ..................................................... -->
    <bean id="workflowStore"
        class="com.opensymphony.workflow.spi.hibernate3.SpringHibernateWorkflowStore" >
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
        <property name="propertySetDelegate">
            <bean id="propertySetDelegate"
                class="com.opensymphony.workflow.util.PropertySetDelegateImpl" />
        </property>
    </bean>
    <bean id="workflowFactory"
        class="com.opensymphony.workflow.loader.XMLWorkflowFactory"
        init-method="initDone" />
    <bean id="osworkflowConfiguration"
        class="com.opensymphony.workflow.config.SpringConfiguration" >
        <property name="store">
            <ref local="workflowStore" />
        </property>
        <property name="factory">
            <ref local="workflowFactory" />
        </property>
    </bean>
    <bean id="workflowTypeResolver"
        class="com.opensymphony.workflow.util.SpringTypeResolver" >
    </bean>
    <bean id="workflow"
        class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean" >
        <property name="transactionManager">
            <ref bean="transactionManager" />
        </property>
        <property name="target">
            <ref local="workflowTarget" />
        </property>
        <property name="transactionAttributes">
            <props>
                <prop key="*">PROPAGATION_REQUIRED</prop>
            </props>
        </property>
    </bean>
    <bean id="workflowTarget"
        class="com.opensymphony.workflow.basic.BasicWorkflow" >
        <constructor-arg>
            <value>workflow</value>
        </constructor-arg>
        <property name="configuration">
            <ref local="osworkflowConfiguration" />
        </property>
        <property name="resolver">
            <ref local="workflowTypeResolver" />
        </property>
    </bean>
</beans>
```

Además de la configuración tipica de "dataSource", "transactionManager", etc. tenemos la configuración propia de OSWorkflow, donde configuramos:

- workflowStore: En este caso, sí queremos que se guarde en la base de datos, por lo que utilizamos "SpringHibernateWorkflowStore" indicando que utilice el "sessionFactory" definido en el fichero.
- workflowFactory: Al seguir teniendo nuestra defiición en XML sigue siendo "XMLWorkflowFactory". Toma por defecto el fichero "workflows.xml" que se encuentra en el classpath.
- osworkflowConfiguration: Utilizamos "SpringConfiguration" indicando el "WorkflowStore" y "WorkflowFactory". Lo pasaremos como referencia al workflow en concreto.
- workflowTypeResolver: Utilizando "SpringTypeResolver" indicamos que podemos hacer referencias a "beans" de Spring, directamente desde los ficheros XML que definen los workflows (lo veremos un poco más adelante).
- workflow: Al estar utilizando como persistencia Hibernate, utilizamos "TransactionProxyFactoryBean" para asegurarnos que el acceso a los métodos de la clase "BasicWorkflow" (declarada como "target") se hacen en una transacción. De esta forma nos aseguramos no tener

problemas de sessiones al ir recuperando las entidades que gestiona (HibernateWorkflowEntry, HibernateCurrentStep e HibernateHistoryStep). En caso de no acceder mediante este proxy, tendríamos excepciones del tipo "LazyInicializationException" al acceder a los métodos de la clase que va a gestionar los workflows.
- workflowTarget: Utilizado como objetivo del bean "workflow" y sera el tipo concreto de workflow a crear. En nuestro ejemplo utilizamos "BasicWorkflow", pero podríamos crear nuestra propia clase de workflow.

Al decidir que las instancias concretas de los workflows se almacenen de forma persistente, hemos tenido que incluir el bean "TransactionProxyFactoryBean" que permita acceder a todas las entidades relacionadas con la instancia del workflow en concreto dentro de una transacción, al encontrase su representación relacionando varias entidades. A continuación ponemos la relación de estas entidades en base de datos (y su mapeo de clases) para comprender mejor como es la representación persistente de las instancias de los workflows.



Ahora ya podemos empezar a desarrollar el resto de nuestras clases, DAO's, entidades, managers, etc.

Vamos a definir la entidad que va a estar sometida al workflow:

```
view plain  print  ?
01. @Entity
02. @NamedQuery(name = "findByWorkflowId", query = "Select c from Commissioning c where c.workflowId = ?" )
03. public class Commissioning {
04. 
05.     public static final String findByWorkflowId =  "findByWorkflowId";
06. 
07.     @Id
08.     @GeneratedValue
09.     private Integer id;
10. 
11.     private String name;
12. 
13.     private String description;
14. 
15.     private Long workflowId;
16. 
17.     private Integer state;
18. 
19.     public Integer getState() {
20.         return this.state;
21.     }
22. 
23.     public void setState(final Integer state) {
24.         this.state = state;
25.     }
26. 
27.     public Long getWorkflowId() {
28.         return this.workflowId;
29.     }
30. 
31.     public void setWorkflowId(final Long workflowId) {
32.         this.workflowId = workflowId;
33.     }
34. 
35.     Commissioning() {
36.         // Sólo el manager puede constuir nuevas instancias
37.     }
38. 
39.     public String getName() {
40.         return this.name;
41.     }
42. 
43.     public void setName(final String name) {
44.         this.name = name;
45.     }
46. 
47.     public String getDescription() {
48.         return this.description;
49.     }
50. 
51.     public void setDescription(final String description) {
52.         this.description = description;
53.     }
54. 
55.     public Integer getId() {
56.         return this.id;
57.     }
58. 
59.     public void setId(final Integer id) {
60.         this.id = id;
61.     }
62. 
63. }
```

Como el control del flujo va a ser a través de OSWorkflow, es conveniente que en nuestra entidad tengamos una referencia a la entrada del workflow para no perder trazabilidad; por eso tenemos el atributo "workflowId", que será asignado al hacer persintente una nueva instancia de la clase "Commissioning".

De igual forma, aunque el estado está controlado por el motor de OSWorkflow, es conveniente que en nuestra entidad tengamos reflejado el estado en el que se encuentra, de esta forma simplificamos posibles informes o cambios del motor de workflow. En el ejemplo, guardaremos en el atributo "state" el paso actual del flujo en el que se encuentra la entidad. Al ser el responsable de un paso a otro el propio workflow, lo debemos delegar en él, creando una función encargada de actualizar el estado de la entidad al cambiar de un paso a otro. Ahora es donde vamos a ver como nos aprovechamos de la integración con Spring, referenciando beans directamente desde la definición del workflow, añadiendo la referencia a la siguiente función en el fichero "commissioning.xml":

```
view plain  print  ?

...
<action ...>
  ....
  <post-functions>
    <function type="spring">
        <arg name="bean.name">setWorkflowState</arg>
    </function>
  </post-functions>
</action>
```

Esta referencia la crearemos para todas las acciones que tenemos definidas en el fichero, excepto para la acción inicial, ya que todavía no hay entidad creada.

Como vemos, la función la hemos definido dentro de un elemento <post-functions> que indica las funciones que se deben ejecutar al terminar la acción (transición). De forma similar podremos definir funciones que se ejecuten antes de ejecutar la acción (al empezar la transición) dentro de un elemento <pre-functions>.

La función la implementaremos en la clase "SetWorkflowState" y será un bean de Spring al que podremos inyectar cualquier otro recurso del contexto de Spring.

```
view plain  print  ?

@Service
public class SetWorkflowState implements FunctionProvider {

    private static final Log log = LogFactory.getLog(SetWorkflowState. class);

    @Resource
    CommissioningMgr commissioningMgr;

    public void execute(final Map transientVars, final Map args, final PropertySet propertySet)
            throws WorkflowException {
        //recuperamos el paso que se acaba de crear en la transición
        final Step step = (Step)transientVars.get( "createdStep");
        Commissioning commissioning;
        try {
            commissioning = this.commissioningMgr.findByWorkflowId(step.getEntryId());
        } catch (final MyAppException e) {
            throw new WorkflowException(e);
        }
        if (commissioning == null) {
            throw new WorkflowException( "no se ha encontrado ningún pedido asociado al workflow" );
        }
        commissioning.setState(step.getStepId());
    }

}
```

El manager responsable de gestionar los pedidos, también ofrecerá los servicios necesarios para el workflow, por ejemplo, será el responsable de asociar cada instancia del workflow con su entidad correspondiente, devolver las acciones disponibles, etc.

view plain print ?

```
@Service
public class CommissioningMgr {

    @Resource
    private Dao dao;

    @Resource
    private WorkflowUtils workflowUtils;

    public Commissioning newCommissioning() {
        return new Commissioning();
    }

    public void persist(final Commissioning commissioning) throws MyAppException {
        // si es nuevo
        if (commissioning.getId() == null) {
            try {
              //se creará una nueva instancia de workflow y se asocia con la instancia de la clase comm
                commissioning.setWorkflowId( this.workflowUtils.newWorkflow());
                commissioning.setState( 1);
                this.dao.persist(commissioning);
            } catch (final WorkflowException e) {
                throw new MyAppException(e);
            }
        } else {
            this.dao.persist(commissioning);
        }

    }

    public List<Commissioning> getCommissionings() {
        final List<Commissioning> list = this.dao.find(Commissioning.class);
        return list;
    }

    public int[] getAvalaibleActions(final Commissioning commissioning) {
        final Long wfId = commissioning.getWorkflowId();
        return this.workflowUtils.getAvalaibleActions(wfId);

    }


    public void executeAction(final Commissioning commissioning, final int actionId) throws MyAppException {
        try {
            final Long wfId = commissioning.getWorkflowId();
            this.workflowUtils.executeAction(wfId, actionId);
            this.dao.refresh(commissioning);
        } catch (final WorkflowException e) {
            throw new MyAppException(e);
        }
    }

    public Commissioning findById(final Integer id) {
        final Commissioning commissioning = this.dao.get(Commissioning.class, id);
        return commissioning;
    }

    public Commissioning findByWorkflowId(final long id) throws MyAppException {
        final Object[] params = { id };
        final List<Commissioning> list = this.dao.findForWorkflowFuntion(Commissioning.findByWorkflowId, params);
        // como máximo sólo debería haber uno
        if (list.size() > 1) {
            throw new MyAppException("solo debería haber un pedido apuntando a este workflow" );
        } else if (list.size() == 1) {
            return list.get(0);
        } else {
            return null;
        }
    }
}
```

Como caso a destacar es el método "executeAction(final Commissioning commissioning, final int actionId)", donde podemos ver que se realiza un "refresh" después de haber ejecutado la acción sobre el workflow. Esto es debido a que como es posible que alguna función del workflow modifique el estado de la entidad (en nuestro caso es así) tenemos que refrescarla para que los cambios se vean reflejados en la propia entidad.

También podemos ver cómo para las acciones propias del workflow nos apoyamos en la clase "WorkflowUtils", que es donde hemos delegado la gestión del workflow a más bajo nivel. Es la clase responsbles de inicializar el workflow, ejecutar las acciones sobre el mismo y recuperar las acciones disponibles.

view plain print ?

```
@Service
public class WorkflowUtils {

    private static final Log log = LogFactory.getLog(WorkflowUtils.class);

    private static final String WF_COMMISSIONING = "commissioning";

    private static final int WorkflowUtils.WF_INITIAL_ACTION = 100;

    @Resource
    private Workflow workflow;


    public Long newWorkflow() throws InvalidActionException, InvalidRoleException, InvalidInputException,
            InvalidEntryStateException, WorkflowException {
        final Long id = this.workflow.initialize(WorkflowUtils.WF_COMMISSIONING, WorkflowUtils.WF_INITIAL_ACTION,  null
        return id;
    }

    public int[] getAvalaibleActions(final long wfId) {
        final int[] actions = this.workflow.getAvailableActions(wfId, null);
        return actions;
    }

    public void executeAction(final long wfId, final int actionId) throws NumberFormatException, InvalidInputException,
            WorkflowException {

        this.workflow.doAction(wfId, actionId, null);
    }
}
```

En esta clase se puede ver como se trabaja con el "gestor de workflows" siendo éste el recurso "workflow".

En el método "newWorkflow()" se crear una nueva instancia del workflow "commissioning" (WF_COMMISSIONING)) ejecutando la acción inicial aquella con id=100 (WF_INITIAL_ACTION) al ejecuta el método "initialize(....)" del "gestor de workflows".

Para obtener las acciones disponibles de una instancia concreta, llamamos al método "getAvalaibleActions(...)" del "gestor de workflows" pasando como parámetro el identificador de la instancia del workflow.

Y para poder ejecutar, método "executeAction(...)" del "gestor de workflows", una determinada acción sobre una instancia de workflow concreta, se pasa como parámetro el identificador de la instancia del workflow sobre la que ejecutar la acción, y el identificador de la acción a ejecutar.

Probando el ejemplo

Habiendo creado ya todas las clases de nuestro ejemplo, pasamos a probarlo recorriendo los caminos posibles del workflow. Para ello nos creamos una clase de "Test" con dos casos de prueba. En el primero hacemos un recorrido completo suponiedo que hay stock. En el segundo caso de prueba suponemos que no tenemos stock, así llevamos el flujo por la otra rama, haciendo que vuelva de nuevo al estado "inDemand" (primer paso), y volviendo de nuevo por la rama de "sin stock" para terminar cancelando.

view plain print ?

```
01. public class WorkflowTest {
02.
03.     private static String[] files = new String[] { "applicationContext.xml" };
04.
05.     final ApplicationContext context;
06.
07.     public WorkflowTest() {
08.         this.context = new ClassPathXmlApplicationContext(WorkflowTest.files);
09.     }
10.
11.     private void crearPedidos() throws MyAppException {
12.         final CommissioningMgr commissioningMgr = (CommissioningMgr) this.context.getBean("commissioningMgr");
13.         // creamos un nuevo pedido con el que trabajar
14.         Commissioning commissioning = commissioningMgr.newCommissioning();
15.         commissioning.setName( "commissioning1");
16.         commissioning.setDescription( "desc1");
17.         // al guardarlo se creara el workflow
18.         commissioningMgr.persist(commissioning);
19.         // vemos las acciones disponibles
20.         int[] workflowActions = commissioningMgr.getAvalaibleActions(commissioning);
21.         // sólo debemos tener la accion de comprobar stock
22.         Assert.assertEquals( 1, workflowActions.length);
23.         // la única acción debe ser preparar el pedido (id=1)
24.         Assert.assertEquals( 1, workflowActions[ 0]);
25.
26.         // creamos otronuevo pedido con el que trabaja
27.         commissioning = commissioningMgr.newCommissioning();
28.         // al guardarlo se crea el workflow
29.         commissioningMgr.persist(commissioning);
30.         // vemos las acciones disponibles
31.         workflowActions = commissioningMgr.getAvalaibleActions(commissioning);
32.         // sólo debemos tener la accion de comprobar stock
33.         Assert.assertEquals( 1, workflowActions.length);
34.         // la única acción debe ser preparar el pedido (id=1)
35.         Assert.assertEquals( 1, workflowActions[ 0]);
36.
37.         // comprobamos que tenemos 2 pedidos en la base de datos
38.         final List<Commissioning> commissionings = commissioningMgr.getCommissionings();
39.         Assert.assertEquals( 2, commissionings.size());
40.     }
41.
42.     /**
43.       * Crea dos pedidos y recupera el primer pedido y ejecuta todo el workflow hasta el final
44.       */
45.     @Test
46.     public void testConStock() {
47.         try {
48.             this.crearPedidos();
49.             final CommissioningMgr commissioningMgr = (CommissioningMgr) this.context.getBean("commissioningMgr");
50.             // creamos un nuevo pedido con el que trabajar
51.             final Commissioning commissioning = commissioningMgr.findById( 1);
52.             // se debe haber recuperado la instancia
53.             Assert.assertNotNull(commissioning);
54.             // vemos las acciones disponibles
55.             int[] workflowActions = commissioningMgr.getAvalaibleActions(commissioning);
56.             // sólo debemos tener la accion de preparar
57.             Assert.assertEquals( 1, workflowActions.length);
58.             // la única acción debe ser preparar el pedido (id=1)
59.             Assert.assertEquals( 1, workflowActions[ 0]);
60.
61.             // ejecutamos la accion
62.             commissioningMgr.executeAction(commissioning, workflowActions[ 0]);
63.             // como estamos con el pedido con id=1
64.             // debe pasar la comprobación de stock, por lo
65.             // que estaremos en el paso con id=5
66.             Assert.assertEquals( 5, commissioning.getState().intValue());
67.             // y si ahora recuperamos las acciones disponibles
68.             // sólo debemos tener una, la acción de empaquetar (id=5)
69.             workflowActions = commissioningMgr.getAvalaibleActions(commissioning);
70.             Assert.assertEquals( 1, workflowActions.length);
71.             Assert.assertEquals( 5, workflowActions[ 0]);
72.
73.             // ejecutamos la acción disponible es empaquetar (id=5)
74.             commissioningMgr.executeAction(commissioning, workflowActions[ 0]);
75.             // hemos pasado al paso 6
76.             Assert.assertEquals( 6, commissioning.getState().intValue());
77.
78.             // si ahora recuperamos las acciones disponibles
79.             // sólo debemos tener una, la acción de enviar (id=6)
80.             workflowActions = commissioningMgr.getAvalaibleActions(commissioning);
81.             Assert.assertEquals( 1, workflowActions.length);
82.             Assert.assertEquals( 6, workflowActions[ 0]);
83.
84.             // ejecutamos la acción disponible es empaquetar (id=5)
85.             commissioningMgr.executeAction(commissioning, workflowActions[ 0]);
86.             // hemos pasado al paso 7
87.             Assert.assertEquals( 7, commissioning.getState().intValue());
88.
89.             // ahora no debemos tener ninguna acción disponible al haber llegado
90.             // al final del workflow
91.             workflowActions = commissioningMgr.getAvalaibleActions(commissioning);
92.             Assert.assertEquals( 0, workflowActions.length);
93.
94.         } catch (final MyAppException e) {
95.             e.printStackTrace();
96.             Assert.fail( "Error");
97.         }
98.
99.     }
100.
101.    /**
102.      * Crea dos pedidos y recupera el segundo pedido, se ejecuta el workflow por la rama de "sin stoc
103.      * suministro, y en la segunda vuelta del workflow se cancela
104.      */
105.    @Test
106.    public void testSinStock() {
107.        try {
108.            this.crearPedidos();
109.            final CommissioningMgr commissioningMgr = (CommissioningMgr) this.context.getBean("commissioningMgr");
110.            // creamos un nuevo pedido con el que trabajar
111.            final Commissioning commissioning = commissioningMgr.findById( 2);
```

Conclusiones

Hemos visto como podemos delegar los flujos en un motor de workflow donde configuraremos los distintos workflows que tengamos; de esta forma queda de una forma más clara el seguimiento del flujo de una entidad; cosa que antes teníamos que ir recorriendo el código fuente, de la misma forma que era muy probable cometer errores (dejando casos sin contemplar); algo que ahora es mucho más fácil de detectar.

También se puede apreciar como utilizando un motor de workflow, es fácilmente extensible la casuística, ya simplemente hay que definir los cambios en el fichero de definición del workflow.

Si queréis, aquí podéis conseguir todo el código fuente de este ejemplo OSWorkflow con Spring.

Un saludo.
Borja Lázaro de Rafael.

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	Valoración	Votos	Pdf
Introducción a Eclipse 3.3 / Europa	En este tutorial se dará a conocer la nueva version de Eclipse, en la que se introducen nuevas funcionalidades, soporte para Java 6.	2007-07-04	8128	Muy bueno	1	
Spring WebFlow con Validator	En este tutorial se muestra como podemos realizar las validaciones más frecuentes de datos mediante Spring WebFlow.	2007-12-11	4610	Muy bueno	1	
Nuevo Eclipse 3.4, code name: Ganymede	Álex ha probado la nueva versión de Eclipse 3.4 (Ganymede) liberada ayer mismo como GA. Conoce las novedades y mejoras en este tutorial.	2008-06-26	5784	Muy bueno	5	
URLs amigables con Spring MVC	En este tutorial se va a hacer un ejemplo práctico utilizando Spring MVC para la configuración de URLs amigables de nuestra aplicación	2007-04-11	9406	Bueno	4	
Introducción a Spring Web Flow	Spring Web Flow es un módulo de extensión del framework Spring, que facilita la implementación del flujo de páginas de una aplicación web	2006-01-03	23193	Bueno	10	
Spring: definición dinámica de Beans	Este tutorial habla sobre la modificación dinámica de los beans del contexto para simplificar la configuración de Spring	2007-05-09	6217	Regular	4	
Manual básico de Spring WebFlow	En este tutorial Javier Antoniucci nos enseña cómo empezar a trabajar cpn el framework de desarrollo web Spring webflow.	2007-11-26	5547	Regular	9	
Comparativa entre EJB3 y Spring	En este tutorial os mostramos una comparativa entre EJB3 y Spring esperando que os ayude a decidir qué tecnología utilizar.	2007-10-17	6469	Malo	2	
Spring WebFlow Tiles	En este tutorial aprenderemos el uso de tiles para usarlo en conjunción con Spring WebFlow.	2007-11-26	3887	Muy malo	1	
Instalación y primeros pasos con Bonita Workflow	Este tutorial pretende ser una guía de instalación por pasos del motor de Workflow Bonita.	2007-08-16	4789	Muy malo	1	

Nota: