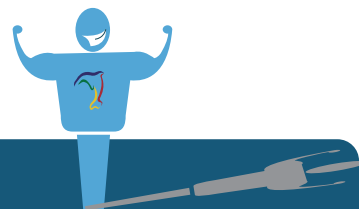


¿Qué ofrece Autentia Real Business Solutions S.L?

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

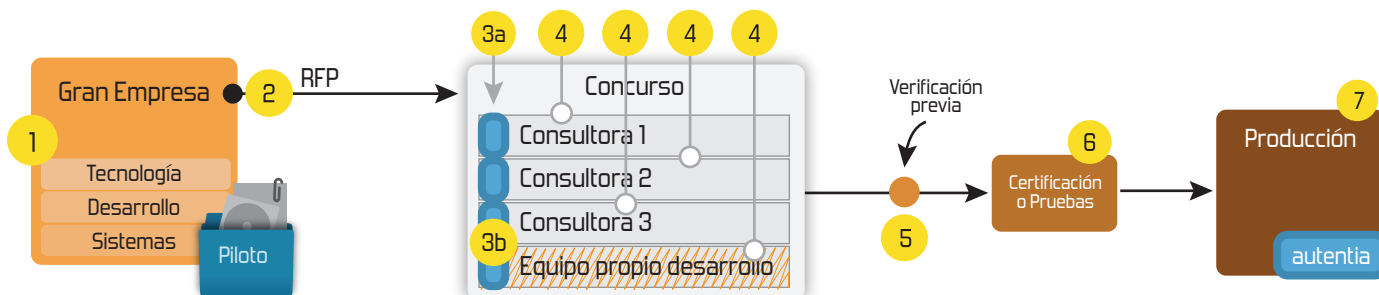
1. Desarrollo de componentes y proyectos a medida



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

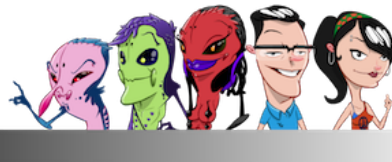
Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

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

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

» Estás en: [Inicio](#) [Tutoriales](#) [Máquina de estados con Apache SCXML](#)[Daniel Ventas López](#)

Becario en autentia.

Ingeniero técnico en informática, especialidad en sistemas.

Puedes encontrarme en [Autentia](#): Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

[Ver todos los tutoriales del autor](#)

Fecha de publicación del tutorial: 2014-02-05

Tutorial visitado 1 veces [Descargar en PDF](#)

Finite State Machine con Apache SCXML

0. Índice de contenidos.

- 1. Entorno.
- 2. Introducción.
- 3. Creando el archivo base scxml.
- 4. Creando proyecto java.
- 5. Conclusiones.

1. Entorno

Este tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Mac Book Pro 17" (2,93 Ghz Intel Core 2 Duo, 8 GB DDR3)
- Sistema Operativo: Mac OS X Mavericks 10.9
- Apache SCXML 0.9
- Eclipse Kepler
- scxmlgui

2. Introducción.

En este tutorial vamos a ver cómo crear una máquina de estados sencillita que pase entre tres estados. Para hacer el comportamiento básico vamos a ayudarnos de la herramienta scxmlgui.

Cuando tengamos el scxml con la funcionalidad básica, vamos a añadirle otras funciones en código java gracias a apache SCXML.

Bien, vamos a explicar cómo es el funcionamiento de una máquina de estados, para ello primero vamos a ver los elementos básicos que tienen.

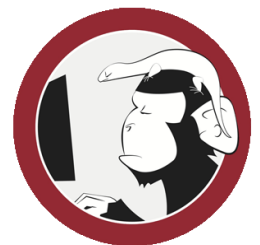
- Estado:** Es la parte más importante, y como su propio nombre indica, es un estado concreto en el flujo de la máquina de estados. Cada estado tiene asociados transiciones y funciones OnEntry y OnExit que se ejecutarán al entrar y al salir del estado, respectivamente. Del estado se sale mediante transiciones que se activan mediante eventos y tienen el estado de destino como Target.
- Transición:** Tienen de propiedades una condición que tiene que cumplir para realizarse, un evento que será la que la dispare y un target que es el estado destino. También se pueden asociar acciones a realizar cuando ocurran.
- OnEntry y OnExit:** Contienen acciones a realizar cuando una transición llegue al estado o salga de él, respectivamente.
- Acción:** Son elementos de código que se incorporan a las transiciones o a OnEntry/OnExit para que sean ejecutadas.

Con esto ya tenemos la base para crear la máquina de estados, así que vamos con ello.

La máquina de estados que voy a realizar tendrá tres estados (A, B y C), con dos transiciones, una entre A y B (transicion_AB) y otra entre B y C (transicion_BC). El estado B tendrá una acción en OnEntry y dos acciones en OnExit. El estado C tendrá una acción en OnEntry. El evento que dispara la transición AB será llamado "Transicion_AB", y el evento de la transición BC "Transicion_BC". Y por último, la transición BC tendrá dos acciones.

En el esquema siguiente queda más claro la explicación

Catálogo de servicios Autentia



Síguenos a través de:



Últimas Noticias

» IX Autentia Cycling Day (ACTUALIZADO)

» Buscamos quien nos ayude en Autentia con WordPress

» XXII Charla Autentia - PhoneGap/Cordova ¡qué bueno que viniste!

» Cerrada búsqueda de CM en @autentia (Enero 2014)

» La historia de la informática

[Histórico de noticias](#)

Últimos Tutoriales

» Intercomunicación de aplicaciones en iOS

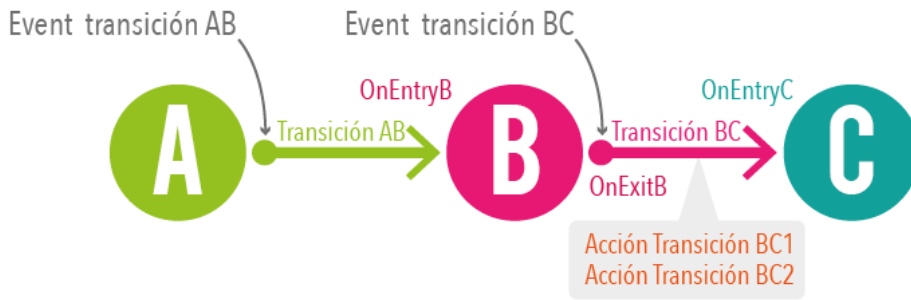
» App iOS para conectar con periférico bluetooth 4.0

» Crashlytics en iOS

» Construir un cliente REST con PowerBuilder .NET

» Acceso a la cámara con PhoneGap

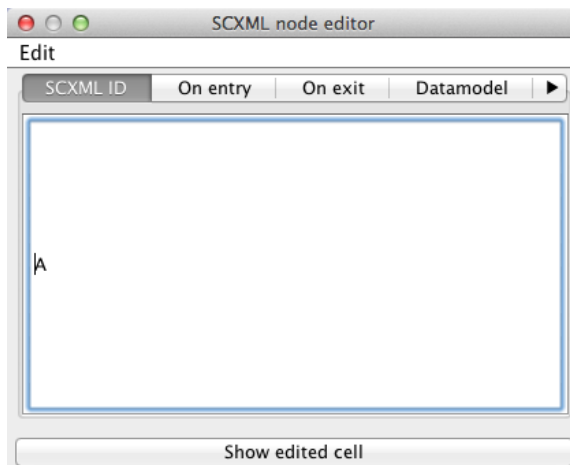
Últimos Tutoriales del Autor



3. Creando el archivo base scxml.

Nos vamos a ayudar de la herramienta scxml, que la podéis encontrar [aquí](#), y su funcionamiento es muy sencillo (ya que sólo vamos a hacer la base y el resto de funcionalidad lo agregaremos desde código java).

Cuando la tengamos abierta, creamos un nuevo archivo (File --> New SCXML). Ahora con el botón derecho sobre la plantilla agregamos un nodo (nodo es el equivalente a estado) con Add node, damos doble click para ponerle de Nombre "A", como aparece en la siguiente imagen.

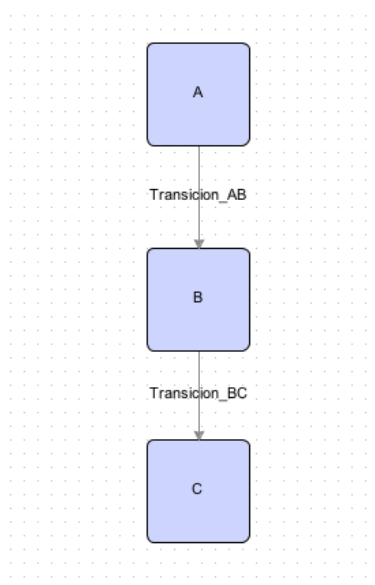


Ahora, con el botón presionado sobre el nodo, arrastramos a otra zona y soltamos, que nos creará un nuevo nodo y una transición que los une, como hicimos antes, renombramos el nuevo nodo y le ponemos "B".

Ahora damos con el botón derecho sobre la transición y "edit transition". Ahora en la pestaña evento ponemos "transición_AB", y en condición ponemos simplemente true.



Ahora ya sólo nos queda hacer lo mismo para crear un nuevo nodo a partir de B, realizamos los pasos anteriores salvo que el nuevo nodo se llamará "C" y la transición que los une "Transición_BC". El resultado tiene que ser el siguiente:



Guardamos el archivo como "fsm.scxml", y su contenido tiene que ser como el que sigue (salvo los comentarios de dónde está colocado cada nodo):

```
view plain print ?
01. <scxml version="0.9" xmlns="http://www.w3.org/2005/07/scxml"><!-- node-size-and-
02. position x=0 y=0 w=664 h=596 -->
03. <state id="A"><!-- node-size-and-position x=166 y=112 w=75 h=75 -->
    <transition cond="true" event="Transición_AB" target="B"></transition>
```

» App iOS para conectar con periférico bluetooth 4.0

» Hola mundo con las tecnologías de SpringMVC, Hibernate, un ejemplo de aspectos y test con JUnit

» Cómo montar un raid1 en una máquina corriendo debian.

» Configuración básica de seguridad en servidor linux con iptables y fail2ban

» Cómo integrar un Job de Talend a nuestro proyecto Java

Últimas ofertas de empleo

2011-09-08

Comercial - Ventas - MADRID.

2011-09-03

Comercial - Ventas - VALENCIA.

2011-08-19

Comercial - Compras - ALICANTE.

2011-07-12

Otras Sin catalogar - MADRID.

2011-07-06

Otras Sin catalogar - LUGO.

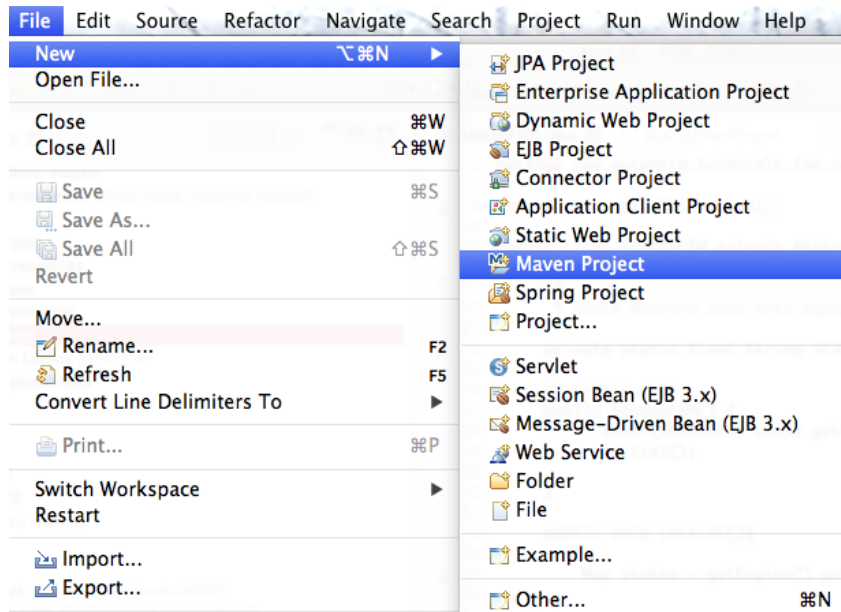
```

04.     </state>
05.     <state id="B"><!-- node-size-and-position x=170 y=240 w=75 h=75 -->
06.         <transition cond="true" event="Transicion_BC" target="C"></transition>
07.     </state>
08.     <state id="C"><!-- node-size-and-position x=170 y=380 w=75 h=75 --></state>
09. </scxml>

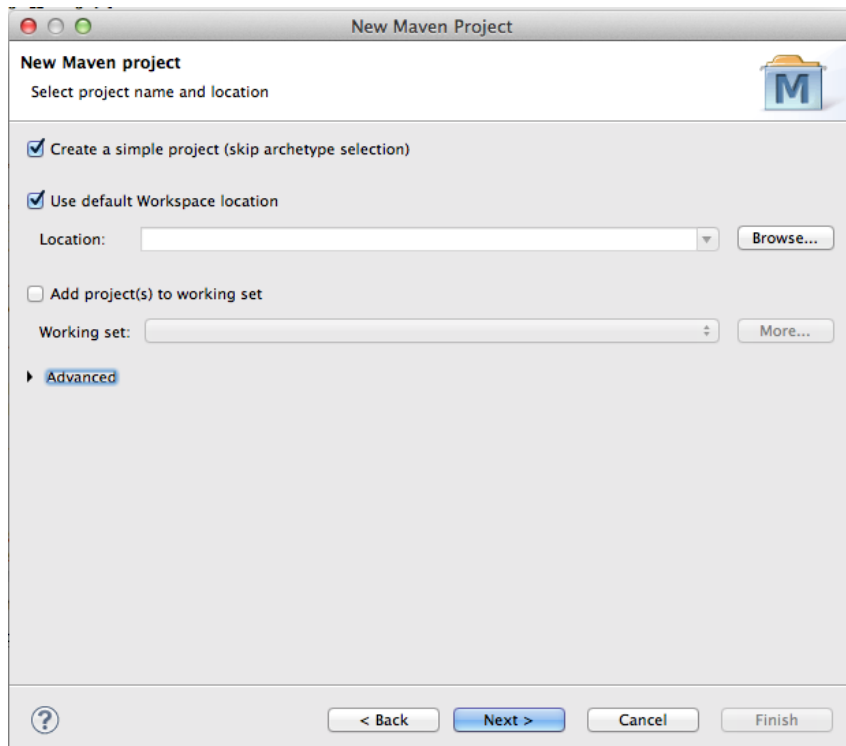
```

4. Creando proyecto java.

Bien, ahora ya tenemos el archivo base, vamos a crear un proyecto en eclipse con maven, para ello vamos a File --> new --> Maven Project. (si no ve Maven Project, pulse en others y lo encontrará si tiene instalado el plugin en eclipse, sino tendrá que instalárselo).



Las siguientes pantallas se describen solas, podéis poner los mismos datos de ejemplo que he puesto yo.



New Maven Project

Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:

Advanced

Bien, ahora vamos a configurar el archivo pom.xml con las dependencias que vamos a necesitar, tiene que quedar así:

```

01. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
02. instance"
03. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
04. 4.0.0.xsd">
05. <modelVersion>4.0.0</modelVersion>
06. <groupId>com.autentia.tutorials</groupId>
07. <artifactId>fsm</artifactId>
08. <version>1.0.0-SNAPSHOT</version>
09. <description>Pruebas de concepto de máquinas de estado finitas con apache SCXML</description>
10.
11. <dependencies>
12. <dependency>
13. <groupId>commons-scxml</groupId>
14. <artifactId>commons-scxml</artifactId>
15. <version>0.9</version>
16. </dependency>
17. <dependency>
18. <groupId>commons-beanutils</groupId>
19. <artifactId>commons-beanutils</artifactId>
20. <version>1.8.2</version>
21. </dependency>
22. <dependency>
23. <groupId>commons-digester</groupId>
24. <artifactId>commons-digester</artifactId>
25. <version>1.8.1</version>
26. </dependency>
27. <dependency>
28. <groupId>commons-el</groupId>
29. <artifactId>commons-el</artifactId>
30. <version>1.0</version>
31. </dependency>
32. <dependency>
33. <groupId>commons-jexl</groupId>
34. <artifactId>commons-jexl</artifactId>
35. <version>1.1</version>
36. </dependency>
37. <dependency>
38. <groupId>commons-logging</groupId>
39. <artifactId>commons-logging</artifactId>
40. <version>1.1.1</version>
41. </dependency>
42. <dependency>
43. <groupId>xalan</groupId>
44. <artifactId>xalan</artifactId>
45. <version>2.6.0</version>
46. </dependency>
47. <dependency>
48. <groupId>junit</groupId>
49. <artifactId>junit</artifactId>
50. <version>4.11</version>
51. </dependency>
52. <dependency>
53. <groupId>org.mockito</groupId>
54. <artifactId>mockito-all</artifactId>
55. <version>1.9.5</version>
56. </dependency>
57. </dependencies>
58. </project>

```

Resumido, hemos incorporado las dependencias para apache SCXML, en la misma versión que viene en la [página oficial](#) (commons-scxml 0.9, commons-beanutils 1.8.2, commons-digester 1.8.1, commons-el 1.0, commons-jexl 1.1, commons-logging 1.1.1 y xalan 2.6.0)

Además he incluido las dependencias para los test (JUnit 4.11 y Mockito 1.9.5).

Una vez hecho esto, vamos a crearnos las clases que vamos a usar. Primero implementaremos nuestra máquina de estados personalizada con los métodos que nos van a hacer falta, heredando de `AbstractStateMachine`, incluida en el paquete `org.apache.commons.scxml.env`. Yo he llamado a la clase `FSM.java`, y la he dejado en la ruta `com/autentia/tutoriales/fsm`, que será donde estén las clases que vamos a implementar.

Dejo el código comentado a continuación:

```
view plain print ?
01. import java.net.URL;
02. import java.util.List;
03.
04. import org.apache.commons.logging.Log;
05. import org.apache.commons.logging.LogFactory;
06. import org.apache.commons.scxml.env.AbstractStateMachine;
07. import org.apache.commons.scxml.model.Action;
08. import org.apache.commons.scxml.model.OnEntry;
09. import org.apache.commons.scxml.model.OnExit;
10. import org.apache.commons.scxml.model.State;
11. import org.apache.commons.scxml.model.Transition;
12.
13.
14. public class FSM extends AbstractStateMachine{
15.
16.     private static final Log LOGGER = LogFactory.getLog(FSM.class);
17.
18.     // Constructor
19.     public FSM(URL xmlPath) {
20.         super(xmlPath);
21.     }
22.
23.     // Añade una accion a una transicion de un estado (stateName) y nombre del evento qu
24.     public void addActionToTransitionByStateAndEvent(String stateName, String event, Action action)
25.     {
26.         State state = getState(stateName);
27.         for(Transition t : (List<Transition>) state.getTransitionsList()){
28.             if (t.getEvent().equals(event)){
29.                 t.addAction(action);
30.             }
31.         }
32.     }
33.
34.     // Añade una condicion a una transicion de un estado (stateName) y nombre del evento
35.     public void addConditionToTransitionByStateAndEvent(String stateName, String event, String cond:
36.     {
37.         State state = getState(stateName);
38.         for(Transition t : (List<Transition>) state.getTransitionsList()){
39.             if (t.getEvent().equals(event)){
40.                 t.setCond(condition);
41.             }
42.         }
43.     }
44.
45.     // devuelve State a partir de nombre en String
46.     public State getState(String stateName){
47.         return (State) getEngine().getStateMachine().getTargets().get(stateName);
48.     }
49.
50.     // Introduce acciones a OnEntry de un estado pasado por su nombre en String
51.     public void setActionInOnEntryByState(String stateName, Action [] actionsToAdd){
52.         State state = getState(stateName);
53.         OnEntry oe = state.getOnEntry();
54.         for(Action action : actionsToAdd){
55.             oe.addAction(action);
56.         }
57.         state.setOnEntry(oe);
58.     }
59.
60.     // Introduce acciones a OnExit de un estado pasado por su nombre en String
61.     public void setActionInOnExitByState(String stateName, Action [] actionsToAdd){
62.         State state = getState(stateName);
63.         OnExit oex = state.getOnExit();
64.         for(Action action : actionsToAdd){
65.             oex.addAction(action);
66.         }
67.         state.setOnExit(oex);
68.     }
69.
70.     // Introduce una nueva transición a un estado por su nombre en String
71.     public void setTransitionInState(String stateName, Transition transitionToAdd){
72.         State state = getState(stateName);
73.         state.addTransition(transitionToAdd);
74.     }
75.
76.     // Crea una transición con los datos que se le pasan y la introduce en el estado sta
77.     // event --
78.     > nombre del evento que hará saltar la transición (si la condición es true)
79.     // cond --> Condición para que la transición tenga lugar
80.     // targetState --> estado destino de la transición
81.     // actionsToAdd --> Acciones que ejecuta la transición
82.     public void addCustomTransition(String stateName ,String event, String cond, String targetState,
83.     {
84.         Transition t = new Transition();
85.         t.setEvent(event);
86.         t.setCond(cond);
87.         t.setTarget(getState(targetState));
88.
89.         for (Action a : actionsToAdd){
90.             t.addAction(a);
91.         }
92.         setTransitionInState(stateName, t);
93.     }
94.
95.     // Devuelve la lista de transiciones de un estado stateName que tienen como evento e
96.     public List<Transition> getTransitionsByStateAndEvent(String stateName, String eventName){
97.         State s = getState(stateName);
98.         return s.getTransitionsList(eventName);
99.     }
100. }
```

```

97.
98. // Callback del estado A, es llamado cuando entra en este estado
99. public void A() {
100.     LOGGER.info("STATE: A");
101. }
102.
103. // Callback del estado B
104. public void B() {
105.     LOGGER.info("STATE: B");
106. }
107.
108. // Callback del estado C
109. public void C() {
110.     LOGGER.info("STATE: C");
111. }
112.
113. }

```

Bien, ya tenemos la clase principal, ahora vamos a crear una clase que herede de Action, ya que es una clase abstracta, para poder crear nuestras acciones personalizadas, yo lo único que he implementado es un método que imprima en el log un string que se le pasa en el constructor, en este caso será un string informativo del método al cuál se le va a asociar la acción.

La clase CustomAction.java ubicada en la misma ruta que FSM.java es la siguiente:

```

view plain print ?
01. import org.apache.commons.logging.Log;
02. import org.apache.commons.logging.LogFactory;
03. import org.apache.commons.scxml.ErrorReporter;
04. import org.apache.commons.scxml.EventDispatcher;
05. import org.apache.commons.scxml.SCInstance;
06. import org.apache.commons.scxml.SCXMLExpressionException;
07. import org.apache.commons.scxml.model.Action;
08. import org.apache.commons.scxml.model.ModelException;
09.
10. public class CustomAction extends Action{
11.
12.     private static final Log LOGGER = LogFactory.getLog(CustomAction.class);
13.
14.     private static final long serialVersionUID = 1L;
15.     String action;
16.
17.     public CustomAction(String action){
18.         super();
19.         this.action = action;
20.     }
21.
22.     @Override
23.     public void execute(EventDispatcher evtDispatcher, ErrorReporter errRep,
24.         SCInstance scInstance, Log appLog, Collection derivedEvents)
25.         throws ModelException, SCXMLExpressionException {
26.         LOGGER.info(action);
27.     }
28.
29.
30.     public void setAction(String action){
31.         this.action = action;
32.     }
33.
34. }

```

Creo que hace falta explicar poco de la clase, es muy simple.

Ahora vamos a copiar el archivo fsm.scxml que creamos con el editor gráfico, en la carpeta resource, tanto dentro de main como de test, ya que ahora crearemos los test. En TDD lo primero que hay que crear son los test, pero para que quedaran más claros los he dejado para el final de la explicación.

En la misma ruta que teníamos las anteriores clases, pero partiendo ahora de test, vamos a crear la clase FSMTTest.java, con el siguiente contenido:

```

view plain print ?
01. import java.net.URL;
02. import java.util.Collection;
03.
04. import org.apache.commons.scxml.ErrorReporter;
05. import org.apache.commons.scxml.EventDispatcher;
06. import org.apache.commons.scxml.SCInstance;
07. import org.apache.commons.scxml.TriggerEvent;
08. import org.apache.commons.scxml.model.Action;
09. import org.apache.commons.logging.Log;
10. import org.junit.BeforeClass;
11. import org.junit.Test;
12. import org.mockito.InOrder;
13. import org.mockito.Matchers;
14. import org.mockito.Mockito;
15. import static org.mockito.Mockito.spy;
16.
17. import com.autentia.tutorials.fsm.CustomAction;
18. import com.autentia.tutorials.fsm.FSM;
19.
20. public class FSMTTest {
21.
22.     public static final String STATE_A = "A";
23.     public static final String STATE_B = "B";
24.     public static final String STATE_C = "C";
25.
26.     public static final String TRANSITION_AB = "Transicion_AB";
27.     public static final String TRANSITION_BC = "Transicion_BC";
28.
29.     public static final URL urlXML = FSM.class.getClassLoader().getResource("fsm.scxml");
30.
31.     public static TriggerEvent transicionAB = null;
32.     public static TriggerEvent transicionBC = null;
33.
34.     public static boolean eventdata;

```

```

35.
36. private static Action actionTransitionCB1;
37. private static Action actionTransitionCB2;
38.
39. private static CustomAction actionEntryB;
40. private static CustomAction actionExitB1;
41. private static CustomAction actionExitB2;
42. private static CustomAction actionEntryC;
43.
44. public static FSM fsm = null;
45.
46.
47. //Inicializamos las variables que nos van a hacer falta
48. @BeforeClass
49. public static void init(){
50.     fsm = spy(new FSM(urlXML));
51.
52.     actionEntryB = spy(new CustomAction("Action onEntry B"));
53.     actionExitB1 = spy(new CustomAction("Action onExit B1"));
54.     actionExitB2 = spy(new CustomAction("Action onExit B2"));
55.     actionEntryC = spy(new CustomAction("Action onEntry C"));
56.
57.     Action [] actionOnEntryB = {actionEntryB};
58.     Action [] actionOnExitB = {actionExitB1, actionExitB2};
59.     Action [] actionOnEntryC = {actionEntryC};
60.
61.     actionTransitionCB1 = spy(new CustomAction("Action Transition CB1"));
62.     actionTransitionCB2 = spy(new CustomAction("Action Transition CB2"));
63.
64.     fsm.setActionInOnEntryByState(STATE_B, actionOnEntryB);
65.     fsm.setActionInOnExitByState(STATE_B, actionOnExitB);
66.
67.     fsm.setActionInOnEntryByState(STATE_C, actionOnEntryC);
68.
69.     fsm.addActionToTransitionByStateAndEvent(STATE_B, TRANSITION_BC, actionTransitionCB1);
70.     fsm.addActionToTransitionByStateAndEvent(STATE_B, TRANSITION_BC, actionTransitionCB2);
71.
72.     eventdata = true;
73.
74.     fsm.addConditionToTransitionByStateAndEvent(STATE_B, TRANSITION_BC, "_eventdata");
75.
76.     transicionAB = new TriggerEvent(TRANSITION_AB, TriggerEvent.SIGNAL_EVENT, eventdata);
77.     transicionBC = new TriggerEvent(TRANSITION_BC, TriggerEvent.SIGNAL_EVENT, eventdata);
78.
79. }
80.
81. // Comprobamos que las acciones introducidas en OnEntry y OnExit de los estados s
82. @Test
83. public void shouldCallOnEntryAndOnExitInOrder(){
84.
85.     try {
86.
87.         InOrder inOrder = Mockito.inOrder(actionEntryB, actionExitB1, actionExitB2, actionEntryC);
88.
89.         fsm.getEngine().triggerEvent(transicionAB);
90.         fsm.getEngine().triggerEvent(transicionBC);
91.
92.         inOrder.verify(actionEntryB, Mockito.atLeastOnce()).execute(Matchers.any(EventDispatch.class));
93.         inOrder.verify(actionExitB1, Mockito.atLeastOnce()).execute(Matchers.any(EventDispatch.class));
94.         inOrder.verify(actionExitB2, Mockito.atLeastOnce()).execute(Matchers.any(EventDispatch.class));
95.         inOrder.verify(actionEntryC, Mockito.atLeastOnce()).execute(Matchers.any(EventDispatch.class));
96.
97.     } catch (Exception e) {
98.         System.err.println("Error");
99.         e.printStackTrace();
100.    }
101. }
102.
103. // Comprobamos que devuelve correctamente las transiciones de los estados A y B
104. @Test
105. public void shouldGetTransitions(){
106.     assert(fsm.getTransitionsByStateAndEvent("A", TRANSITION_AB) == transicionAB);
107.     assert(fsm.getTransitionsByStateAndEvent("B", TRANSITION_BC) == transicionBC);
108. }
109.
110. //Comprobamos que llama en orden las acciones de las transicion entre B y C
111. @Test
112. public void shouldCallActionsInOrderInTransitionsCB(){
113.
114.     InOrder inOrder = Mockito.inOrder(actionTransitionCB1, actionTransitionCB2);
115.
116.     try {
117.
118.         fsm.getEngine().triggerEvent(transicionAB);
119.
120.         fsm.getEngine().triggerEvent(transicionBC);
121.
122.         inOrder.verify(actionTransitionCB1, Mockito.atLeastOnce()).execute(Matchers.any(EventDispatch.class));
123.         inOrder.verify(actionTransitionCB2, Mockito.atLeastOnce()).execute(Matchers.any(EventDispatch.class));
124.
125.     } catch (Exception e) {
126.         System.err.println("Error");
127.         e.printStackTrace();
128.     }
129. }
130.
131. //Comprobamos que pasa correctamente de un estado a otro al llegar el evento corr
132. @Test
133. public void runInAllStates(){
134.
135.     fsm.addConditionToTransitionByStateAndEvent(STATE_B, TRANSITION_BC, "_eventdata");
136.
137.     try {
138.
139.         assert(fsm.getEngine().getCurrentStatus().equals("A"));
140.         fsm.getEngine().triggerEvent(transicionAB);
141.
142.         assert(fsm.getEngine().getCurrentStatus().equals("B"));
143.

```

```

144.         fsm.getEngine().triggerEvent(transicionBC);
145.         assert(fsm.getEngine().getCurrentStatus().equals("C"));
146.
147.     } catch (Exception e) {
148.         e.printStackTrace();
149.     }
150. }
151. }

```

Al inicio de los métodos hay un pequeño resumen de lo que comprueba cada test. Creo que no tiene mucha complejidad y es bastante legible el código por sí sólo, pero voy a explicar un par de cosas por si no quedan claras.

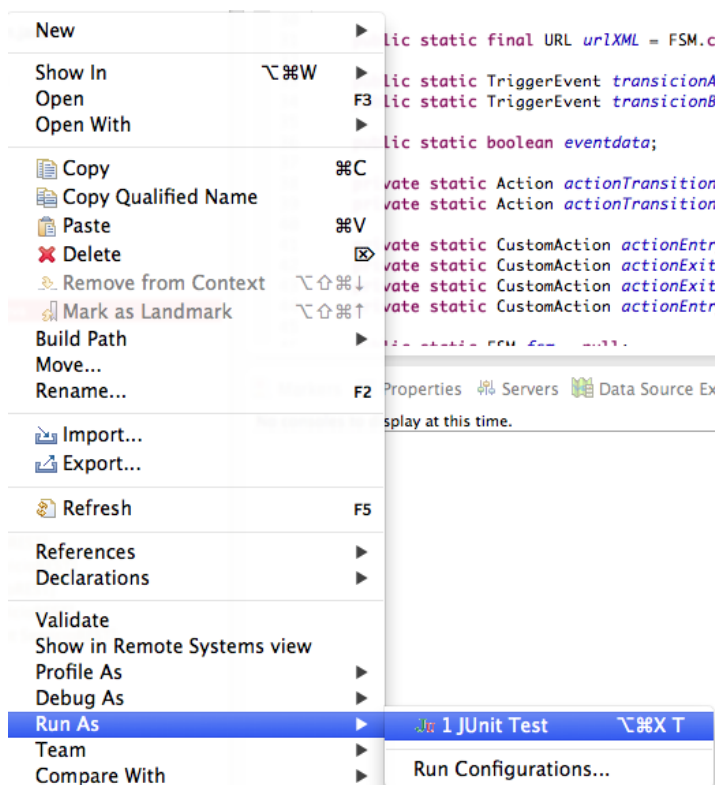
El método spy, incluido con mockito nos va a permitir saber el orden en el que se llaman a los métodos de esos objetos, y los crea llamando al constructor, cosa que con mock no se hace y no podemos seguir el orden de algunos objetos.

Otra cosa a tener en cuenta es la variable booleana eventdata, la cual se le pasa al lanzar el evento, y si os fijáis, llamamos al método de fsm addConditionToTransitionByStateAndEvent, el cual pasamos el estado b, la transicion que queremos configurar y el String que queremos que tenga la condición de esa transición (STATE_B, TRANSITION_BC, "_eventdata"). _eventdata es el nombre de la variable que da Apache SCXML a las variables que vienen desde la transición.

Como vemos líneas más abajo, introducimos la condición eventdata a la transición TransicionCB (transicionBC = new TriggerEvent(TRANSITION_BC,TriggerEvent.SIGNAL_EVENT, eventdata); y con lo que conseguimos que _eventdata que recoge apache SCXML tenga el valor de la variable booleana eventdata(true) con lo que conseguimos que la transición que pasamos sea lanzada dependiendo de la variable eventdata.

Realmente no necesitamos esto, sólo lo he incluido como información para hacer nuestro código más flexible y configurable.

Ahora ejecutamos la clase como JUnit test:



Vemos que pasan los test:



Y la salida tendría que ser algo parecido a esto:

```

Feb 4, 2014 12:04:04 PM com.autentia.tutorials.fsm.FSM A
INFO: STATE: A
Feb 4, 2014 12:04:04 PM com.autentia.tutorials.fsm.CustomAction execute
INFO: Action onEntry B
Feb 4, 2014 12:04:04 PM com.autentia.tutorials.fsm.FSM B
INFO: STATE: B
Feb 4, 2014 12:04:04 PM com.autentia.tutorials.fsm.CustomAction execute
INFO: Action onExit B1
Feb 4, 2014 12:04:04 PM com.autentia.tutorials.fsm.CustomAction execute
INFO: Action onExit B2
Feb 4, 2014 12:04:04 PM com.autentia.tutorials.fsm.CustomAction execute
INFO: Action Transition CB1
Feb 4, 2014 12:04:04 PM com.autentia.tutorials.fsm.CustomAction execute
INFO: Action Transition CB2
Feb 4, 2014 12:04:04 PM com.autentia.tutorials.fsm.CustomAction execute
INFO: Action onEntry C
Feb 4, 2014 12:04:04 PM com.autentia.tutorials.fsm.FSM C
INFO: STATE: C

```

5. Conclusiones.

Hemos visto cómo crear una máquina de estados sencilla que nos ayudará a que nuestras tareas tengan un flujo definido claro y configurable.

[El código en github del proyecto.](#)

Para cualquier duda o aclaración, en los comentarios.

Un saludo.

A continuación puedes evaluarlo:

[Regístrate para evaluarlo](#)

Por favor, vota +1 o compártelo si te pareció interesante

[Share](#) |

Anímate y coméntanos lo que pienses sobre este **TUTORIAL**:

» **Regístrate** y accede a esta y otras ventajas «



Esta obra está licenciada bajo licencia [Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

PUSH THIS

Page Pushers

Community

Help?

no clicks

0 people brought clicks to this page

+ + + + + + + +

powered by [karmacracy](#)

Copyright 2003-2014 © All Rights Reserved | [Texto legal y condiciones de uso](#) | [Banners](#) | [Powered by Autentia](#) | [Contacto](#)

W3C XHTML 1.0

W3C CSS

XML RSS

XML RSD