

# ¿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)



[Home](#) | [Quienes Somos](#) | [Empleo](#) | [Tutoriales](#) | [Contacte](#)

<p><b>Tutorial desarrollado por:</b>  <a href="#">Alberto Carrasco Montenegro</a></p> <p><b>Puedes encontrarme en <a href="#">Autentia</a></b>  <b>Somos expertos en Java/J2EE</b>  <b>Contacta en <a href="mailto:info@autentia.com">info@autentia.com</a></b></p>	
---	--

Descargar este documento en formato PDF [validacionJSF.pdf](#)

[Firma en nuestro libro de Visitas](#)

#### **IntelliJ IDEA**

Advanced JSP Editor for professional developers. Get Trial

#### **JSP Editor**

Edit JSP, XML, DTD, Schema, XSLT & SOAP. Easy-to-Use! Free Trial.

#### **Integrar SOA, WebServices**

Sus datos 3270/5250 en J2EE & Java Integrar CICS/IMS con BEA, CRM

#### **Visual Web Developer 2005**

Dokumente und Downloads rund um die neuen Microsoft Entwickler-Tools.

Anuncios Goooooogle

Anunciarse en este sitio

En [Autentia](#) trabajamos constantemente en el desarrollo de aplicaciones web para nuestros clientes. En este aspecto, investigamos con interés las posibilidades y evolución del framework que parece destinado a suceder (o al menos complementar) a Struts en el futuro: Java Server Faces (JSF). Este framework se apoya en tecnologías que ya utiliza el propio Struts, como por ejemplo, el uso del Commons Validator para gestionar los mecanismos de validación. Hoy os queremos mostrar como integrar y extender en JSF los mecanismos de validación proporcionados por el Commons Validator.

### **1. Aspectos básicos de validación en JSF utilizando Commons Validator**

En este apartado inicial se procederán a explicar algunos aspectos relativos a la validación en JSF utilizando el framework *Commons Validator*. Esto será necesario para entender, tal como se explicará más adelante, las acciones necesarias para extender los mecanismos de validación en JSF.

JSF incorpora por defecto algunos mecanismos básicos para facilitar la validación de formularios. Estos se refieren fundamentalmente a aspectos básicos como la comprobación de inserción de texto, longitud de las entradas o tipos de datos introducidos en los campos de un formulario. Además, estos mecanismos de validación sólo proporcionan servicio en el lado del servidor de las aplicaciones.

Para ampliar los mecanismos de validación, JSF utiliza otros frameworks especializados, como puede ser el *Commons Validator*, al igual que hace Struts.

Para integrar consigo al *Commons Validator*, JSF utiliza el framework *Shale* basado en Struts. Este framework proporciona, tal como se verá más adelante, un par de etiquetas que permiten integrar de forma sencilla el *Commons Validator* en JSF. De esta forma, y análogamente a como lo hace Struts, JSF puede utilizar el *Commons Validator* para usar mecanismos básicos de validación, extender dichos mecanismos y utilizarlos también para validar en el lado del cliente de las aplicaciones.

Los métodos asociados a la validación que realiza el *Commons Validator* se registran en un fichero xml a través de elementos etiquetados como *validator*. No hace falta registrar dichos métodos en un fichero xml concreto; basta con indicar correctamente su ruta y nombre en el *web.xml*, como se indicará más adelante. Un ejemplo básico de la definición de uno de estos métodos podría ser lo siguiente:

```
<validator name="validateWords"
  classname="org.autentia.moreValidation.MyValidation"
  method="maliciousWords"
  methodParams="java.lang.String"
  msg="errors.maliciousWords">
</validator>
```

Como puede observarse, en la etiqueta *validator* que define el método de validación se pueden definir diversos campos. En este caso:

- **name** : Nombre asociado a la regla de validación.
- **classname** : Clase que contiene el método con la lógica de validación correspondiente.
- **method** : Método que contiene la lógica de validación, situado en la clase especificada en *classname*.
- **methodParams** : Parámetros de entrada del método indicado en *method*. Pueden especificarse varios separándolos por comas.
- **msg** : Mensaje de error que debe emitir la validación en caso de error. En este campo se suele especificar una etiqueta asociada a un mensaje en un fichero *.properties* que forme parte de los recursos de la aplicación.

Pueden emplearse mensajes suministrados por defecto por el *Commons Validator*, o bien elaborar mensajes propios, para informar de

errores en el proceso de validación. En tal caso, debe implementarse un fichero `.properties` con los mensajes correspondientes. Por ejemplo, para el caso anterior podría implementarse el fichero `Messages.properties` con el siguiente contenido:

```
errors.maliciousWords={0} contiene palabras maliciosas.
```

Para indicar la utilización de este recurso por parte del *Commons Validator* a fin de comunicar errores en el proceso de validación, debe especificarse el elemento correspondiente etiquetado como `message-bundle` (y descendiente de `application`) en el fichero `faces-config.xml`. En este caso de ejemplo, se supone que el fichero `Messages.properties` se crea dentro del paquete `org.autentia.bundle` el fichero `faces-config.xml` debería contener las siguientes líneas:

```
<application>
  <message-bundle>org.autentia.bundle.Messages</message-bundle>
</application>
```

Para indicar la validación que *Shale* integrará sobre JSF utilizando el *Commons Validator*, deben especificarse en el fichero de configuración `web.xml` los archivos donde se encuentren registrados todos los métodos empleados para la validación, registrados tal como se acaba de indicar.

Es conveniente especificar el archivo que contiene las definiciones de los métodos incluidos por defecto para realizar tareas de validación (`/org/apache/shale/validator/validator-rules.xml`), así como aquel que el usuario implemente para registrar sus propios métodos de validación (por ejemplo, `/WEB-INF/my-rules.xml`). Debe añadirse un elemento etiquetado como `context-param` con el siguiente aspecto:

```
<context-param>
  <param-name>org.apache.shale.validator.VALIDATOR_RULES</param-name>
  <param-value>
    /org/apache/shale/validator/validator-rules.xml,
    /WEB-INF/my-rules.xml
  </param-value>
</context-param>
```

Para aplicar sobre el campo de un formulario una regla de validación creada de la forma que se acaba de indicar, basta con utilizar la etiqueta de *Shale* destinada a tal efecto: `<s:commonsValidator>`. Una especificación sencilla del uso de cierta regla de validación sobre el campo de texto de un formulario, podría tener el siguiente aspecto:

```
<h:inputText id="user" value="#{nameBean.name}">
  <s:commonsValidator
    type="testValidation"
    server="true"
    client="true"/>
</h:inputText>
```

La inclusión de la etiqueta `<s:commonsValidator>` como descendiente de la etiqueta `<h:inputText>` permite utilizar el método asociado a la regla llamada `testValidation` (campo `type`) como mecanismo de validación del contenido de entrada del campo de texto. Esta regla debe encontrarse registrada en un fichero xml de la forma que se indicó anteriormente.

Los campos `server` y `client` indican en que lado de la aplicación se llevará a cabo la validación: cliente o servidor. Para habilitar la validación en cualquiera de estos dos lados, basta con poner el campo correspondiente (`server` o `client`) a `true`.

En caso de querer habilitar la validación en el lado del cliente, debe incluirse en la página que contenga el formulario el código `javascript` que la realice. Este código debe implementarse en el registro de la la regla de validación, dentro de la etiqueta `<javascript>`.

```
<validator name="testValidation"
  classname="org.autentia.moreValidation.MyValidator"
  method="validateWords"
  methodParams="java.lang.String"
  msg="errors.myError">
  <javascript><![CDATA[
    function validateWords(form) {
      ....
```

El código *javascript* implementado dentro de esta etiqueta es el que se copiará en el lado del cliente para realizar la validación que corresponda. Para ello, debe utilizarse la etiqueta `<s:validatorScript>` de *Shale* en la página que contenga el formulario que se desea validar. Del mismo modo, debe invocarse dicha validación en *javascript* desde el campo *onsubmit* de la etiqueta `<h:form>`. Por ejemplo, para habilitar la validación en el lado del cliente de un formulario con *id* igual a *checkForm*, debería escribirse algo como lo siguiente en la *jsp* que corresponda:

```
<h:form id="checkForm" onsubmit="return validateCheckForm(this);">

  <h:inputText id="userName" value="#{ nameBean.name }">
    <s:commonsValidator
      type="testValidation"
      server="true"
      client="true"/>
  </h:inputText>

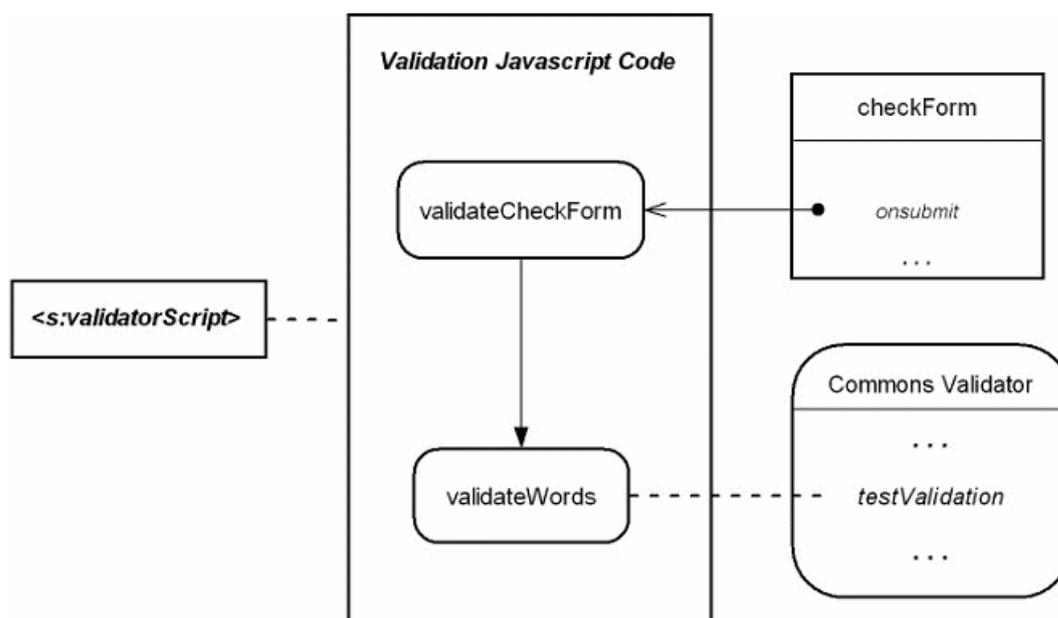
  <h:commandButton id="submit" action="check" value="Comprobar" />

  <s:validatorScript functionName="validateCheckForm"/>

```

El nombre de la función a invocar en el campo *onsubmit* debe ser *validateXXX*, donde *XXX* corresponde con el valor del campo *id* del formulario correspondiente. Esta función la genera el *Commons Validator* y es la que se encargará de llamar a las funciones *javascript* de validación que necesite el formulario, implementadas en el registro de las reglas de validación, tal como se indicó. En este caso, la función *validateCheckForm* llamaría a la función *javascript* asociada a la regla *testValidation* (que es *validateWords*). Si la función *validateXXX* devuelve *false* significa que falló la validación y el formulario no se envía al servidor.

Como se comentó anteriormente, la etiqueta `<s:validatorScript>` copia el código *javascript* de las reglas de validación que utilice el formulario que se desea validar. En el campo *functionName* debe indicarse el nombre de la función *javascript* encargada de la validación del formulario de nombre *validateXXX*, como se explicó antes. Se copiarán en la página aquellas funciones *javascript* que necesite la mencionada función, correspondientes a las reglas de validación que utilice el formulario. En el siguiente esquema se resume esta situación.



Con estas nociones básicas, se está en disposición de entender la especificación del itinerario básico para probar un ejemplo sencillo de integración y extensión de la validación del *Commons Validator* sobre JSF:

1. Implementar un *bean* para un formulario básico.
2. Registrar dicho *bean* en el *faces-config.xml*.
3. Crear los métodos que realizarán la validación extra que se procederá a añadir a la ya incluida por defecto en el *Commons Validator*.
4. Registrar en un fichero xml, tal como se explicó anteriormente, las reglas nuevas de validación que se apoyarán en los métodos que se acaban de crear.
5. Crear, si se desea, mensajes de error asociados a cada regla de validación que se añadirá, en el fichero *.properties* que corresponda.
6. Indicar en el *web.xml* los ficheros xml donde se encuentran registradas las reglas de validación que utilizará la aplicación.
7. Crear una *jsp* con un formulario básico como el que se generó en (1) que pruebe la validación implementada. Compilar, empaquetar y desplegar la aplicación de prueba utilizando un contenedor de aplicaciones web (Tomcat, JBoss o similares).

## 2. Instalación y requisitos

El ejemplo sencillo que se estudiará más adelante en este documento, se desarrolló y ejecutó en un entorno *Windows XP* que disponía de la distribución Java *j2sdk-1.4.2*. Esta distribución de Java puede obtenerse gratuitamente desde la web de *SUN* en el enlace:

<http://java.sun.com/products/archive/j2se/1.4.2/index.html>

Para desarrollar con JSF se utilizó una versión 3.1.0 de Eclipse integrada con Exadel Studio 3.0.5. Los fichero asociados *eclipse-SDK-3.1-win32.zip* y *ExadelStudio-3[1].0.5.exe* pueden obtenerse gratuitamente desde las direcciones:

<http://download.eclipse.org/eclipse/downloads/drops/R-3.1-200506271435/index.php>

<http://www.exadel.com/web/portal/products/Overview>

Se utilizó una versión 1.2.0 del framework *Commons Validator* que puede obtenerse gratuitamente desde el enlace:

[http://jakarta.apache.org/site/downloads/downloads\\_commons-validator.cgi](http://jakarta.apache.org/site/downloads/downloads_commons-validator.cgi)

Puede obtenerse una distribución del framework *Shale* de forma gratuita desde la siguiente dirección:

<http://cvs.apache.org/builds/struts/nightly/struts-shale/>

En caso de problemas de compatibilidad utilizando el framework *Shale*, consultar sus requerimientos en la dirección:

<http://struts.apache.org/struts-shale/>

Para desplegar el ejemplo, se utilizó el contenedor de aplicaciones web de Tomcat. Se utilizó la distribución *jakarta-tomcat-5.0.30.zip*, que puede obtenerse gratuitamente desde el enlace:

<http://tomcat.apache.org/download-55.cgi>

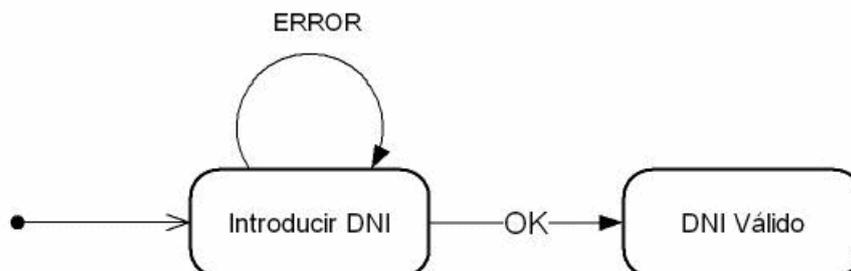
## 3. Un ejemplo sencillo

### 3.1. Idea

Se pretende realizar un ejemplo que ilustre la integración del *Commons Validator* sobre JSF, así como la extensión de sus mecanismos de validación. Para ello se seguirán las directrices proporcionadas en el primer apartado y sólo queda pensar en una regla de validación que amplíe las funciones de validación incluidas por defecto con JSF.

En este caso se ha elegido implementar una sencilla regla que sirva para validar un documento nacional de identidad (DNI) español. La regla que se procederá a implementar comprobará para el DNI considerado que su número y letra son compatibles. La letra en el DNI español se obtiene a través de una sencilla fórmula sobre el número, tal como se mostrará más tarde.

La aplicación que se desarrollará para probar la integración del *Commons Validator* sobre JSF, así como la extensión de sus mecanismos de validación con dicha regla, estará compuesta por un formulario con un campo donde se introducirá un DNI completo (número y letra), a fin de comprobar si es válido o no. Si el DNI no es válido, mostrará un mensaje de error y ofrecerá la posibilidad de volver a introducir un DNI a comprobar. Si el DNI es válido, la aplicación muestra una página informando de dicho suceso. El siguiente diagrama de estados muestra el funcionamiento de la aplicación de prueba que se acaba de especificar:



### 3.2. Creando la nueva regla de validación

Para extender la validación de JSF utilizando el *Commons Validator*, la primera tarea es crear el método que contenga la lógica de

validación necesaria para crear la nueva regla de validación.

En este caso se crea la clase *org.autentia.moreValidation.MyValidator*, y en ella los métodos que contendrán la lógica de validación. El código Java correspondiente de esta clase se muestra a continuación.

```
package org.autentia.moreValidation;

import java.io.Serializable;

import java.util.StringTokenizer;

import java.lang.Integer;

public class MyValidator implements Serializable{

    public static boolean validateWords (String field) {

        //Secuencia de posibles letras que pueden corresponder a un DNI

        String cadena="TRWAGMYFPDXBNJZSQVHLCKET";

        //Capturamos el numero y la letra del DNI

        StringTokenizer tokens=new StringTokenizer(field,"-");

        if(tokens.countTokens() != 2)

            return false;

        Integer numero = new Integer(tokens.nextToken());

        int n = numero.intValue();

        //Se calcula la letra correspondiente al numero del DNI

        int posicion = n % 23;

        String letra = cadena.substring(posicion,posicion+1);

        //Se comprueba si el DNI es correcto

        if( letra.equals(tokens.nextToken()) )

            return true;

        else

            return false;

    }

}
```

El método que invocará la lógica de la validación, y que se indicará en la especificación de la regla de validación más adelante, es *validateWords*. Este método recibe como argumento de entrada un *String* que contendrá el DNI a comprobar. Si toda va bien, devuelve *true*. Si detecta que el formato del DNI no es correcto (tiene que ser [número]-[letra] ) o que el DNI no es válido, devuelve *false*.

Una vez creada la lógica de validación para la nueva regla con la clase *MyValidator*, se procede a registrar el método que la realiza en el fichero xml correspondiente, tal como se comentó anteriormente. Se introduce también el código *javascript* que se emplearía para la validación en el lado del cliente, si se optase por ello. Para ello se utiliza la etiqueta *<javascript>* seguida del código *javascript* necesario, como se explicó antes.

```
<validator

    name="testValidation"

    classname="org.autentia.moreValidation.MyValidator"

    method="validateWords"
```

```

methodParams="java.lang.String"

msg="errors.myError">

<javascript><![CDATA[

    function validateWords(form) {

        //Secuencia de posibles letras de DNI

        var cadena="TRWAGMYFPDXBNJZSQVHLCKET";

        //Capturamos el numero y la letra del DNI

        var dni=form[0].value.split("-");

        if (dni.length != 2)

        {

            alert("DNI no valido. Formato debe ser [Numero-Letra]");

            return false;

        }

        //Se calcula la letra correspondiente al numero del DNI

        var posicion = dni[0] % 23;

        var letra = cadena.substring(posicion,posicion+1);

        //Se comprueba si el DNI es correcto

        if(dni[1] == letra)

            return true;

        else

        {

            alert("Letra de DNI no valida");

            return false;

        }

    }]]> </javascript>

</validator>

```

En el campo *msg* se especifica el mensaje asociado a la etiqueta *errors.myError* en un fichero *.properties* que formará parte de los recursos de la aplicación. En este caso se empleó el fichero *org.autentia.bundle.Messages.properties*, y el mensaje asociado a *errors.myError* para informar de errores en la validación es el siguiente:

```
errors.myError={0} no es un DNI correcto
```

Será necesario registrar este fichero, que contendrá también otros elementos de la aplicación susceptibles de internacionalización, en el *faces-config.xml*, como recurso de la aplicación. Debe insertarse entonces la siguiente línea anidada en el elemento etiquetado como *application*:

```
<message-bundle>org.autentia.bundle.Messages</message-bundle>
```

En este punto ya se tiene generada una extensión del *Commons Validator* utilizado por JSF, que podría reutilizarse en futuros proyectos.

### 3.3. Creación del formulario de prueba

Para poder realizar pruebas con la nueva regla de validación, es necesario crear un formulario para introducirlo en una *jsp*.

Se implementa primero la clase asociada a la *bean* de dicho formulario con sus métodos *set/get* correspondientes. El atributo *dni* contendrá la cadena de texto con el DNI a validar que se introducirá como entrada en el formulario.

```
package org.autentia.bean;

public class DniBean {
    String dni;

    public String getDni() {
        return dni;
    }

    public void setDni(String d) {
        dni = d;
    }
}
```

Se registra esta *bean* en el fichero *faces-config.xml*.

```
<managed-bean>
    <description>DNI de Entrada</description>
    <managed-bean-name>dniBean</managed-bean-name>
    <managed-bean-class>org.autentia.bean.DniBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
        <property-name>dni</property-name>
        <property-class>java.lang.String</property-class>
        <value/>
    </managed-property>
</managed-bean>
```

Se define también la regla de navegación que accionará el formulario si supera la validación. Si no hay error en la validación del formulario, la regla de navegación *checkDNI* conducirá a la aplicación hacia la página *success.jsp*, que mostrará un mensaje confirmando que el DNI analizado es válido.

```
<navigation-rule>
    <from-view-id>/pages/inputDNI.jsp</from-view-id>
    <navigation-case>
        <from-outcome>checkDNI</from-outcome>
        <to-view-id>/pages/success.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

### 3.4. Generación de *jsp*'s para probar la validación del formulario

El último paso antes de iniciar las pruebas consiste en implementar la *jsp* que contendrá el formulario que utilizará la nueva regla de validación (*testValidation*). Su código podría ser algo como lo siguiente:

```

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<%@ taglib uri="http://struts.apache.org/shale/core" prefix="s" %>

<f:loadBundle basename="org.autentia.bundle.Messages" var="Message"/>

<HTML>

<HEAD> <title>Usando Commons Validator en JSF</title> </HEAD>

<body bgcolor="white">

<f:view>

    <h1><h:outputText value="#{Message.intro}"/></h1>

    <h:form id="checkForm" onsubmit="return validateCheckForm(this);">

        <h:outputText value="#{Message.campo}"/>

        <h:inputText id="userDNI" value="#{dniBean.dni}">

            <s:commonsValidator

                type="testValidation"

                server="true"

                client="true"/>

        </h:inputText>

        <h:commandButton id="submit" action="checkDNI" value="Comprobar"/>

        <br><br>

        <h3>

            <font color="red">

                <h:message for="userDNI" styleClass="errors"/>

            </font>

        </h3>

        <s:validatorScript functionName="validateCheckForm"/>

    </h:form>

</f:view>

</body>

</HTML>

```

El campo donde se introducirá un DNI para comprobar si es correcto es aquel etiquetado con `<h:inputText>` con `id` igual a `userDNI`.

Para que la etiqueta `<h:message>` imprima los mensajes de error que produzca la validación de la entrada de texto del formulario, es necesario cargar al principio de la `jsp` el fichero `.properties` que contenga los mensajes de error implementados para la nueva regla de validación. Esto se hace, como puede observarse, con la etiqueta `<f:loadBundle>`.

Puede comprobarse que la `bean` implementada anteriormente para capturar el DNI de entrada en el campo de texto, en la línea:

```
<h:inputText id="userDNI" value="#{dniBean.dni}">
```

Igualmente, puede comprobarse que la regla de navegación implementada anteriormente para mostrar la página que confirme la

validez del DNI (*checkDNI*), se utiliza como acción asociada al botón del formulario:

```
<h:commandButton id="submit" action="checkDNI" value="Comprobar" />
```

En este caso, se ha optado por realizar la validación tanto en el lado del cliente como en el servidor, tal como se indica en la etiqueta `<s:commonsValidator>` asociada al campo de entrada de texto del formulario. El código *javascript* necesario para realizar la validación en el cliente se genera a partir del código implementado para la regla *testValidation*, creada anteriormente. Para copiar dicho código en el lado del cliente a fin de que pueda utilizarlo, se utiliza la etiqueta `<s:validatorScript>`, tal como se explicó anteriormente.

```
<s:validatorScript functionName="validateCheckForm"/>
```

Para hacer efectiva esta validación en el lado del cliente, se edita el campo *onsubmit* en la etiqueta `<html:form>`. Este campo contendrá una llamada a la función *validateCheckForm*, que es la que realizará la validación en el lado del cliente, utilizando el código *javascript* de las reglas de validación que precise el formulario. Si esta función devuelve *true* se enviará el formulario al servidor. Si devuelve *false* el formulario no será enviado al servidor.

```
<h:form id="checkForm" onsubmit="return validateCheckForm(this);">
```

Si sólo se quisiera realizar validación en el lado del servidor, se podría utilizar una versión de la jsp como la siguiente:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://struts.apache.org/shale/core" prefix="s" %>

<f:loadBundle basename="org.autentia.bundle.Messages" var="Message"/>

<HTML>
<HEAD> <title>Usando Commons Validator en JSF</title> </HEAD>
<body bgcolor="white">
<f:view>
    <h1><h:outputText value="#{Message.intro}"/></h1>
    <h:form id="checkForm">
        <h:outputText value="#{Message.campo}"/>
        <h:inputText id="userDNI" value="#{dniBean.dni}">
            <s:commonsValidator
                type="testValidation"
                server="true"
                client="false"/>
        </h:inputText>
        <h:commandButton id="submit" action="checkDNI" value="Comprobar"/>
        <br><br>
    <h3>
        <font color="red">
            <h:message for="userDNI" styleClass="errors"/>
    </h3>
</f:view>
</body>
</HTML>
```

```

        </font>
    </h3>
</h:form>
</f:view>
</body>
</HTML>

```

Como puede verse, se ha eliminado la etiqueta `<s:validatorScript>` y el campo `onsubmit` de la etiqueta `<h:form>`. El campo `client` de la etiqueta `<s:commonsValidator>` se pone a `false` para indicar que no se realizará validación en el lado del cliente.

Se implementa igualmente la página `success.jsp` a la que dará paso la acción del formulario de la `jsp` anterior si el DNI introducido a la entrada es válido.

```

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<f:loadBundle basename="org.autentia.bundle.Messages" var="Message"/>

<HTML>
<HEAD> <title>Usando Commons Validator en JSF</title> </HEAD>
<body bgcolor="white">
<f:view>
    <h3>
        <h:outputText value="#{Message.ok}" /> :
        <h:outputText value="#{dniBean.dni}" />
    </h3>
</f:view>
</body>
</HTML>

```

Con esto ya se dispone de todas las partes necesarias para ejecutar el ejemplo de prueba explicado inicialmente. Sólo queda compilar y empaquetar la aplicación en un fichero `.war`, y desplegarlo en un servidor de aplicaciones web.

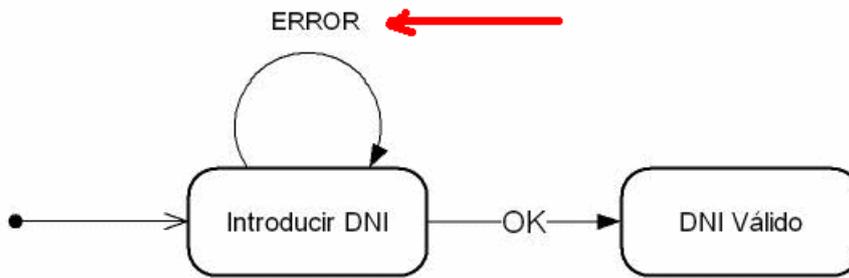
### 3.5. Ejecución

Una vez compilado, empaquetado y desplegado el proyecto en un contenedor de aplicaciones web, la ejecución del ejemplo produce los siguientes resultados. Se mostrará la correspondencia entre el flujo de páginas de la aplicación y el diagrama de estados de la misma que se diseñó al principio de este apartado.

#### 3.5.1. Validación en el lado del cliente

Si se empleara la `jsp` indicada anteriormente para realizar la validación en el lado del cliente y en el servidor, algunas pruebas darían los siguientes resultados:

##### 1. Relleno del formulario (DNI no válido)



## Introduzca un DNI con formato [Numero-Letra]:

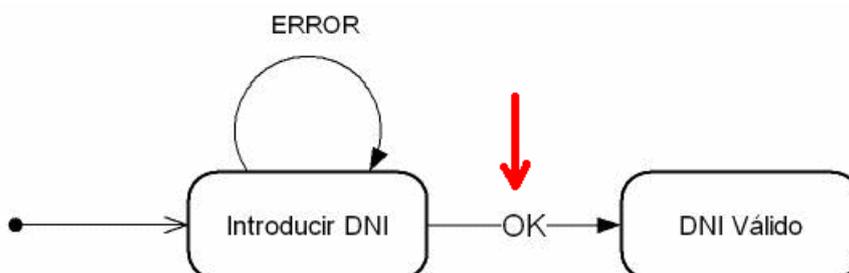
DNI



### 2. Relleno del formulario (ok)

## Introduzca un DNI con formato [Numero-Letra]:

DNI   ←



**El siguiente DNI es correcto : 11111111-H**

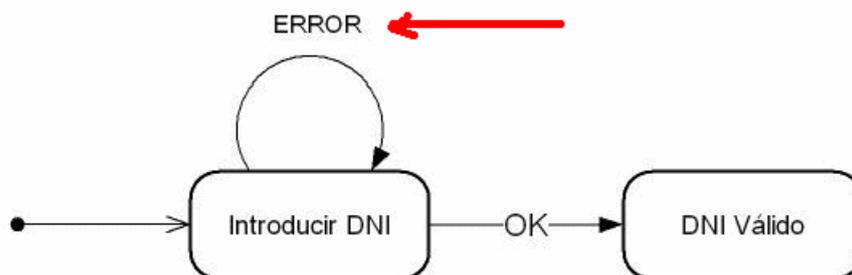
### 3.5.2. Validación sólo en el lado del servidor

Si se empleara la *jsp* indicada anteriormente para realizar la validación sólo en el lado del servidor, algunas pruebas darían los siguientes resultados:

#### 1. Relleno del formulario (DNI no válido)

### Introduzca un DNI con formato [Numero-Letra]:

DNI   ←



### Introduzca un DNI con formato [Numero-Letra]:

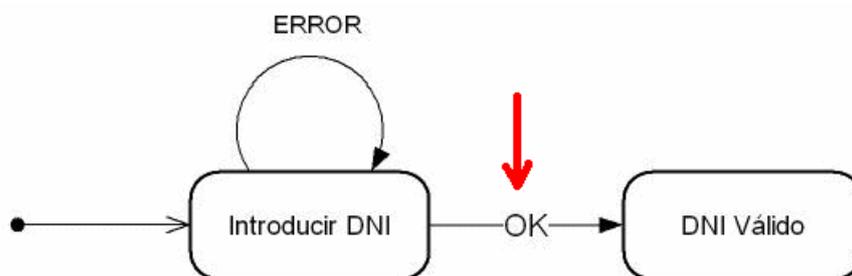
DNI

**11111111-A no es un DNI correcto**

#### 2. Relleno del formulario (ok)

### Introduzca un DNI con formato [Numero-Letra]:

DNI   ←



**El siguiente DNI es correcto : 11111111-H**

#### 4. Conclusiones

Los iniciados en Struts con experiencia en la utilización del *Commons Validator* en tareas de validación, pueden comprobar que extender la validación utilizando dicho framework desde JSF (con ayuda de *Shale*), es un proceso similar al que habría que realizar desde Struts. El coste de aprendizaje en este sentido es bastante bajo.

Como observación final, señalar la importancia de establecer los mecanismos de validación oportunos en cualquier aplicación en general, y en este caso sobre aquellas construidas sobre JSF, a fin de generar software más robusto y fiable.

## 5. Fuentes

- Página del framework *Shale*:

<http://struts.apache.org/struts-shale/features-commons-validator.html>

- Wiki de *Shale*:

<http://wiki.apache.org/struts/ShaleValidation>

## Recuerda

que el personal de [Autentia](#) te regala la mayoría del conocimiento aquí compartido ([Ver todos los tutoriales](#))

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?

**¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?**

[info@autentia.com](mailto:info@autentia.com)

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....

**Autentia = Soporte a Desarrollo & Formación**

soluciones reales para **SU**

[Autentia S.L.](#) Somos expertos en:

**J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ..**  
y muchas otras cosas

## Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	<input type="text"/>
	<input type="button" value="Enviar"/>

## Otros Tutoriales Recomendados ([También ver todos](#))

### Nombre Corto

[Conversión y validación en JSF](#)

[Utilizando JSTL en JSF](#)

[Manejar tablas de datos con JSF](#)

[Introducción a Spring Web Flow](#)

[Upload de ficheros en Struts](#)

[Probando entornos para JSF](#)

### Descripción

En este nuevo tutorial sobre JSF os mostramos como utilizar y extender los mecanismos básicos de conversión y validación

Os mostramos como utilizar la librería estandar de etiquetas en JSF, implementando una sencilla aplicación web

En este tutorial os mostramos un ejemplo de utilización de la extensión del componente DataTable, realizada por la implementación Tomahawk de MyFaces

Spring Web Flow es un módulo de extensión del framework Spring, que facilita la implementación del flujo de páginas de una aplicación web

En este tutorial os mostramos paso a paso como construir una sencilla aplicación de upload de ficheros utilizando Struts

En este tutorial os mostramos con ejemplos como utilizar dos conocidos entornos de desarrollo para JSF: Exadel Studio y Sun Studio Creator

Os mostramos de una forma sencilla y guiada como crear una utilidad de upload de

[Upload de ficheros en JSF](#)

ficheros utilizando JSF

[JSF y comparativa con Struts](#)

Os mostramos los pasos necesarios para empezar a utilizar JSF (Java Server Faces) y su comparación / relación con Struts

[Struts Jakarta](#)

Cuando se ha trabajado creando aplicaciones Java poco a poco se va viendo la necesidad de normalizar los desarrollo. Uno de los Framework (entornos) más extendidos es Struts

[Introducción a Struts Flow](#)

Struts Flow es un módulo de extensión del conocido framework Struts, que facilita la implementación del flujo de páginas de una aplicación web

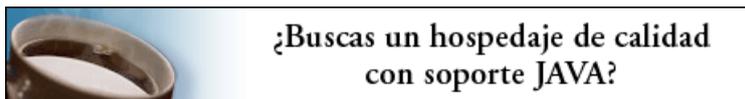
Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.

[Patrocinados por enredados.com .... Hosting en Castellano con soporte Java/J2EE](#)



[www.AdictosAlTrabajo.com](http://www.AdictosAlTrabajo.com) Optimizado 800X600