Diseño e implementación de un marco de trabajo (framework) de presentación para aplicaciones JEE

José María Galindo Haro
ITIG / ITIS

Josep Maria Camps Riba

Fecha de entrega
14-01-2008

Resumen

Internet ha ido desarrollándose con el paso de los años hasta hoy día en el que tenemos un nivel de páginas Web disponibles del orden de millones de ellas. Uno de los factores de crecimiento y expansión del Web ha sido la evolución de la tecnología que sustenta la infraestructura Software. Dentro de las diferentes opciones de desarrollo de los sitios Web "WebSite" la plataforma Java en su edición J2EE Empresarial ofrece un conjunto de herramientas para favorecer la creación de portales Web que da soporte a las demandas de dichas aplicaciones, seguridad, modelo Vista Controlador etc. A parte de este entorno de trabajo han surgido plugins y herramientas para facilitar el desarrollo de las páginas Web. Estas herramientas están enmarcadas dentro de los que se denomina Frameworks. Con la utilización de un Framework de presentación, los desarrolladores ya no es necesario que tengan que un nivel tal alto a nivel técnico, el Framework se encarga de facilitar el desarrollo, estandarizar las páginas.

Este trabajo fin de carrera ha tenido como objetivos el estudio de los diferentes elementos que existen a la hora de construir la parte visual de aplicaciones Web desarrollados sobre la plataforma de construcción de J2EE, los patrones de Diseño de la capa de presentación y lo que se denominan Frameworks de presentación. Y por otro lado a partir de dicho estudio se a realizado la creación de un Framework propio que permita la creación de interfaces para páginas Web creadas para la gestión de manejo de aplicaciones empresariales de una forma óptima, estándar y simplificada.

Índice de contenidos

1. Introducción *10*

1.1. Justificación del TFC y contexto en el que se desarrolla: punto de partida y aportación del TFC. **10**

1.2. Objetivos del TFC. **11**

1.3. Enfoque y método seguido. **11**

1.4. Planificación del proyecto. **12**

Primera Entrega (PEC 2) 29/10/2007 12

Segunda Entrega (PEC 3) 17/12/2007 12

Entrega Final 14/01/2008 12

1.5. Productos obtenidos **12**

1.6. Breve descripción de los otros capítulos de la memoria. **13**

2. J2EE *14*

2.1 J2EE una plataforma de componentes distribuida **14**

2.1.1 ¿Qué es J2EE? 15

2.1.2 Modelo de programación de aplicaciones (*Blueprints*) 16

2.2 La capa intermedia (*middle layer*) **28**

2.2.1 Capa de presentación 28

3. Evolución de las arquitecturas web: MODELOS JSP y Arquitectura MVC *35*

3.1 Modelo JSP 1 Capa **35**

3.2 Modelo JSP 2 Capas **36**

3.3 Arquitectura MVC **36**

4. Frameworks *38*

4.1 Definición **38**

4.2 Características **38**

4.3 Ventajas de un framework **39**

4.4 Desventajas de un Framework **40**

4.5 Opciones en el Mercado **40**

5. Patrones de Diseño *41*

5. 1¿Qué es un Patrón? **41**

5.2 Abstracción de Patrones **41**

5.3 Identificar un Patrón **41**

5. 4 Patrones contra Estrategias **41**

6. Struts *45*

6.1 Introducción **45**

6.1.1 Capacidades de Struts 45

6.2 Visión de Conjunto de una arquitectura 45

6.3 Struts y ámbito 47
6.3.1 Ámbito de petición 47
6.3.2 Ámbito de sesión 48
6.3.3 Ámbito de aplicación 48
6.3.4 Page Scope 48

6.4 Utilizar parámetros URL 49
6.4.1 Reenviar vs redirigir 49
6.4.2 Utilizar un redireccionamiento 49
6.4.3 Utilizar un reenvío 49
6.4.4 ¿Qué es mejor en Struts Reenviar o redirigir? 50

6.5 Los paquetes de componentes de Struts 50
6.5.1 Componentes de controlador Struts 51
6.5.2 Componentes de Modelo Struts 55
6.5.3 Componentes de vista de Struts 55
6.5.4 Soporte de múltiples aplicaciones 61

7. Java Server Faces 62

7.1 Introducción 62
7.1.1 Beneficios de la Tecnología JavaServer Faces 62

7.2 ¿Qué es una Aplicación JavaServer Faces? 63

7.3 Una Sencilla Aplicación JavaServer Faces 63
7.3.1 Pasos del Proceso de Desarrollo 64
7.3.2 Desarrollar los Objetos del Modelo 64
7.3.3 Añadir las Declaraciones del Bean Controlado 65
7.3.4 Crear las Páginas 66
7.3.5 Definir las Navegación por las Páginas 67

7.4 El Ciclo de Vida de una Página JavaServer Faces 68
7.4.1 Escenarios de Procesamiento del Ciclo de Vida de una Petición 68
7.4.2 Ciclo de Vida Estándar de Procesamiento de Peticiones 69
7.4.3 Reconstituir el Árbol de Componentes 70
7.4.4 Aplicar Valores de la Petición 70
7.4.5 Procesar Validaciones 71
7.4.6 Actualizar los Valores del Modelo 71
7.4.7 Invocar Aplicación 71
7.4.8 Renderizar la Respuesta 72

7.5 Modelo de Componentes de Interface de Usuario 72
7.5.1 Las Clases de los Componentes del Interface de Usuario 72
7.5.2 El Modelo de Renderizado de Componentes 73
7.5.3 Modelo de Conversión 75
7.5.4 Modelo de Eventos y Oyentes 75
7.5.5 Modelo de Validación 76

7.6 Modelo de Navegación 76

7.7 Creación del Bean Controlado 76

7.8 Configuración de la Aplicación 76

8. Spring Web MVC Framework 78

8.1 Introduccion 78
8.1.1Capacidades de Spring Web MVC 78

8.2 El DispatcherServlet **79**

8.3 Controladores **82**
8.3.1 AbstractController y WebContentGenerator 83
8.3.2 Otros controladores simples 84
8.3.3 El MultiActionController 84
8.3.4 Controladores de comando 84

8.4 Mapeos manejadores **85**
8.4.1 BeanNameUrlHandlerMapping 85

8.5 Vistas y su resolución **86**
8.5.1 Vistas de resolución – el interfaz ViewResolver 86
8.5.2 Encadenar ViewResolvers 87
8.5.3 Redireccionar a vistas 88

8.6 Utilización de los valores locales del cliente **89**

8.7 Usando Temas de Spring **89**
8.7.1 Introducción 89
8.7.2 Definiendo temas 89
8.7.3 Resolutores de Temas 90

8.8 Soporte en varias partes (multipartes) de Spring **90**

8.9 Usando la librería de tag para formularios de Spring **90**

8.10 Manejar excepciones **91**

9. Comparación de los tres Frameworks. ***92***

10. Usabilidad ***93***

10.1 Introducción **93**

10.2 Objetivos y proceso de creación **93**
10.2.1 La Guía de Estilos en las aplicaciones 93
10.2.2 Objetivos de las Normas de Estilo 93

10.3 Diseño centrado en el usuario **94**
10.3.1 Usabilidad ISO 9241-11 94
10.3.2 Modelo de calidad en desarrollo de Software 94
10.3.3 Diseño Centrado en el Usuario en sistemas interactivos 95
10.3.4 Proceso de trabajo ISO 13407 96
10.3.5 Beneficios del Diseño Centrado en el Usuario 96
10.3.6 El comportamiento del usuario 97

11. Arquitectura Básica Struts ***99***

12. Framework de Desarrollo JUoc ***101***

12.1 Capa Cliente **103**

12.2 Capa de presentación **103**

12.3 Resto de Capas **103**

13. Arquitectura Básica ***104***

13.1 Componentes Controlador **104**
13.1.1 ActionServlet 104
13.1.2 RequestProcessor 106
13.1.3 Plugins 109

13.1.4 Fichero de Recursos multiidioma 110
13.1.5 Action 110

13.2 Componentes Modelo 113

13.3 Componentes Vista. 113
13.3.1 ActionForm 113
13.3.2 Paso de Componentes Vista a Objetos del Negocio 117

14. Servicio de Pantallas 122

14.1 Introducción 122
14.1.1 Mecanismos JSP Estándar 122
14.1.2 Template 122
14.1.3 Construcción Pantalla a devolver 124

14.2 Instalación y Configuración 125
14.2.1 Uso de la librería 125
14.2.2 Uso de Tiles con Struts 125

14.3 Definiciones 126
14.3.1 Extender Definiciones 126
14.3.2 Nombre de Definición en el Forward de los Action 127

14.4 Layouts del Framework 127
14.4.1 Layout Básico 127
14.4.2 Layout Frames 129

15. Servicio de Multidiioma 131

15.1 Selección Idioma Usuario 131
15.1.1 Inicio por Idioma por Defecto del Navegador 131

15.2 Ficheros de Recursos 132

15.3 Tags Multiidioma 133

15.4 Imágenes Multiidioma 133

15.5 Presentación de Mensajes Multiidioma 133

15.6 Obtención de Literales desde las Clases y JSPs 133

15.7 Definiciones Necesarias para el Framework 134

16. Servicio de Tags. 135

16.1 Configuración Librerías 135

16.2 Uso de Formularios 136

16.4 Tags HTML 139
16.4.1 Aspectos Generales 139
16.4.2 Tag HTML 139
16.4.3 Uso de frames 140
16.4.4 Tag Formulario 140
16.4.5 Tag Imagen Submit 142
16.4.6 Aviso de Confirmación de Borrado 143
16.4.7 Tag TextField 144
16.4.8 Indicación de Campos Requeridos 146
16.4.9 Ayuda Sensitiva 146
16.4.10 Selección de Valor en una Nueva Ventana 148
16.4.11 Tag Double 149

16.4.12 Tag TextArea 150
16.4.13 Tag Fecha 151
16.4.15 Tag Radios 153
16.4.16 Tag Options 153
16.4.17 Tag Option 155
16.4.17 Tag Checkbox 155
16.4.18 Tag Img 156
16.4.19 Tag Link 156
16.4.20 Tag Imagen Texto Link 157
16.4.21 Tag Password 157
16.4.22 Tag File 158
16.4.23 Tag Modo 158
16.4.24 Tag Línea 158
16.4.25 Tags de Menú 159

17. Servicio de Validación 161

17.1 Configuración Inicial 161

17.2 Formularios 161
17.2.1 Formulario Estático 162
17.2.2 Formulario Dinámico 162

17.3 Configuración de Validadores 162

17.4 Definición de las validaciones a realizar para los formularios 162
17.4.1 Declaración de los Formularios 163
17.4.2 Declaración de Validaciones Generales 165

17.5 Código de validación en la página JSP 166

18. Aplicación de Ejemplo 167

19. Futuras líneas de desarrollo/mejoras en el Framework. 169

20. Conclusiones. 170

Bibliografía. 172

Anexos 173

Índice de figuras

Figura 2-1 Componentes, contenedores, servicios y servidores de aplicaciones. 16
Figura 2-2.Componentes, contenedores, servicios y de la plataforma J2EE. 18
Figura 2-3.Esquema simplificado de los contenedores y componentes de la plataforma J2EE. 19
Figura 2-4.Capas en las que se puede dividir una aplicación J2EE. 21
Figura 2-5. Componentes, módulos y aplicaciones J2EE. 25
Figura 2-6. Ciclo de vida completo de una aplicación J2EE 26
Figura 2-7. La capa intermedia. 28
Figura 2-8. Paradigma de petición respuesta. 30
Figura 2-9.Ciclo de vida de un servlet. 30
Figura 2-10 Ciclo de vida una JSP 33
Figura 3-1Evolución de las arquitecturas web. 35
Figura 3-2 Arquitectura Model-1 35
Figura 3-3. Arquitectura Model-2 36
Figura 3-4. Modelo Vista Controlador: Esquema de Participación 37
Figura 5-1. Esquema de Patrones J2EE. 43
Figura 6-1. Capas funcionales de aplicación 46
Figura 6-2 El framework Struts esta incrustado dentro de la capa Web. 47
Figura 6-3. El modelo petición / respuesta HTTP. 47
Figura 6-4. Un redireccionamiento ocasiona que el navegador dispare una nueva petición 49
Figura 6-5. Durante un reenvío, la petición es enviada a un recurso secundario del servidor sin que el cliente se entere. 50
Figura 6-6. El framework Struts se compone de 7 paquetes principales. 50
Figura 6-7. El framework de Strus utiliza un servlet de Java para manejar las responsabilidades del controlodor. 51
Figura 6-8. El método execute() de la clase Action es llamado por el controlador 52
Figura 7-1. Esquema general de envío de una petición en Java Server Faces. 62
Figura 7-2. Ciclo de vida de una Petición-Respuesta Estándar en Java Server Faces. 70
Figura 7-3. Esquema de acciones en una página con Java Server Faces. 70
Figura 8-1. Flujo de trabajo (a alto nivel), en el proceso de una petición en el Spring Web MVC ... 79
Figura 8-2. Jeraquía del Contexto en el Spring Web MVC 80
Figura 10-1. Categorías relevantes para la calidad del Software. 95
Figura 10-2. Proceso de trabajso ISO 13407. 96
Figura 10-3. Proporción de los gastos en las distintas fases de un producto Software 97
Figura 10-4. Ejes de definición de la usabilidad. 98
Figura 11-1. Esquema general de Struts 100
Figura 11-2. Diagrama de Secuencia Petición y Tratamiento Framework 100
Figura 12-1. Arquitectura lógica de aplicaciones construidas con JUoc. 101
Figura 12-2. Servicios que ofrece JUoc 102
Figura 13-1. Mapeo del patron ‘*.do´ de las urls al ActionServet. 106
Figura 13-2.Diagrama de Secuencia RequestProcessor 107
Figura 13-3. Jerarquía BaseRequestProcessor Arquitectura 108
Figura 13-4. Clases Action del Framework 112
Figura 13-5 Diagrama de clases Formulario 117
Figura 13-6. Clase Formulario y Clase ValueObjec 118
Figura 13-7. Diagrama de Clases BeanUtils 120
Figura 14-1. Proceso General de Creación de Pantallas 125
Figura 16-1. Ejemplo de Modo de Formulario 137
Figura 16-2. Comprobación de Borrado 144
Figura 16-3. Ayuda sensitiva 147
Figura 16-4. Estilos Ayuda sensitiva 147
Figura 16-5. Servicio Presentación: Tags - Selección valor ventana 149
Figura 16-6. Uso del contendor textArea 151
Figura 16-7. Tag Fecha: Conversión Automática 151

Figura 17-1 Jerarquía del ActionForm del framework (BaseActionForm)..... 162
Figura 18-1 Estructura de ficheros de la aplicación de test. 167

Capítulo 1.

1. Introducción

1.1. Justificación del TFC y contexto en el que se desarrolla: punto de partida y aportación del TFC.

Internet ha ido desarrollándose con el paso de los años hasta hoy día en el que tenemos un nivel de páginas Web disponibles del orden de millones de ellas. Uno de los factores de crecimiento y expansión del Web ha sido la evolución de la tecnología que sustenta la infraestructura tanto Hardwere como Software. Desde el punto de vista de Software en los inicios podíamos encontrar contenidos estáticos de texto y enlaces entre ellos, con pocos contenidos vistosos. Hoy en día las páginas Web pueden soportar contenido dinámico, mantener el estado entre flujos de pantallas, potentes gráficos, motores que gestionan los contenidos dinámicos de forma óptima (AJAX), conexiones seguras y autenticadas etc. Los servicios que se pueden obtener de Internet son muchísimos, hacer la compra, realizar operaciones bancarias, votar, contratar servicios de todo tipo, comprar música, incluso portales que facilitan relaciones personales.
Estas ofertas/demandas provocan que las empresas tengan un conjunto de necesidades bastante diferentesde las que tenían hace unos años. En la era de la información, toda empresa que quiera ser competitiva en su sector tiene que ofrecer sus servicios a sus clientes, empleados, *partners*, etc. Para las empresas que quieren ofertar servicios las características de este tipo de productos es muy atractiva son servicios disponibles 24 horas al día durante los 365 días al año, no es necesario un Software especifico en el ordenador del cliente, ni conocimientos elevados de informática para poder hacer uso de la página web, y el número de potenciales clientes que pueden acceder a los servicios es enormemente elevado. Con apenas unos costes reducidos en Hardware, la pagina Web puede ser visitada desde casi cualquier lugar del planeta, no solo ya con un ordenador, sino con dispositivos móviles se puede visualizar páginas Web. Es un escaparte abierto al mundo para cualquier Empresa o persona.
Dentro de una aplicación Web una de las partes más importante es la parte visual, otras partes también importantes son la disponibilidad, la velocidad de respuesta a sus acciones etc, pero nos vamos a centrar en la parte visual, en lo que lo que ve el usuario y con lo que interactúa. La parte visual de una aplicación Web consiste en un conjunto de páginas Web, que contiene texto, imágenes, diversos colores, elementos multimedia: clip de video, sonido etc. Así la imagen que ofrezca al usuario nuestro sitio Web “WebSite”, será la que reciba de nuestra empresa. Homogeneidad en las estructuras, interactuación con flujos sencillo, dan valor añadido a nuestro WebSite.
Para poder crear páginas Web con todo este tipo de aplicaciones se necesita que las tecnologías que subyacen a estas aplicaciones estén en constante evolución para adaptarse a las necesidades cada vez mayores que se requieren para los WebSite. Nos encontramos así con un conjunto de diferentes tecnologías a la hora de poder crear nuestros espacios Web que si bien algunas de ellas son mas eficaces para cierto tipos de desarrollos otras resultas más optimas para otros. Por ejemplo, un portal Web de un Banco ofrece distinto aspecto y funcionalidad que un Portal Web de un grupo de Rock, utilizándose distintas tecnologías, J2EE y Flash respectivamente. Pero aunque existen diferentes tecnologías, existen estándares a nivel internacional y organismos oficiales que certifican el cumplimiento de estos estándares, certificando de esta manera parámetros de calidad en los WebSites. Opciones de aumento de tamaño de letra, navegabilidad sencilla, no abuso de colores, son aspectos a tener en cuenta para la visualización correcta de páginas Web por parte de personas con algún tipo de discapacidad visual por ejemplo.
Las empresas que han visto el potencial de ofrecer servicios a través de Internet han empezado a demandar tecnología que favorezca la creación de portales que cubran sus necesidades. El perfil técnico para el desarrollo de estas aplicaciones Web es un perfil altamente demandado por ellas. Además los desarrollos Web están en constante evolución, manteniéndose y adaptándose a cada momento a lo que demandan tanto los usuarios como la propia empresa.
La plataforma Java en su edición J2EE Empresarial ofrece un conjunto de herramientas para favorecer la creación de portales Web que da soporte a las demandas de dichas aplicaciones, seguridad, modelo Vista Controlador etc. A parte de este entorno de trabajo han surgido plugins y herramientas para facilitar el desarrollo de las páginas Web. Estas herramientas están enmarcadas dentro de los que se denomina Frameworks. Con la utilización de un Framework de presentación, los desarrolladores ya no es necesario que tengan que un nivel tal alto a nivel técnico, el Framework se encarga de facilitar el desarrollo, estandarizar las páginas. Hace de caja negra a la tecnología que sustentan los desarrollos, cumpliendo estándares de calidad tanto los internos de la empresa como los externos, optimizan los recursos hardware, facilitan la navegabilidad etc.

El Trabajo Fin de carrera “Diseño e implementación de un marco de trabajo (Framework) de presentación para aplicaciones JEE” va a consistir en dos tareas. Por un lado el estudio de los diferentes elementos que existen a la hora de construir la parte visual de aplicaciones Web desarrollados sobre la plataforma de construcción de J2EE, los patrones de Diseño de la capa de presentación y lo que se denominan Frameworks de presentación. Por otro lado a partir de dicho estudio se realizara la creación de un Framework propio que permita la creación de interfaces para páginas Web creadas para la gestión de manejo de aplicaciones empresariales de una forma óptima, estándar y simplificada. Además se realizara una aplicación de ejemplo que utiliza dicho Framework.
El FrameWork que se va a crear, tendrá como destino aplicaciones de Intranet. Este tipo de aplicaciones son utilizadas por los empleados de la Empresa para realizar labores de manipulación de la información, labores de gestión de introducción de datos, los CAU’s de las empresas, gestiones bancarias etc. Este tipo de aplicaciones tiene un diseño común y estándar, en cuanto a componentes, eventos de los mismos, navegabilidad, el resultado es una imagen de marca para la Empresa. De esta manera los usuarios necesitan menos sesiones formativas en el uso de los interfaces, se crean de manera que el usuario se canse lo menos posible en su utilización (visualmente, psicamente) , las validaciones son comunes , así un usuario sabe que al dar un botón Borrar, le saldrá un mensaje de si realmente está seguro de realizar el borrado, en todo la aplicación etc.
El FrameWork tendrá un conjunto de componentes visuales que permitan crear páginas de gestión: tablas, botones, cajas de texto, radiobutton etc. También establecerá la navegavilidad entre las pantallas y la comunicación con la capa de negocio. Estos componentes visuales serán como objetos con propiedades y atributos.

1.2. Objetivos del TFC.

Los objetivos de este proyecto fin de carrera son duales.
En una primera etapa se realizara un estudio de los modelos y tecnologías que existen ahora mismo a la hora de desarrollar productos Web. Estos desarrollos se realizan en arquitecturas multicapa (veremos porque se ha decidido desarrollarlos en este tipo de arquitecturas) en lo que se denominan Modelos Vista Controlador. Así estudiaremos en que consiste este tipo de modelos y como afrontan el diseño de productos Web. En este sentido se incluirá el estudio de los diferentes patrones de diseño que sustentan este tipo de modelos.
Como tecnología subyacente estudiaremos la plataforma de desarrollo empresarial J2EE propuesta por Sun Microsystems que define un estándar para el desarrollo de aplicaciones empresariales multicapa. Estudiaremos las posibilidades que ofrece de componentes modulares estandarizados, el conjunto de servicios de estos componentes, y como gestiona automáticamente muchas de las funcionalidades o características complejas que requiere cualquier aplicación empresarial (seguridad, transacciones, etc).
Una vez entendamos globalmente como se desarrollan aplicaciones Web bajo J2EE, centraremos el estudio en los elementos de presentación que intervienen en este tipo de desarrollos. Estudiaremos los FrameWorks Struts y JavaServer Faces, viendo como se han creado, que componentes nos ofrecen para poder crear nuestras páginas, como abstraen los conceptos etc. En definitiva que ventajas ofrecen a la hora del desarrollo y sus limitaciones.

En la segunda etapa una vez estudiados todos puntos anteriores la creación de un Framework de presentación propio será el resultado práctico de todo lo aprendido. Este Framework deberá ofrecer una serie de componentes reutilizables que facilitaran el desarrollo en la capa de presentación de aplicaciones Web. De las distintas posibilidades de Framework crearemos uno para desarrollos de gestión empresariales. Estos desarrollos comparten en su mayoría elementos comunes y una navegación parecida. La creación de un Framework para facilitar este tipo de desarrollos es altamente práctico.
Una vez desarrollado el FrameWork se creara una pequeña aplicación Web con dicho FrameWork. Está aplicación nos servirá para comprobar si el uso del FrameWork resulta sencillo, y si cubre las necesidades planteadas.
Analizaremos si las páginas obtenidas como resultado son las esperadas, en cuanto a diseño, calidad en la presentación. Veremos si los componentes admiten versatilidad en las opciones de configuración para poder ofrecer la máxima variedad. Cuestiones de tamaño o peso de la página y tiempo de carga de la misma serán evaluadas. También se examinara la navegabilidad entre las páginas, así como la conexión con la parte de negocio de los elementos de presentación.
Una vez realizada la evaluación se realizara un punto de futuras líneas de desarrollo / mejoras en el Framework.

1.3. Enfoque y método seguido.

Se ha partido de una lista de tareas o puntos de consecución de objetivos muy concretos establecidos a partir de un plan de trabajo que comprendía las grandes líneas generales del proyecto:

- el análisis de tecnología actual,
- posteriormente 3 ejemplos de frameworks,
- análisis de cómo se tiene que crear interfaces web
- discusión de la tecnología a utilizar
- creación del framework
- uso del framework para la creación de páginas web
- mejoras del framework.

Con estas líneas generales se ha profundizado en cada punto siempre desde una visión del producto final que queríamos obtener, en resumen, servicios y componentes que ayuden al programador en la creación de Web Site empresariales. Así en cada punto hemos obtenido la información necesaria y desechado la superflua que entendíamos que no formaban parte del objetivo. El punto de mejoras del framework siempre ha estado presente desde el principio, porque el desarrollo de un framework de estas características puede llegar a ser muy extenso.

1.4. Planificación del proyecto.

Acorde con el calendario establecido de entregas dividiremos la carga de trabajo de la siguiente manera:

Primera Entrega (PEC 2) 29/10/2007

En esta primera entrega se realizara el estudio de cómo se desarrollan aplicaciones Web hoy en día. Estudiaremos la tecnología actual y más concretamente J2EE.
Una vez obtenida la visión global nos centraremos en la parte de presentación de los desarrollos Web, estudiando los patrones de diseño más comunes que encierran.
Después estudiaremos dos Frameworks concretos Struts y JavaServer Faces realizando una evaluación de los mismos, viendo las características que ofrecen, ventajas, puntos a mejorar. Este punto servirá para ver como tenemos que confeccionar nuestro propio Framework.
También se redactara los puntos iniciales del proyecto

Segunda Entrega (PEC 3) 17/12/2007

En esta entrega se confeccionara el Framework. Se realizara el diseño del mismo estableciendo las bases de lo que vamos a ofrecer y lo desarrollaremos .Se crearan los componentes que van a conformar nuestro Framework.
Se redactara la documentación en cuanto al diseño llevado a cabo y el producto ofrecido (en forma de documentación de uso de los componentes).

Entrega Final 14/01/2008

En esta entrega se realiza la creación de la aplicación de ejemplo de uso del Framework.
Se detallaran los pasos que hemos seguido con el Framework para seguir el desarrollo de la misma. A partir de ella se evaluara tanto la realización del portal como los resultados obtenidos, obteniéndose las conclusiones de utilidad de uso del Framework creado.
Se redactaran los siguientes puntos:
. Futuras líneas de desarrollo/mejora en el Framework.
. Conclusiones del proyecto
. Glosario
. Bibliografía.
. Anexos
También se realizara un repaso general de la memoria, redactando la dedicatoria y agradecimientos, el resumen, y el índice de contenidos.

1.5. Productos obtenidos

Se ha obtenido como producto el Framework de presentación objetivo de este proyecto, empaquetado con todas las clases, ficheros de configuración, y la aplicación de ejemplo de uso. Con estos recursos se puede utilizar el Framework baja una arquitectura de Struts en la creación de Web Sites Empresariales.

1.6. Breve descripción de los otros capítulos de la memoria.

El documento está distribuido en base a los siguientes capítulos:

- **Capítulo 2. J2EE.** En este capitulo vemos el panorama actual de los desarrollos que se están efectuando y demandando. Se estudian las características que se solicitan de las aplicaciones Software actuales y como Java en su versión empresarial J2EE ha creado una metodología y herramientas estándar para favorecer la creación de aplicaciones Web.
- **Capítulo 3. Evolución de las arquitecturas Web: MODELOS JSP y Arquitectura MVC.** En este capitulo se estudia con detalle la evolución que han tenido los desarrollos Web desde los modelos mas primarios hasta el estándar actual MVC.
- **Capítulo 4. Frameworks.** En este capitulo se da la definición de lo que es un Framework viendo sus características, ventajas y desventajas.
- **Capítulo 5. Patrones de Diseño.** Los patrones de diseño han surgido como herramientas a la hora de desarrollar el software para facilitar la claridad, robustez, modularización en los desarrollos software. Veremos como los framework hacen uso de los patrones.
- **Capítulo 6. Struts.** Estudiaremos el Framework de Struts viendo como se ha desarrollado, que nos ofrece a la hora de realizar aplicaciones Web con él etc.
- **Capítulo 7. Java Server Faces.** Estudiaremos el Framework que ofrece Java Server Faces viendo como se ha desarrollado, que nos ofrece a la hora de realizar aplicaciones Web con él etc.
- **Capítulo 8. Spring Web MVC Framework**. Estudiaremos el Framework MVC Web de Spring viendo como se ha desarrollado, que nos ofrece a la hora de realizar aplicaciones Web con él etc.
- **Capítulo 9. Comparación de los tres FrameWorks.** Una vez presentados los tres Frameworks, se realiza una comparación de los mismos y se decide de que Framework se partirá para realizar nuestro propio Framework.
- **Capitulo 10**. **Usabilidad**. Un framework de presentación ofrece facilidades para la creación de la parte visual de una aplicación Web. Veremos lo importante que es realizar un interfaz óptimo, fácil de usar etc, para los usuarios que lo van utilizan. De esta manera examinaremos los conceptos de creación de interfaces usables.
- **Capítulo 11. Arquitectura Básica Struts.** En este capitulo se repasara el tipo de arquitectura que presenta el Framework de Struts.
- **Capítulo 12. Framework de Desarrollo JUoc.** Este capitulo expone los servicios que ofrece el Framework desarrollado JUoc.
- **Capítulo 13. Arquitectura Básica.** Este capitulo detalla que servicios se han creado a partir de los existentes en Struts para facilitar la comunicación entre los clientes Web y el servidor, extendiendo las clases Action, ActionForm etc.
- **Capítulo 14. Servicio de pantallas.** Este capítulo muestra los servicios de pantalla ofrecidos por el FrameWork, esto es, la distribución de las áreas de trabajo de las aplicaciones que se desarrollen bajo el Framework.
- **Capítulo 15. Servicio de Multidioma.** Este capitulo muestra como se ha desarrollo el tema de la internalización de aplicaciones en el Framework.
- **Capítulo 16. Servicio de Tags.** En este capitulo se presentan los Tags creados, elementos que facilitan la construcción de los JSP, que ofrecerá el Framework.
- **Capítulo 17. Servicio de Validación.** Este capitulo muestra como se han desarrollados los servicios que ofrece el Framework con respecto a mecanismos de validación de la información que viaja a través de las pantallas hacia la capa de negocio.
- **Capítulo 18. Aplicación de Ejemplo.** En este capitulo veremos la creación de páginas web a partir del Framework desarrollado.
- **Capítulo 19. Futuras líneas de desarrollo/mejoras en el Framework.** Un desarrollo como el propuesto tiene una gran magnitud, con lo que en este punto veremos aquellos servicios que se pueden añadir al Framework desarrollado.
- **Capítulo 20. Conclusiones.** En este capitulo se realizara una evaluación de lo que ha sido la elaboración de este proyecto fin de carrera.

Capítulo 2.

2. J2EE

2.1 J2EE una plataforma de componentes distribuida

Actualmente, las empresas tienen un conjunto de necesidades bastante diferentes de las que tenían hace unos años. En la era de la información, toda empresa que quiera ser competitiva en su sector tiene que ofrecer sus servicios a sus clientes, empleados, *partners*, etc.

Por ejemplo, hoy todas las entidades financieras permiten a sus clientes operar por Internet (incluso hay entidades que sólo operan por Internet), ofreciendo servicios que van desde la apertura de una cuenta de ahorro hasta la consulta de los movimientos asociados a una tarjetea de crédito.

Estos servicios y los sistemas que los proporcionan tienen que cumplir una serie de características:
• Alta disponibilidad. Los servicios que se ofrecen no pueden dejar de funcionar. Se tienen que proporcionar mecanismos para asegurar que no dejen de funcionar y, si lo hacen, que el tiempo de interrupción sea mínimo.
• Seguros. Se tiene que garantizar que no haya accesos no autorizados a los servicios y se deben establecer políticas de acceso para los diferentes tipos de usuarios de estos servicios.
• Fiables. Los servicios tienen que ser el máximo de fiables y libres de errores. Se debe ofrecer mecanismos de detección y diagnosis de errores.
• Escalables. Si aumenta la carga del sistema, tiene que ser fácil añadir servidores o bien ampliar los que ya tenemos para dar la misma calidad de servicio sin tener que modificar las aplicaciones existentes.
• Mantenibles. Se tiene que poder añadir servicios a los sistemas y modificar los existentes fácilmente.
• Portables. Se tiene que poder cambiar de plataforma de una manera no traumática. Un cambio de plataforma no puede implicar volver a implementar todos los servicios.
• Rápidos de desarrollar y de desplegar. Se tiene que poder desarrollar y ofrecer servicios a los usuarios del sistema de manera ágil y rápida. Hoy en día, el famoso *time-to-market* es vital para las empresas.
• Fácilmente integrables con los sistemas existentes. Raramente se desarrollarán servicios nuevos partiendo de cero. Lo normal será integrar los nuevos servicios con servicios o desarrollos ya existentes, y lo tenemos que poder hacer fácilmente.

Estas características son comunes a todos los servicios que queremos ofrecer. La mejor forma de obtenerlas es utilizando un conjunto de servicios de bajo nivel que las proporcionen. Éstos se pueden conseguir de dos maneras: utilizando una plataforma de desarrollo que facilite todos estos servicios o cargando estas tareas de bajo nivel y bastante complejas en los desarrollado-res de los servicios de negocio. Esta es la diferencia básica entre utilizar una plataforma de desarrollo empresarial y no hacerlo. En el segundo caso, los desarrolladores perderán mucho tiempo implementando servicios de bajo nivel que tienen poco que ver con los servicios de negocio que hay que ofrecer (y que son realmente el objetivo de la empresa).

Por ejemplo, los desarrolladores del servicio de consulta de cuentas de una aplicación de banca electrónica no se tendrían que preocupar de desarrollar el código necesario para que la aplicación de consulta de cuentas no permita a un usuario consultar cuentas que no sean suyas. Los desarrolladores sólo deberían definir las políticas de acceso y dejar la implementa-ción de éstas en el servicio de seguridad de la plataforma.

Por lo tanto, necesitamos una plataforma que ofrezca un conjunto de servicios a los arquitectos y a los desarrolladores para facilitar el desarrollo de aplicaciones empresariales.

Java 2 Enterprise Edition (J2EE) es un ejemplo de plataforma de desarrollo empresarial que tiene como objetivo primordial hacer más sencillo y ágil el desarrollo y la puesta en marcha de aplicaciones empresariales.

J2EE consigue su objetivo definiendo una arquitectura estándar con los siguientes elementos:
1) Modelo de programación de aplicaciones (Blueprints)
2) Plataforma J2EE
3) Test de compatibilidad J2EE
4) Implementación de referencia

La implantación de la Arquitectura J2EE tiene como objetivo mejorar la eficiencia en el desarrollo, ejecución y mantenimiento de las aplicaciones de para desarrollos empresariales.

2.1.1 ¿Qué es J2EE?

J2EE es una plataforma de desarrollo empresarial (propuesta por Sun Microsystems en el año 1997) que define un estándar para el desarrollo de aplicaciones empresariales multicapa.

J2EE simplifica el desarrollo de estas aplicaciones basándolas en componentes modulares estandarizados, proporcionando un conjunto muy completo de servicios a estos componentes y gestionando automáticamente muchas de las funcionalidades o características complejas que requiere cualquier aplicación empresarial (seguridad, transacciones, etc.), sin necesidad de una programación compleja.

J2EE es una plataforma abierta y estándar para desarrollar y desplegar aplicaciones empresariales multicapa con *n*-niveles, basadas en servidor, distribuidas y basadas en componentes.

Así pues, J2EE se enmarca dentro de un estilo arquitectónico heterogéneo, y aglutina distintas características correspondientes a estilos arquitectónicos en capas o niveles, cliente-servidor, orientada a objetos distribuidos e, incluso, orientada a servicios .

Los conceptos básicos y los puntos clave que hay detrás de la plataforma J2EE:

• J2EE es una plataforma abierta y estándar. No es un producto, sino que define un conjunto de estándares que todos los contenedores deben cumplir para comunicarse con los componentes. En algunos ámbitos, se dice que J2EE es una especificación de especificaciones.
• J2EE define un modelo de aplicaciones distribuido y multicapa con ***n***-niveles. Con este modelo, podemos dividir las aplicaciones en partes y cada una de estas partes se puede ejecutar en distintos servidores. La arquitectura J2EE define un mínimo de tres capas: la capa cliente, la capa intermedia
y la capa de sistemas de información de la empresa (EIS - *Enterprise Information Systems*).
• J2EE basa las aplicaciones empresariales en el concepto de componentes modulares y estandarizados. Este concepto está muy vinculado al concepto contenedor. Los contenedores son entornos estándar de ejecución que proporcionan un conjunto de servicios a los componentes que ahorran mucho trabajo a la hora de desarrollar los componentes.

Por ejemplo, el servicio de transacciones nos permitirá definir un componente con comportamiento transaccional sin tener que codificar ni una línea de código de transacción. El desarrollador indicará el comportamiento transaccional del componente en un descriptor XML y el contenedor donde se despliegue el componente se encargará de llamar al servicio de transacciones de la plataforma para implementar este comportamiento.

J2EE dentro de la plataforma Java 2

J2EE es el resultado de un esfuerzo realizado por Sun Microsystems para alinear las diferentes tecnologías y API Java existentes en una plataforma unificada de desarrollo de aplicaciones empresariales.

Sun ha organizado la plataforma Java 2 en tres dominios o áreas diferentes:
Micro Edition (J2ME), Standard Edition (J2SE) y Enterprise Edition (J2EE).
• Java 2 Micro Edition (J2ME): plataforma de desarrollo para dispositivos móviles (PDA, teléfonos móviles, *paggers*, etc.), con capacidad para ejecutar Java. Se trata de una versión reducida de Java que tiene en cuenta las limitaciones de memoria, capacidad y rendimiento de estos dispositivos.
• Java 2 Estandar Edition (J2SE): ésta es la plataforma de desarrollo de Java que todo el mundo conoce, y que permite ejecutar desarrollos hechos en Java en cualquier ordenador que tenga instalada una máquina virtual Java.
• Java 2 Enterprise Edition (J2EE): plataforma Java para desarrollar y desplegar aplicaciones empresariales que requieren un conjunto de características complejas como seguridad, transaccionalidad, robustez, alta disponibilidad, etc.

J2EE -> JEE Hay que tener en cuenta que las próximas versiones de la plataforma Java han simplificado sus nombres. El “2” se elimina del nombre y también se elimina el número y el punto de la versión. Con esto, la siguiente versión de la plataforma Java para desarrollar aplicaciones empresariales es Java Platform, Enterprise Edition 5 (Java EE 5).

Cada una de las tres plataformas son un subconjunto de la siguiente en un ámbito conceptual. De éstas, J2EE es una plataforma focalizada en el desarrollo de aplicaciones empresariales y se ha definido tomando como base la plataforma J2SE.

2.1.2 Modelo de programación de aplicaciones (*Blueprints*)

La plataforma J2EE utiliza un modelo de aplicación distribuida, basada en componentes y multicapa con *n*-niveles.
Los *Blueprints* para la plataforma J2EE describen recomendaciones de diseño para dividir una aplicación J2EE en capas y decidir qué componentes hay que poner en cada capa. Definen el modelo de programación que deben seguir de las aplicaciones que se desarrollan bajo la plataforma y todo un conjunto de buenas prácticas a la hora de utilizar la plataforma.

J2EE propone una arquitectura multicapa como estilo arquitectónico para las aplicaciones que se desarrollen bajo esta plataforma. Esto quiere decir que para crear una aplicación típica J2EE es necesario dividirla en capas, desarrollar los componentes que sean necesarios y colocarlos en la capa correspondiente.

El modelo de programación de aplicaciones de J2EE, se basa en los conceptos de componente, contenedor y servicio y establecer la relación que hay entre éstos y el modelo de programación de aplicaciones J2EE.

Componentes, contenedores, servicios y servidores de aplicaciones

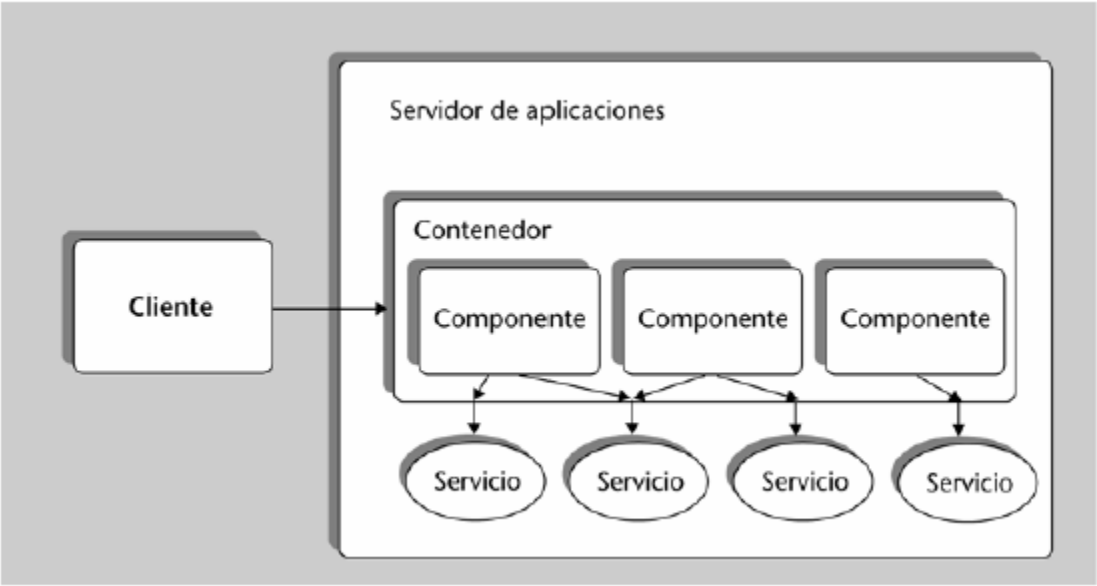


Figura 2-1 Componentes, contenedores, servicios y servidores de aplicaciones.

Una aplicación J2EE estará formada por un conjunto de componentes que se ensamblan y despliegan en contenedores J2EE para ofrecer a los usuarios la funcionalidad deseada.

Los contenedores son piezas de software que ofrecen acceso a los servicios a los componentes que están desplegados dentro del contenedor.

Los servidores de aplicaciones son piezas de software que implementan los servicios que ofrecen los contenedores a los componentes. Los contenedores forman parte de los servidores de aplicaciones.

La mayoría de las aplicaciones empresariales que queremos desarrollar tienen que utilizar un conjunto de servicios de bajo nivel idénticos: gestión del ciclo de vida de los componentes, seguridad, transacciones, balanceo de carga, gestión de recursos, etc.

Una primera opción es que cada aplicación implemente estos servicios. Sin embargo, puesto que estos servicios se repiten en todas las aplicaciones que desarrollamos, es conveniente definirlos e implementarlos una sola vez y dejar que todas las aplicaciones puedan hacer uso de los mismos.

Las aplicaciones J2EE se construyen ensamblando componentes y desplegándolos en un contenedor. Las aplicaciones están formadas por componentes que se ejecutan dentro de contenedores. Los contenedores proporcionan un entorno de ejecución y el acceso a un conjunto de servicios de bajo nivel a los componentes que forman la aplicación.
El proceso de ensamblaje de un componente consiste en empaquetar en un formato estándar todo lo que forma el componente (clases, interfaces, descriptores, etc.) para poder hacer su despliegue.
El despliegue de un componente es la fase de instalación del componente en el contenedor correspondiente. En esta fase, se configuran los servicios del contenedor que necesita el componente. La configuración de los servicios se suele hacer de manera declarativa, con ficheros XML que se empaquetan con los componentes.

Por ejemplo, un componente puede configurar la seguridad de manera declarativa al ser desplegado en el contenedor y decir qué usuarios o grupos de usuarios están autorizados a ejecutar qué métodos del componente.

En un modelo de componentes y contenedores, los componentes tienen la lógica de negocio y los contenedores les proporcionan un entorno de ejecución y todo un conjunto de servicios de bajo nivel.

Es importante tener en cuenta que en un modelo de componentes y contenedores los clientes interactúan con los contenedores en lugar de hacerlo con los componentes directamente. Los contenedores interceptan las llamadas a los componentes, ejecutan las tareas que se han definido de manera declarativa para el componente y le pasan el control para que ejecute la lógica programada por esta llamada.
Para que los contenedores puedan gestionar el ciclo de vida, proporcionar un entorno de ejecución y ofrecer un conjunto de servicios a los componentes, se tienen que definir una serie de normas (un contrato estándar) que los componentes y los contenedores deben cumplir.

Por ejemplo, los *servlets*, un tipo de componente de la plataforma J2EE que se despliega dentro de un contenedor web, definen un contrato con el contenedor web según el cual todas las peticiones de usuario hacia el componente ejecutarán un método llamado *service()*.

J2EE define un modelo de componentes y contenedores abierto y estándar. Esto quiere decir que los contratos entre los componentes, los contenedores y los servicios que tienen que proporcionar los define una especificación estándar.
De esta manera, se consigue que cualquier fabricante pueda desarrollar contenedores capaces de ejecutar componentes J2EE; sólo tiene que cumplir los contratos estándar e implementar los servicios requeridos por la especificación. Las plataformas de software que implementan estos contenedores y servicios se llaman servidores de aplicaciones y hay muchos fabricantes diferentes de los mismos, algunos comerciales, otros de código libre.

La manera que tiene Sun de certificar que una implementación es compatible con la especificación de J2EE es haciendo que el fabricante del servidor de aplicaciones pase un conjunto de tests denominados *Tests de Compatibilidad*.
La siguiente figura muestra todos los componentes, contenedores y servicios que podemos encontrar en la plataforma J2EE:

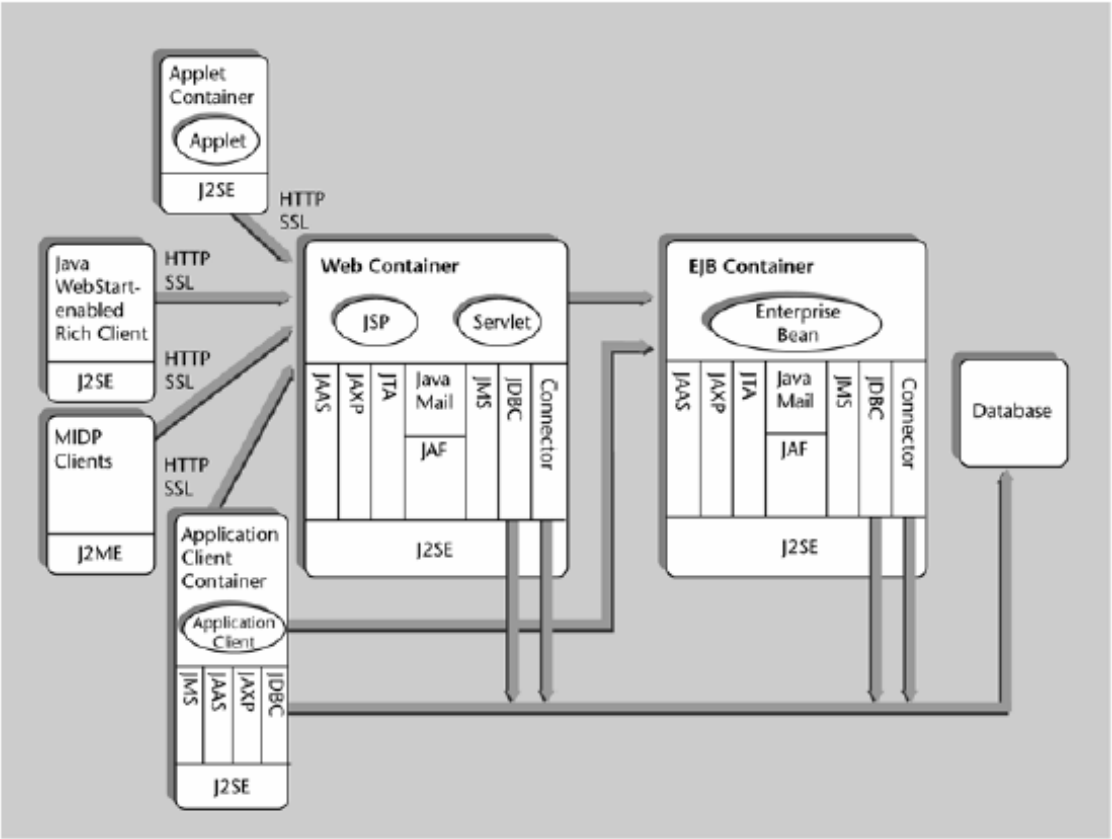


Figura 2-2.Componentes, contenedores, servicios y de la plataforma J2EE.

Componentes

En el contexto particular de J2EE, se definen estos componentes:

• Componentes cliente. Un componente cliente es un programa que se ejecuta en un entorno cliente, ya sea como *applet* o bien como aplicación Java *stand-alone.*

• Componentes web. Un componente web es una pieza de software que genera una respuesta a una petición HTTP. J2EE define dos tipos de componentes web, los ***servlets*** y las ***Java Server Pages*** (JSP).

• Componentes de negocio. Un componente de negocio es una pieza de software que implementa algún concepto de la lógica del dominio particular de negocio en el que se está desarrollando la aplicación. J2EE define los ***Enterprise Java Beans*** (EJB) como componentes de negocio, y pueden ser de tres tipos: EJB de sesión, EJB de entidad y EJB de mensaje.

Contenedores

En el contexto particular de J2EE, se definen los contenedores siguientes:

• Contenedor de aplicaciones. Contiene aplicaciones Java *stand-alone*. Lo encontramos en las máquinas cliente con aplicaciones Java.

• Contenedor de ***applets***. Proporciona un entorno de ejecución por *applets*. Lo encontramos en los navegadores de las máquinas cliente.

• Contenedor web. Proporciona un entorno de ejecución para los componentes web que define la plataforma J2EE, *servlets* y JSP. Lo encontramos en el servidor.

• Contenedor de EJB. Proporciona un entorno de ejecución para los EJB, los componentes de negocio que ofrece J2EE. Lo encontramos en el servidor.

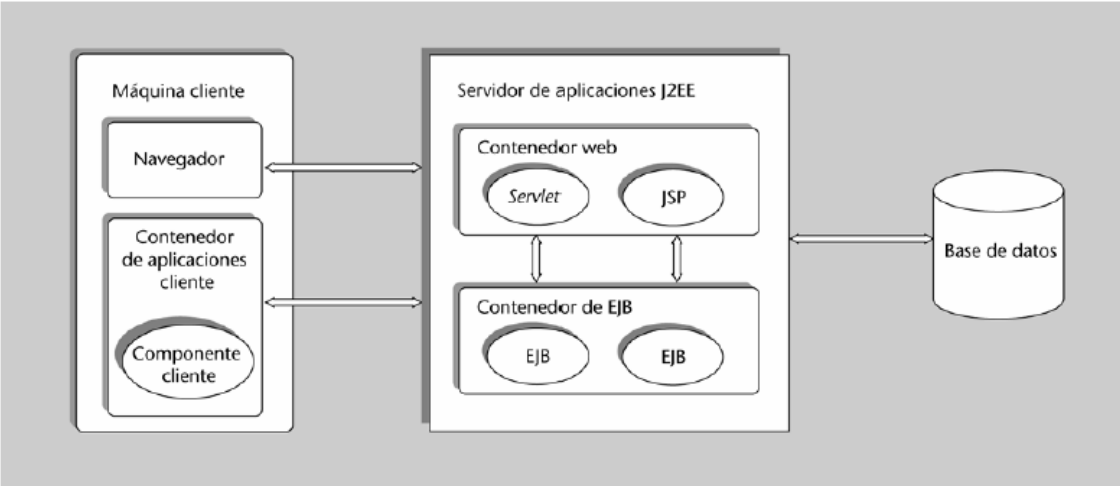


Figura 2-3.Esquema simplificado de los contenedores y componentes de la plataforma J2EE.

Servicios

Los contenedores proporcionan a los componentes un conjunto de servicios que pueden ser configurables o no configurables.

• Los servicios configurables permiten a los desarrolladores especificar el comportamiento de los componentes en tiempo de despliegue de la aplicación. Entre éstos, podemos citar los servicios de transacciones, de seguridad y el servicio de nombres.

• Los contenedores también ofrecen servicios no configurables, como la gestión del ciclo de vida de los componentes, la gestión de los *pools* de conexiones a recursos externos, el acceso a las API de J2EE, etc.

Los contenedores acceden a los servicios mediante las API estándar de cada servicio que define la especificación de J2EE. Por ejemplo, el acceso al servicio de transacciones se hace mediante la API *Java Transaction API* (JTA).

API

Application Program Interface (API) son las interfaces que proporciona un sistema de software (componente, servicio, aplicación, etc.) para que los clientes puedan interactuar. Por ejemplo, la API de JDBC (API para acceder a bases de datos relacionales) nos ofrece un conjunto de métodos que los clientes pueden utilizar para interactuar con una base de datos relacional (abrir una conexión, hacer una consulta, hacer una actualización,etc.).

Servidores de aplicaciones

Los servidores de aplicaciones J2EE son piezas de software que implementan los contenedores, servicios y API que define la plataforma J2EE. Hacen de “cola” entre todos estos elementos, y proporcionan un entorno de gestión, despliegue y ejecución integrado para las aplicaciones empresariales desarrolladas bajo la plataforma J2EE.

Los servidores de aplicaciones hacen que los componentes, contenedores y servicios estén sincronizados y puedan trabajar conjuntamente para proporcionar una serie de funcionalidades. En los servidores de aplicaciones residirán los componentes empresariales, bien sean objetos distribuidos accesibles remotamente, componentes web, páginas web o, incluso, *applets*.

Es importante remarcar que cualquier aplicación J2EE se tiene que desplegar en un servidor de aplicaciones para que los clientes puedan interactuar con ella.

Cualquier servidor de aplicaciones que implemente las especificaciones de la plataforma J2EE tiene que ofrecer todas las tecnologías, API y servicios que estén incluidos como obligatorios en la especificación. Ya hemos visto que esta premisa es clave, ya que proporciona al desarrollador un conjunto de servicios estándar disponibles y le permite centrarse en la lógica de negocio, sin preocuparse de implementar los servicios de bajo nivel.

Un servidor de aplicaciones J2EE tendrá normalmente un contenedor de *servlets*, un contenedor de EJB, un sistema de mensajería y muchas herramientas que nos ayudarán a aumentar la productividad en el desarrollo de aplicaciones.

El hecho de que J2EE se base en especificaciones públicas hace, tal y como hemos visto, que existan múltiples implementaciones de servidores de aplicaciones. Actualmente, el mercado de los servidores de

aplicaciones es uno de los mercados de productos software más activos y hay muchas empresas que lanzan sus productos y luchan por ofrecer cada vez más prestaciones. También existen varias implementaciones de servidores de aplicaciones de código libre que no tienen nada que envidiar a sus homónimos comerciales.

Como ejemplos de servidores de aplicaciones, podemos mencionar:

- JBoss
- Apache Geronimo
- Sun Java System Application Server
- Oracle Application Server
- BEA WebLogic
- etc.

La diferencia que hay entre los distintos servidores de aplicaciones aunque todos tengan que implementar el mismo conjunto de especificaciones reside en que cada servidor pueda ofrecer más o menos servicios de valor añadido, cada servidor implementa las especificaciones a su manera y esto hace que algunos sean más rápidos, otros más robustos, etc.

Hasta ahora hemos visto una breve introducción a J2EE y dentro del modelo de programación de aplicaciones hemos analizado los conceptos de componente, contenedor, servicio y servidor de aplicaciones. Ahora pasaremos a ver bajo qué estilo arquitectónico podemos enmarcar las aplicaciones que desarrollamos con la plataforma J2EE.

Arquitectura multicapa con *n*-niveles

En este apartado veremos qué quiere decir que una aplicación J2EE es una aplicación multicapa con *n*-niveles, analizaremos con detalle cuántas capas define el modelo de aplicaciones J2EE, cuáles son, qué contenedores hay en cada capa y qué componentes podemos desplegar en cada uno de estos contenedores.

El modelo de aplicaciones que define J2EE se enmarca dentro de un estilo arquitectónico heterogéneo que combina características de diferentes estilos, y se centra en un estilo cliente-servidor, basado en componentes y organizado en capas o niveles.

J2EE extiende la arquitectura típica en tres capas o niveles para definir una arquitectura en ***n***-capas. Esta extensión se hace de manera natural, considerando que la capa intermedia se puede subdividir en el nivel lógico en diferentes capas, cada una con unas responsabilidades diferenciadas.

El ejemplo más típico para una arquitectura multicapa en *n*-niveles es una arquitectura con un servidor web que permite conectar la aplicación a Internet. Podemos ver el servidor web como la cuarta capa, la capa web, separada a nivel lógico de la capa que implementa la lógica de negocio. Ya veremos que la capa web no sólo se encarga de servir y encaminar las peticiones web, sino que puede tener una lógica de presentación compleja.

Una aplicación J2EE se puede dividir en tres capas de alto nivel que, normalmente, se subdividen en cinco capas lógicas.

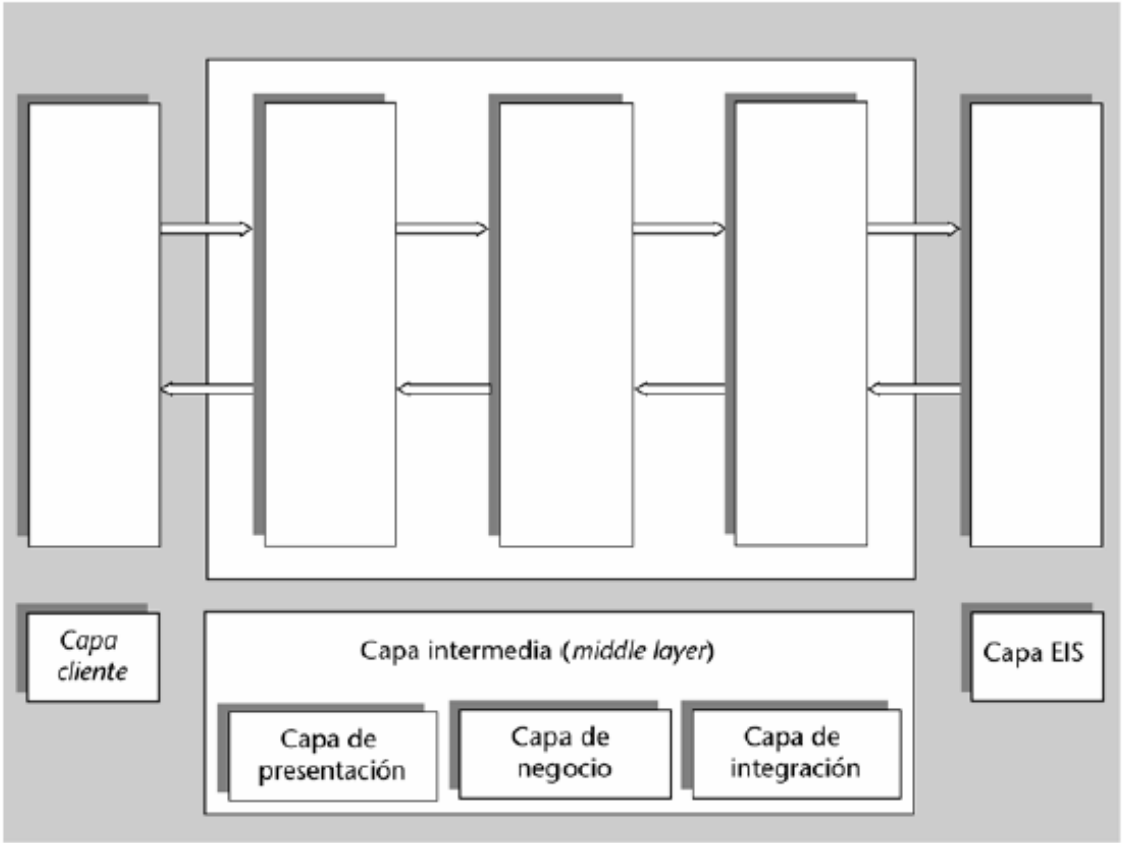


Figura 2-4.Capas en las que se puede dividir una aplicación J2EE.

• **Capa cliente**. Contiene los componentes que se ejecutan en las máquinas cliente. Por ejemplo, páginas HTML que se ejecutan en los navegadores de los usuarios.

• **Capa intermedia** (*middle layer* en el nivel físico, podemos tener la capa de presentación y la capa de negocio en la misma máquina física o bien en máquinas diferentes) . Normalmente, contiene el procesamiento de la lógica de negocio, situada entre la capa cliente y la capa EIS. Los componentes de esta capa se ejecutan en los diferentes contenedores que forman el servidor de aplicaciones. En la arquitectura J2EE, podemos dividir la capa intermedia en tres subcapas formadas por componentes distintos que se ejecutan en contenedores diferentes y que tienen funcionalidades distintas:

– Capa de presentación. Controla la lógica de interacción entre el usuario y la aplicación, así como la forma con la que se presenta la información. Contiene los componentes de presentación que se ejecutan en el contenedor web del servidor de aplicaciones.

– Capa de negocio. Contiene el código y las reglas de negocio para nuestra aplicación. J2EE define los componentes de negocio como EJB que se ejecutan en el contenedor de EJB del servidor de aplicaciones.

– Capa de integración. Capa en la que se hacen las tareas de integración de la aplicación con los diferentes sistemas con los que tiene que interactuar (bases de datos, sistemas *legacy*, etc.). Uno de los componentes de la capa de integración que define J2EE son los EJB de entidad, que también se ejecutan en el contenedor de EJB del servidor de aplicaciones.

• **Capa EIS**. En esta capa, encontramos el resto de los componentes del sistema de información que hay en la empresa. Por ejemplo, contiene los componentes de la capa de datos que residen en un servidor de datos (como podrían ser las tablas de una base de datos relacional).

En el nivel físico, los componentes de la capa cliente se ejecutan en máquinas cliente, los componentes de la capa de presentación, negocio e integración se ejecutan en el servidor J2EE (es posible que este servidor esté formado por varias máquinas) y los componentes de la capa EIS se ejecutan en el servidor EIS.

Esta separación de la capa intermedia en dos o tres capas lógicas no tiene porqué implicar una separación física. Los componentes de la capa de presentación se pueden ejecutar en la misma máquina que los componentes de negocio y los de integración, o bien en máquinas separadas. Lo habitual es que los componentes de negocio y los componentes de integración se ejecuten en la misma máquina física, mientras que los componentes de presentación lo hagan en otra máquina.

Capa cliente

J2EE soporta muchos tipos diferentes de clientes que se comunicarán con la capa intermedia para proporcionar a los usuarios la funcionalidad de negocio deseada.
Estos clientes se pueden ejecutar en varios dispositivos, como pueden ser teléfonos móviles, ordenadores personales, PDA, etc. Ahora veremos qué tipos de clientes soporta la plataforma J2EE y cómo se comunica cada uno de ellos con la capa intermedia.
Una aplicación J2EE puede tener cuatro tipos fundamentales de clientes:
• Las miniaplicaciones (*applets*)
• Los clientes web (*thin client*)
• Los clientes MIDlet
• Los clientes de escritorio, ya sean clientes Java o clientes no Java
Las miniaplicaciones (*applets*) son programas hechos en Java que se ejecutan en la máquina virtual que tienen los navegadores y que se descargan de los servidores para ser ejecutados. Normalmente, se descargan por HTTP y se ejecutan incrustados en una página web.
La principal utilidad de las miniaplicaciones es proporcionar una riqueza visual y operativa a la interfaz de usuario que sería muy difícil de conseguir con una interfaz con HTML.

(*) Usuario Nos referimos a **usuario** en el sentido amplio de la palabra. Pueden ser tanto usuarios humanos como otros sistemas software.

Los clientes web son páginas HTML o XML enviadas por el servidor al navegador y que éste se encarga de redireccionar. Las páginas HTML o XML que forman la interfaz de usuario para un cliente web las genera dinámicamente un componente web de la capa de presentación, y el servidor de aplicaciones envía el resultado de esta generación al navegador que le ha hecho la petición. La comunicación entre el cliente y el servidor se hace, normalmente, en una o varias peticiones HTTP.

(*) Los *applets* no sólo se pueden ejecutar incrustadas en el navegador, sino que lo pueden hacer en todas las aplicaciones o dispositivos que soporten el modelo de programación de los *applets*.

El cliente web es el tipo de cliente por excelencia en las aplicaciones J2EE.

Los clientes MIDlet son pequeñas aplicaciones Java basadas en la tecnología MIDP (*Mobile Information Device Profile*) que se ejecutan en dispositivos móviles. Este tipo de cliente utiliza J2ME como entorno de ejecución para comunicarse con la capa intermedia. Los podemos ver como una aplicación Java que se ejecuta en un dispositivo móvil.
Una aplicación MIDP se empaqueta en un fichero JAR, que puede estar instalado *a priori* en el dispositivo móvil o bien se puede descargar dinámicamente de un servidor.

Los clientes Java se ejecutan en su propio contenedor, en una máquina virtual instalada en el aparato del cliente. Las aplicaciones cliente se empaquetan con un JAR y se tienen que instalar explícitamente en la máquina del cliente. Los clientes Java acceden directamente a los componentes EJB de la capa de negocio del servidor y normalmente utilizan RMI-IIOP como protocolo de comunicaciones.

La manera más usual de comunicar un cliente no Java con un componente de servidor desarrollado con J2EE es utilizando servicios web.

Es muy importante hacer una buena elección del tipo de cliente que proporcionáis con la aplicación, ya que es la “cara” de la aplicación, la parte con la que interactuará el usuario. Hay que conseguir que el cliente escogido reúna el conjunto de características idóneas para el tipo de aplicación que estáis desarrollando.

Capa de presentación

La capa de presentación puede localizarse tanto en el lado del cliente para las aplicaciones de escritorio como en el lado del servidor, dentro de un contenedor web, para las aplicaciones web. Independientemente de si la tenemos en el cliente o en el servidor, la capa de presentación se encargará de controlar la generación del contenido que se enviará al usuario para que éste interactúe con la aplicación. La capa de presentación se comunica con lógica de negocio para conseguir los datos y ensamblarlos, proporcionando de esta manera las vistas adecuadas a cada perfil de usuario.

La capa de presentación de una aplicación web proporciona acceso a la capa de negocio de la aplicación a los clientes que acceden por HTTP, y genera las vistas con las que interactuarán los clientes. Puesto que nos centramos en la capa de presentación de las aplicaciones web, podemos denominar a la capa de presentación capa web.

(*) No es obligatorio ... que la capa web llame a la capa de negocio. En aplicaciones simples, puede pasar que el mismo componente web tenga codificada la lógica de negocio.

En una arquitectura por capas, la capa web se encarga de recibir las peticiones HTTP de los clientes, invocar la lógica de negocio adecuada y generar la respuesta que se envía al cliente. La capa web puede generar cualquier tipo de contenido, aunque normalmente es HTML o XML.
Para proporcionar las funcionalidades anteriores es necesario desarrollar, empaquetar y desplegar lo que se denomina aplicación web. Una aplicación web está formada por un conjunto de componentes, contenido estático y ficheros de configuración que se empaquetan (se ensamblan) y despliegan como una unidad. Las aplicaciones web se empaquetan en ficheros WAR con una estructura estándar y se despliegan en un contenedor web de la plataforma J2EE para que los clientes tengan acceso a las mismas.

El contenedor web es el encargado de gestionar el ciclo de vida de los componentes que forman la aplicación web, pasar las peticiones de los clientes a los componentes y dar acceso al contexto de ejecución.

La especificación de la plataforma J2EE define el contrato entre el contenedor web y el componente web, y define de esta manera el ciclo de vida de los componentes web, las posibles interacciones entre los componentes y el contenedor web, y los servicios que tiene que ofrecer el contenedor al componente.
(*) WAR Un fichero WAR (*web archive*) sigue una estructura estándar y contiene los componentes, el contenido estático y los ficheros de configuración de la aplicación web. Este fichero es el que se despliega en el contenedor web.

Este factor, y el hecho de que la manera de empaquetar las aplicaciones web esté estandarizada, hace que los componentes web y las aplicaciones web que los contienen sean portables entre diferentes implementaciones de contenedor web y se puedan desplegar en cualquier servidor de aplicaciones sin tener que recompilar el componente ni regenerar el fichero que empaqueta la aplicación web.

La portabilidad nos permite, por ejemplo, tomar una aplicación web empaquetada en un fichero WAR desplegado en un servidor de aplicaciones Jboss y desplegarla en un contenedor web Tomcat sin hacer ninguna modificación.

La plataforma J2EE define dos tipos de componentes web:
• ***Servlets***. Clase Java que recibe peticiones HTTP y genera contenido dinámico en respuesta a estas peticiones.
• ***Java Server Pages*** (JSP). Documentos de texto que se ejecutan como *servlets*, pero que permiten una aproximación más natural a la creación de contenido estático. Las JSP tienen un aspecto muy similar a los ficheros HTML, pero se les puede poner código Java.

Una opción común para implementar la capa web de una aplicación J2EE es utilizar un *framework web* que implemente un patrón modelo-vista-controlador.

(*) Frameworks Los *frameworks* suelen denominarse **marcos de trabajo.**

Capa de negocio

La capa de negocio es, seguramente, la capa más importante dentro de una aplicación empresarial. Esta capa contendrá los componentes, las relaciones y las reglas que resuelven los procesos de negocio de la empresa.

La capa de negocio proporciona a sus clientes las funcionalidades de negocio específicas en un dominio particular. En la plataforma J2EE, la lógica de negocio se implementa con componentes de negocio

reutilizables denominados ***Enterprise Java Beans*** (EJB), que se empaquetan, se despliegan y se ejecutan en el contenedor de EJB del servidor de aplicaciones.

Los EJB son, según Sun, la tecnología más importante de la especificación de J2EE. Se trata de una tecnología de componentes de servidor que permite construir aplicaciones empresariales basadas en Java. El desarrollo de la lógica de negocio con EJB hace que las aplicaciones sean escalables, transaccionales, portables, distribuidas y seguras. Igual que con las aplicaciones web, los EJB se empaquetan en ficheros JAR con una estructura estándar y se tienen que desplegar dentro de un contenedor de EJB de la plataforma J2EE para que los clientes tengan acceso a los mismos. La especificación de EJB 2.x define tres tipos básicos de EJB:
• EJB de sesión
• EJB de entidad
• EJB de mensaje
Sun Microsystems recomienda utilizar los EJB como sesión para implementar la capa de negocio, aunque cada vez más se aceptan otras alternativas como los ***Plain Old Java Objects*** (POJO son clases Java normales).

La gran ventaja de los POJO es que son objetos muy ligeros que no añaden ningún tipo de carga adicional (gestión de transacciones, seguridad, ciclo de vida, etc.). Esto los hace muy fáciles de desarrollar, además de proporcionar muy buen rendimiento. Su principal inconveniente es que no tenemos ninguna de las características que tienen los EJB y si las necesitamos, las tendremos que implementar (o usar algún *framework* que las proporcione).

Capa de integración

La capa de integración se encarga de acceder y tratar los datos que pueden estar almacenados en fuentes muy heterogéneas (bases de datos relacionales, sistemas *legacy*, bases de datos orientadas a objetos, transaccionales, ERP, etc.).

La capa de integración obtiene los datos de las diferentes fuentes de datos y los pasa a la capa de negocio para que ésta haga las transformaciones adecuadas en cada caso.
La división de la capa intermedia en capa de negocio y capa de integración suele ser puramente lógica, es decir, los componentes de las dos capas están físicamente en la misma máquina y, posiblemente, desplegados en el mismo contenedor.
En la plataforma J2EE hay diferentes aproximaciones posibles para representar las entidades (datos) en lo que se refiere al acceso a bases de datos relacionales:
• **POJO**. Utilizar clases Java normales que usen alguna técnica para acceder a los datos (JDBC directo o *frameworks* de persistencia).
• **EJB de entidad**. Componentes EJB que nos permiten tratar como un objeto los datos que tengamos en una base de datos relacional. Podemos escoger entre acceder a los datos con persistencia gestionada por el *bean* (BMP) o bien hacerlo con persistencia gestionada por el contenedor (CMP).

***(*) Frameworks* de persistencia** Los *frameworks* de persistencia son sistemas que automatizan las operaciones CRUD (creación, lectura, actualización y borrado) de entidades en las bases de datos relacionales. Estos *frameworks* pueden llegar a ser muy completos y convertirse en verdaderas herramientas de mapeo objeto/relacional.

Capa EIS

La capa de recursos (o capa EIS) contiene todos los sistemas a los que se accederá desde la capa de integración.

Estos sistemas son totalmente heterogéneos y casi siempre estarán separados físicamente.
Pueden ser servidores de bases de datos relacionales, bases de datos orientadas a objetos, ERP, transaccionales, sistemas SAP, sistemas de ficheros, etc.

Hemos visto que las aplicaciones J2EE se enmarcan en un estilo arquitectónico multicapa con *n*-niveles. También hemos realizado una breve descripción de las capas con las que podemos dividir la aplicación y de los componentes que podemos encontrar en cada una de éstas.

En el siguiente apartado, analizaremos el ciclo de vida de una aplicación J2EE, fijándonos especialmente en los procesos de ensamblaje (empaquetado) y despliegue de la aplicación y en los roles que intervienen en el ciclo de vida de un desarrollo.

Ciclo de vida

Una vez desarrollados los componentes que formarán la aplicación J2EE, se tienen que empaquetar y desplegar en el servidor de aplicaciones para que los clientes los puedan utilizar.
Para desplegarlos, primero hay que crear un módulo empaquetando los componentes desarrollados con los ficheros relacionados y un descriptor de despliegue del componente.
• Un **descriptor de despliegue** es un fichero XML asociado a un componente, un módulo o una aplicación J2EE que proporciona la información necesaria para hacer el despliegue en el servidor de aplicaciones. Indica en el contenedor declarativamente qué comportamiento tendrán los componentes que forman el módulo con respecto a seguridad, transacciones, persistencia, etc. El hecho de tener una configuración declarativa y no programática facilita la portabilidad de los componentes y módulos.
• Un **módulo** es la unidad mínima que se puede desplegar en un servidor de aplicaciones. Aunque los módulos se pueden desplegar sin que formen parte de una aplicación, lo habitual es juntar estos módulos para crear una aplicación J2EE.

Para desplegar una aplicación J2EE es necesario, por lo tanto, empaquetar todos los módulos que forman la aplicación y crear un descriptor de despliegue para la aplicación que liste todos los módulos que forman parte de la misma.

Una aplicación J2EE está formada por uno o más módulos y un descriptor de despliegue empaquetados con una estructura determinada. Los módulos son las unidades básicas de construcción de aplicaciones J2EE.

La siguiente figura esquematiza cómo se pueden estructurar los componentes, módulos y aplicaciones J2EE para desplegarlos en el servidor de aplicaciones.

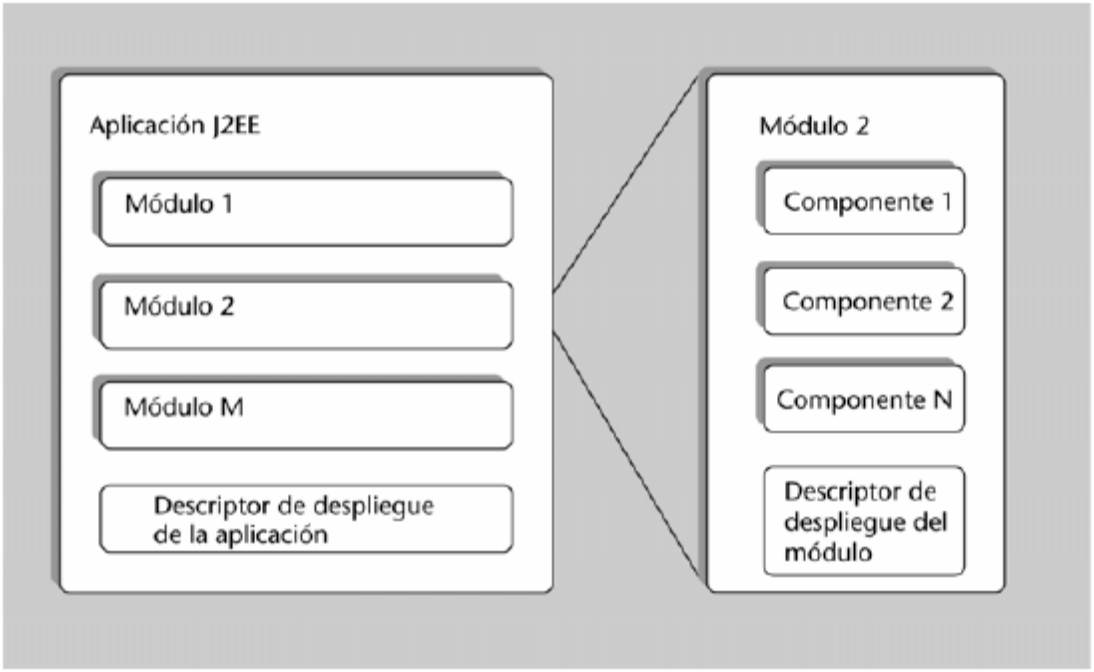


Figura 2-5. Componentes, módulos y aplicaciones J2EE.

El ciclo de vida típico para una aplicación J2EE empieza con la creación de los componentes que la formarán y el empaquetado de éstos para formar los módulos. La aplicación está formada por varios módulos que se juntan en el proceso de empaquetado y se instalan en el servidor de aplicaciones en el proceso de desplegado.
La siguiente figura muestra el ciclo de vida completo de una aplicación J2EE:

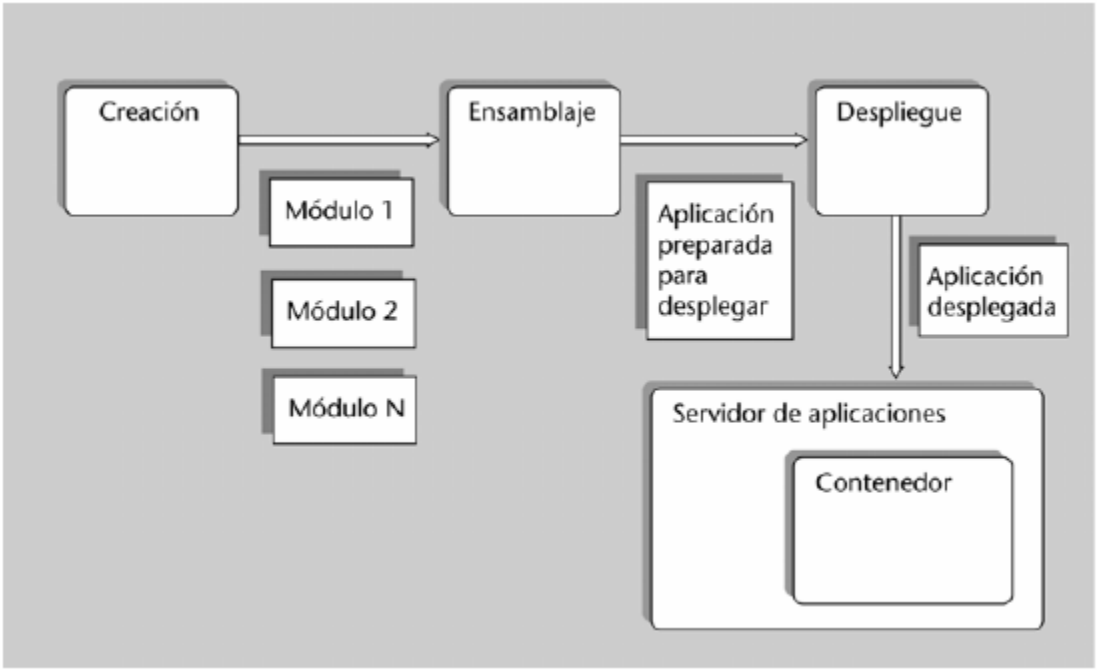


Figura 2-6. Ciclo de vida completo de una aplicación J2EE

A continuación, veremos con un poco más de detalle cada una de estas fases:

Creación

En la fase de creación se toman los componentes desarrollados, se crea el fichero de desplegado para cada componente y se empaquetan con una estructura determinada para producir el módulo.

Todas las especificaciones de componentes de la plataforma J2EE incluyen en su definición la estructura que tiene que seguir para empaquetarlos y desplegarlos en los contenedores, así como el formato (esquema XML) que debe tener el fichero de despliegue del módulo.

La salida del proceso de creación es un módulo, la unidad básica a partir de la cual se forman las aplicaciones J2EE.

J2EE define cuatro tipos de módulos:
• Módulo cliente. Fichero con extensión JAR que contiene las clases de la aplicación cliente, el descriptor de despliegue y los ficheros relacionados.
• Módulo web. Fichero con extensión WAR que contiene los componentes web (*servlets* y JSP), el descriptor de despliegue y los recursos relacionados (páginas HTML, JavaScript, imágenes, etc.)
• Módulo EJB. Fichero con extensión JAR que contiene todos los ficheros del componente EJB, su descriptor de despliegue y los ficheros relacionados.
• Módulo de adaptadores de recursos. Fichero con extensión RAR que contiene los adaptadores de recursos, su descriptor de despliegue y los ficheros relacionados. Estos componentes no los veremos en este curso.

Ensamblaje

La fase de ensamblaje crea el fichero con la aplicación que se quiere desplegar en el servidor de aplicaciones.

Los módulos creados en la fase de creación se pueden desplegar directamente en el servidor de aplicaciones o bien combinarlos para crear una aplicación J2EE.

Una aplicación J2EE está formada por un conjunto de módulos y un descriptor de despliegue de la aplicación. Se empaqueta en un fichero con extensión EAR para formar una unidad de despliegue.

En la fase de ensamblaje de la aplicación tenemos que generar el descriptor de despliegue de la aplicación y crear el fichero EAR con los módulos que forman la aplicación.

Despliegue

La fase de despliegue instala y configura la aplicación J2EE en el servidor de aplicaciones.

Para hacer la instalación de la aplicación, el servidor de aplicaciones lee el descriptor de despliegue e instala cada componente listado en el contenedor que le toca (los módulos EJB en el contenedor de los EJB, los módulos web en el contenedor web, etc.), asignando a los componentes de cada módulo las propiedades que tienen configuradas en los respectivos descriptores de despliegue.
Los servidores de aplicaciones suelen proporcionar herramientas para hacer el despliegue de aplicaciones.

2.2 La capa intermedia (*middle layer*)

Hasta ahora hemos visto que J2EE sigue un estilo arquitectónico cliente-servidor multicapa con *n*-niveles y que las aplicaciones se pueden dividir, básicamente, en tres capas:
• Capa cliente
• Capa intermedia
• Capa EIS

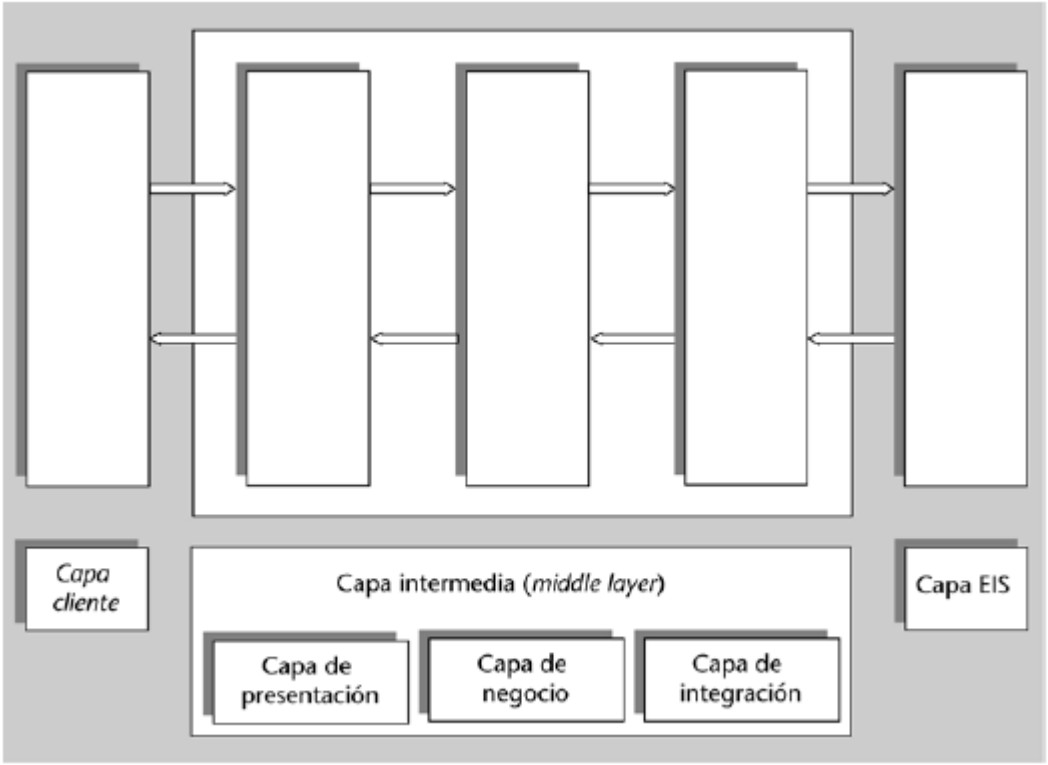


Figura 2-7. La capa intermedia.

La capa intermedia contendrá los componentes que se ejecutarán en los contenedores del servidor de aplicaciones. Esta capa se puede dividir lógicamente en tres capas más:
• Capa de presentación
• Capa de negocio
• Capa de integración
En este apartado, trataremos con profundidad las tres capas con las que podemos dividir la capa intermedia.
En primer lugar, analizaremos la capa de presentación para aplicaciones web y veremos qué componentes define J2EE para implementar esta capa. Después analizaremos las capas de lógica de negocio y de integración con las propuestas de J2EE para desarrollar estas dos capas. Por último, veremos un par de propuestas de arquitectura para la capa intermedia de las aplicaciones J2EE.

2.2.1 Capa de presentación

En este apartado nos centraremos en la tecnología y los componentes que define J2EE para implementar la capa de presentación para entornos web (a partir de ahora, la denominaremos *capa web*).
Veremos qué objetivos tiene la capa web de una aplicación y qué componentes permiten alcanzar estos objetivos; analizaremos cómo se empaqueta y se despliega una aplicación web, cuáles son las arquitecturas más típicas en este tipo de aplicaciones y, por último, daremos algunas recomendaciones a la hora de hacer el diseño de esta capa.

La capa web

La capa web de una aplicación J2EE tiene como objetivo primordial hacer accesible la lógica de negocio a clientes que accedan a la aplicación mediante el protocolo HTTP (por ejemplo, con un navegador).

HTTP y el paradigma petición-respuesta

La capa web de una aplicación J2EE está desarrollada sobre el protocolo http siguiendo el paradigma de petición-respuesta. Los componentes web reciben peticiones HTTP del servidor web, las procesan y generan una respuesta según esta petición.

Existe un servidor en la capa web que recibe las peticiones HTTP de los clientes. Si éstas van dirigidas a un componente web, las pasa al contenedor web, que se encargará de procesarlas y retornar la respuesta al cliente. Si la petición no va dirigida a ningún componente web (por ejemplo, una petición de una imagen, de una página HTML estática o de una película de Flash), el servidor web envía la respuesta al cliente sin pasar por el contenedor web.

Objetivos de la capa web

La capa web de una aplicación J2EE no es sólo la capa encargada de dar formato al contenido dinámico que se quiere enviar a los clientes; hace muchas otras tareas, entre las que hay que destacar las siguientes:
• Permitir a los clientes HTTP que llamen a la lógica de negocio de la aplicación recogiendo las peticiones de éstos, llamando al método de negocio correspondiente y presentando los resultados a los clientes.
• Generar contenido dinámico.
• Gestionar el flujo de presentación, es decir, dada una entrada y un resultado en la ejecución de la lógica de negocio, decidir cuál es la siguiente pantalla que hay que mostrar al usuario.
• Hacer ciertas validaciones a los datos que recibe como entrada.
• Mantener el estado entre diferentes peticiones sobre el protocolo http (que no tiene estado).
• Dar soporte a distintos tipos de clientes. Por ejemplo, la capa web de una aplicación J2EE podría servir contenido a clientes ligeros con HTML y a teléfonos móviles con WML.
• Puede implementar la lógica de negocio (aunque quizá no es lo más habitual).

Evolución

Inicialmente, se utilizaba Internet para proporcionar contenido estático: había un conjunto de servidores HTTP que recibían las peticiones de los clientes y les devolvían la información con HTML estático.
Pronto se vio el gran potencial de Internet para proporcionar a los usuarios no sólo contenido estático, sino también contenido dinámico y personalizado. El siguiente objetivo fue conseguir generar contenido dinámico según una petición HTTP. Para conseguirlo, se desarrollaron módulos que extendían los servidores HTTP y podían generar contenido dinámico denominado *Common Gateway Interface* (CGI).
Los CGI, aunque se convirtieron en un mecanismo muy utilizado para generar contenido dinámico, tienen algunos problemas importantes como, por ejemplo, la eficiencia.
La plataforma J2EE y los componentes web surgen como alternativa a los CGI. Se basan en el uso de Java como lenguaje para desarrollar las “extensiones”, y en utilizar un contenedor para proporcionar a estas extensiones el soporte a los servicios de sistema.
Los componentes web solucionan los problemas de rendimiento de los CGI, son portables y pueden acceder a muchos servicios proporcionados por el contenedor en el que se despliegan.
La especificación de J2EE define dos tipos de componentes web: los *servlets* y las *Java Server Pages* (JSP), que analizaremos en los siguientes apartados.

Servlets

Un *servlet* es un objeto Java que extiende la funcionalidad de un servidor web, recibe peticiones HTTP y genera contenido dinámico como respuesta a estas peticiones.
Los *servlets* “viven” dentro de un contenedor weby se cargan y ejecutan dinámicamente como respuesta a las peticiones que los clientes hacen a una URL determinada. Basan su funcionamiento en el paradigma de petición-respuesta bajo el protocolo HTTP.

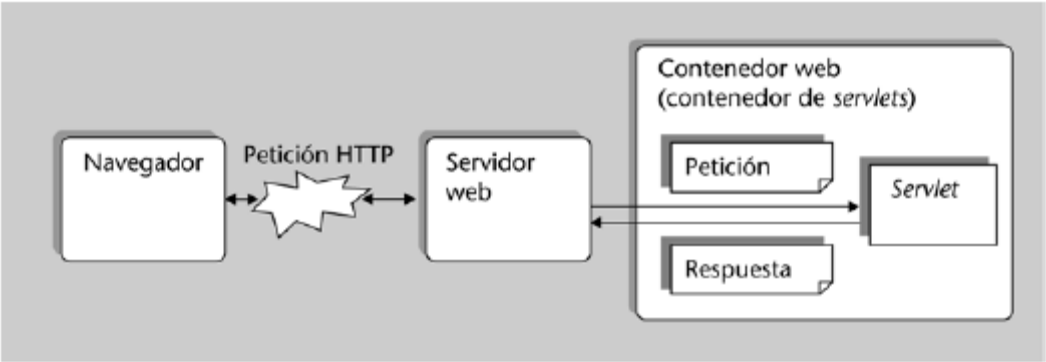


Figura 2-8. Paradigma de petición respuesta.

El servidor web y el contenedor de *servlets* pueden estar en la misma máquina o bien en máquinas diferentes. Los pasos para gestionar una petición HTTP son los siguientes:
1) El cliente hace una petición HTTP a una URL a la que hay asociado un *servlet*.
2) El servidor web recibe esta petición, se da cuenta de que va dirigida a un *servlet* y pasa la petición al contenedor de *servlets*.
3) El contenedor de *servlets* crea un objeto Java que representa la petición y lo pasa al *servlet* llamando al método service() de éste.
4) El *servlet* procesa la petición:
- – Analiza los parámetros de la petición.
- – Ejecuta las tareas que requiere para completar la petición.
- – Crea una respuesta (muchas veces la respuesta es una página HTML).

5) El *servlet* retorna la respuesta al contenedor de *servlets*.
6) El contenedor de *servlets* envía la respuesta al servidor web.
7) El servidor web la envía al cliente.

El ciclo de vida de los *servlets* lo controla el contenedor (la gestión del ciclo de vida de los componentes es un servicio que proporcionaban los contenedores a los componentes que tienen desplegados) donde se ha desplegado. Cuando el contenedor recibe una petición que está mapeada con un *servlet*, se producen los pasos siguientes:

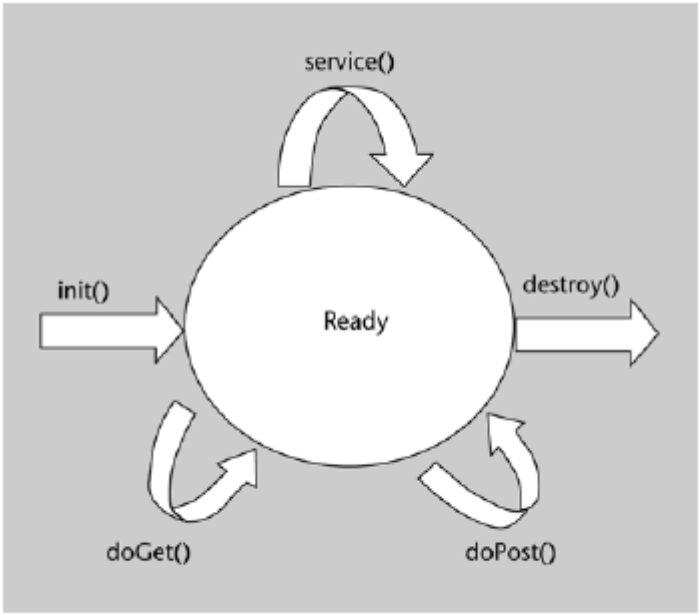


Figura 2-9.Ciclo de vida de un servlet.

1) El contenedor crea los objetos que modelan la petición y la respuesta.

2) Si no hay ninguna instancia del componente en memoria, el contenedor instancia una y la inicializa llamando al método init() y después al método service().
3) Si ya había una instancia del componente, llama al método service() directamente.
4) El desarrollador programa el comportamiento del *servlet* para cada petición en su método service().
Ejemplo de *servlet* que muestra una página HTML que dice "¡Hola!":

```
public class ServletSimple extends HttpServlet {
        public void service
        (HttpServletRequest   req,   HttpServletResponse   nada)
        throws IOException, ServletException
        {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML><HEAD>");
        out.println("<TITLE>Hola</TITLE>");
        out.println(</HEAD>);
        out.println("<BODY>¡Hola!</BODY></HTML>");
        out.close();
        }
}
```

Las peticiones HTTP más habituales son peticiones POST o bien peticiones GET. Los *servlets* tienen los métodos doGet() y doPost() para servir directamente estas peticiones. Si el programador ha sobreescrito el método doPost() y llega una petición POST, se ejecuta el código que haya en este método.

En este ejemplo es el *servlet* que genera el contenido dinámico con código Java, que escribe directamente el código HTML que se enviará al cliente.
Esta manera de generar el aspecto gráfico de las aplicaciones tiene distintos problemas graves:
• Hace muy complejo crear la presentación, ya que no se pueden utilizar herramientas de diseño gráfico para dar aspecto a las páginas HTML que recibe el cliente. No podemos utilizar editores WYSIWYG (What You See Is What You Get 'lo que ves es lo que obtienes') para crear la capa de presentación y se hace muy difícil realizar el diseño gráfico sofisticado que requieren la mayoría de las aplicaciones.
• El coste de mantenimiento es muy alto, más todavía si tenemos en cuenta que el aspecto de las aplicaciones será el componente que más cambia a lo largo del tiempo.
• No promueve una separación clara de roles en el desarrollo. Los mismos desarrolladores que programan con Java la lógica de presentación tienen que crear el HTML que dé el aspecto gráfico. Para los diseñadores gráficos, es difícil generar el aspecto de la aplicación programando con Java.

Para solucionar los problemas anteriores aparecen las JSP, que nos permitirán hacer la capa de presentación de las aplicaciones web con una sintaxis más próxima al HTML que al Java.

Java Server Pages

Una JSP es un documento similar a una página HTML pero que, además de generar contenido estático, puede incluir elementos para generar contenido dinámico como respuesta a una petición HTTP.

Los elementos para generar contenido dinámico son pedazos de código escritos en Java y unas etiquetas del mismo tipo que las etiquetas HTML que interactúan con objetos Java del servidor.
La tecnología JSP tiene todas las capacidades dinámicas de los *servlets*, pero proporciona una manera más natural de crear el contenido estático.
Por ejemplo, el siguiente trozo de código es una JSP que muestra una página HTML que dice "¡Hola!". Es el mismo ejemplo que hemos visto con *servlets*, pero escrito con una JSP:

```
<Html>
<Head>
<Title>Hola</title>
</Head>
<Body>
¡Hola!
</Body>
</Html>
```

En el ejemplo anterior sólo se generaba código estático. En cambio, las JSP también son capaces de generar código dinámico. Así, por ejemplo, la siguiente JSP también mostrará la fecha y la hora de cada petición recibida:

```
<Html>
<Head>
<Title>Hola</title>
</Head>
<Body>
¡Hola!
<Br>
La fecha actual es: <%=
java.util.Date() %>
</Body>
</Html>
```

La tecnología de *servlets* y la de JSP están muy relacionadas. De hecho, las JSP son transformadas internamente en *servlets* cuando se despliegan en el contenedor web. Por este motivo, su ciclo de vida es muy similar al que hemos estudiado para el caso de las *servlets*; la diferencia fundamental radica en la sintaxis.
El contenedor web convierte las JSP en *servlets* antes de servir las peticiones de los clientes.
Cuando un servidor web recibe una petición de una página JSP pasa la petición al contenedor web, y lo primero que hace éste es comprobar si ya ha generado el *servlet* asociado a la JSP y si está actualizado (la JSP no ha cambiado desde la última generación del *servlet* asociado).
Si no existe el *servlet* asociado o no está actualizado, el contenedor procesa la página para generar el código fuente del *servlet* asociado y compilarlo. El *servlet* generado se carga en el contenedor y sigue el ciclo de vida de un *servlet*, tal y como hemos visto en el apartado anterior.

El siguiente esquema muestra el flujo que sigue una petición HTTP:

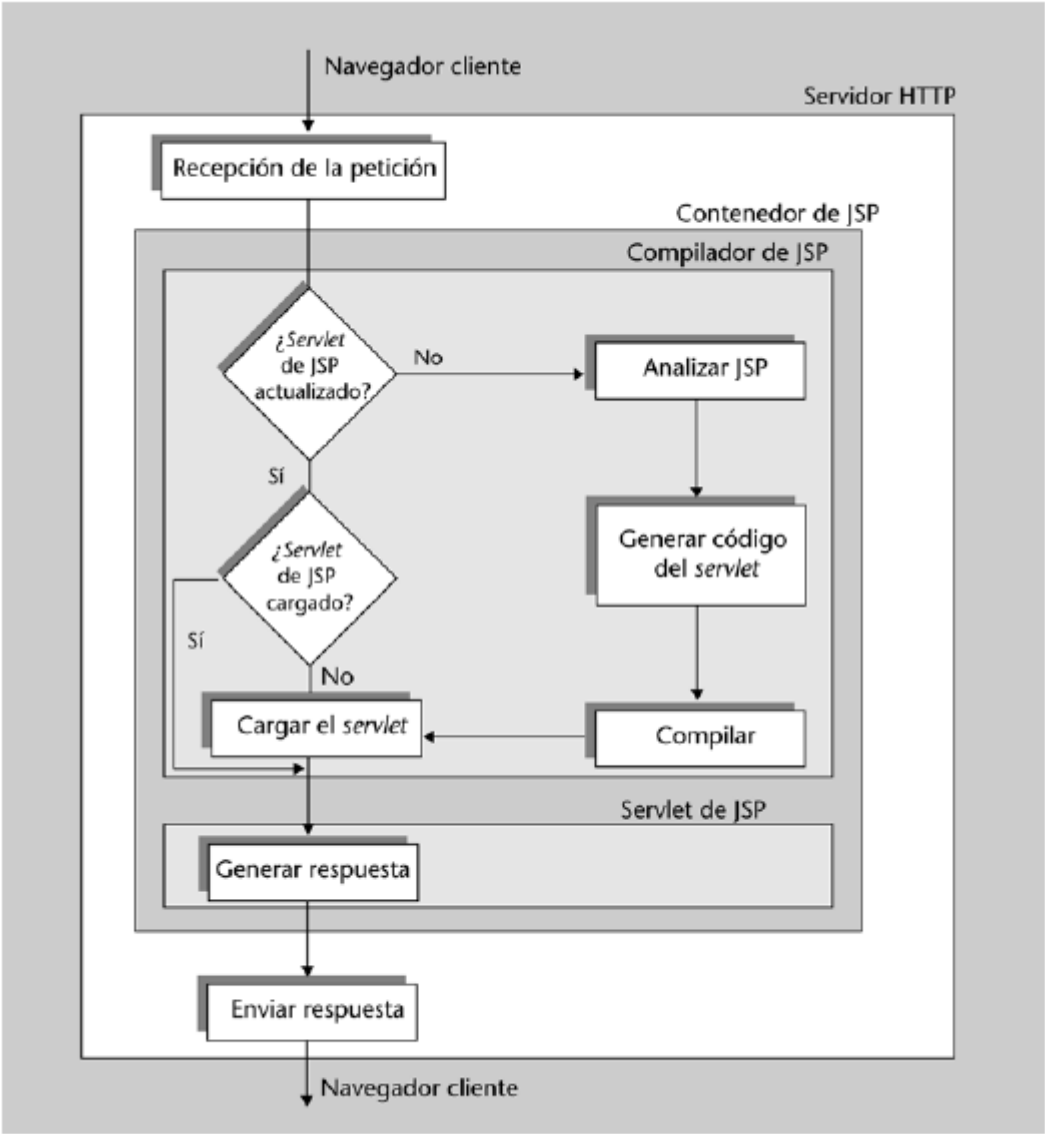


Figura 2-10 Ciclo de vida una JSP

Las aplicaciones web están formadas por componentes web (*servlets* y JSP) y contenido estático que se empaqueta en un formato estándar para poder ser desplegado en el contenedor web. Los ficheros de empaquetado de las aplicaciones web tienen extensión WAR y obedecen a las directrices que marca la especificación.

Los componentes web se parametrizan en un fichero denominado ***web.xml*** que describe su comportamiento. Por ejemplo, en el fichero ***web.xml*** se pueden especificar parámetros de inicialización, el URL en el que atenderá peticiones el componente (mapeo del *servlet*), etc.

La combinación de las tecnologías de *servlets* y JSP promueve una separación clara de roles: los diseñadores gráficos se dedicarán a crear el aspecto gráfico de la aplicación con las herramientas y los lenguajes que mejor saben utilizar, y los programadores Java crearán la lógica de presentación que se encargue de invocar a la lógica de negocio y seleccionar la vista adecuada para cada caso con *servlets* o clases Java auxiliares.

Recomendaciones de diseño para la capa de presentación

A la hora de realizar una implementación con una tecnología de componentes determinada (en este caso, J2EE), es necesario hacer una fase previa de diseño con el objetivo de evitar errores y reducir la distancia existente entre la especificación y la implementación concreta.
En esta fase de diseño se toman decisiones de arquitectura que dependen de cada tecnología concreta. En este apartado, haremos una revisión breve de las arquitecturas más habituales que podemos encontrar en la capa web de una aplicación J2EE. También daremos algunas recomendaciones de diseño para hacer el paso de la especificación a la implementación de la capa de presentación para las aplicaciones web.
La capa de presentación es una de las capas que cambia más frecuentemente. Un mal diseño puede hacer que la aplicación que desarrolláis se convierta en muy compleja y poco mantenible.

¿Necesitamos una aplicación web?

Para implementar la capa de presentación como una aplicación web, lo primero que debemos hacer siempre es evaluar si realmente una aplicación web es la mejor alternativa para implementar la capa de presentación.

J2EE no cierra la puerta al desarrollo de la capa de presentación con tecnologías no basadas en clientes ligeros.
Una alternativa a considerar es hacer la presentación con una aplicación de escritorio.
Evaluad cuestiones como:
• ¿Necesitáis mucha riqueza en la interfaz gráfica, con muchos controles de usuario complejos e interrelacionados?
• ¿Tiene que dar la aplicación una respuesta muy rápida a las interacciones del usuario?
• ¿Es accesible sólo por usuarios internos?
• ¿Necesitáis capacidades de procesamiento *off-line*?
• ¿Tiene que interactuar vuestra aplicación con otras aplicaciones locales o con el sistema operativo de las máquinas cliente?

Si la respuesta a las cuestiones anteriores es afirmativa, posiblemente el diseño de la capa de presentación como una aplicación web y el cliente como un cliente ligero no es la mejor alternativa. En este caso, podemos considerar hacer la presentación con una aplicación de escritorio (ya sea con Java o bien con alguna otra tecnología como .NET) o, últimamente, el uso de AJAS (*Asynchronous JavaScript and XML*).

AJAX es una unión de varias tecnologías, basadas en JavaScript, llamadas asíncronas y XML, que permite refrescar el contenido de partes de una página HTML sin tener que volver a enviarla toda.

Capa de presentación simple y delgada

El principio de diseño más importante que hay que tener siempre en la cabeza cuando se diseña la capa web de una aplicación J2EE es separar el control del flujo y la llamada a la capa de negocio de la presentación. Normalmente, utilizaremos *servlets* para controlar el flujo e invocar la lógica de negocio y JSP para hacer la presentación.
La capa de presentación debe ser lo más delgada posible. Idealmente, sólo tendría que contener código para traducir las peticiones de usuario a llamadas a métodos de negocio, nunca la implementación de los métodos de negocio, y transformar los resultados de estas llamadas al formato que espera la presentación.

Actualmente se está hablando mucho de *Smart Client.* Las aplicaciones de tipo *Smart Client*
intentan crear clientes que conjuguen el mejor de los clientes ligeros y las aplicaciones de escritorio.

Capítulo 3.

3. Evolución de las arquitecturas web: MODELOS JSP y Arquitectura MVC

La arquitectura de las aplicaciones web ha evolucionado bastante desde su inicio.

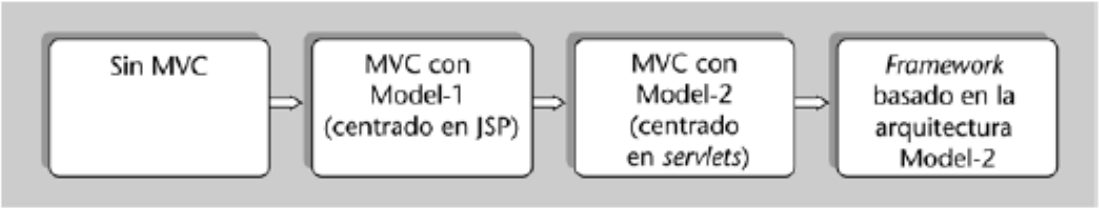


Figura 3-1Evolución de las arquitecturas web.

Inicialmente, la mayoría de las aplicaciones web mostraban contenido estático y las que necesitaban generar contenido dinámico lo hacían directamente con CGI o bien con *servlets* o JSP. Esta manera de proceder hacía aplicaciones muy difíciles de mantener y no proporcionaba ningún tipo de reutilización.
La utilización de JSP da una seria de problemas:

- Mezcla de código de lógica y presentación difícil de mantener
- Demasiado código en la JSP
- Lógica de flujo incrustada en la página
- Difícil de debugear

3.1 Modelo JSP 1 Capa

Para solventarlo los problemas de programación con JSP, se pasó a utilizar algunas características de las JSP, que permitían hacer una aplicación más estructurada y un poco más reutilizable. Esta arquitectura se denomina *Model-1*.

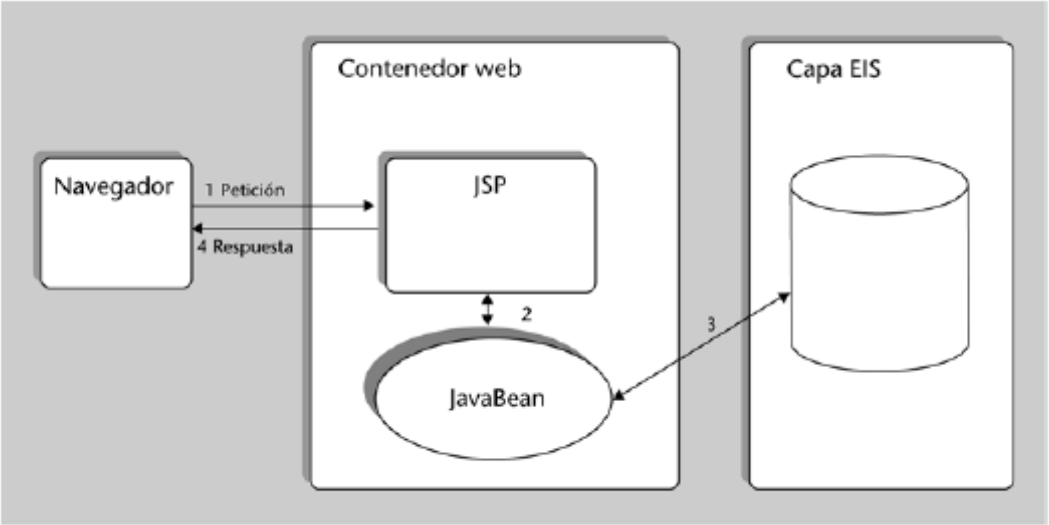


Figura 3-2 Arquitectura Model-1

En una arquitectura Model-1, los clientes acceden directamente a páginas JSP las páginas JSP procesan la petición y devuelven la respuesta al cliente. Las páginas JPS utilizan *JavaBeans* para ejecutar la lógica de

negocio (modelo de la aplicación) y determinan la siguiente página que hay que mostrar (es decir JavaBeans realizan una separación del contenido y presentación).
La arquitectura Model-1 tiene algunas desventajas importantes:
• La selección de la siguiente vista está descentralizada.
• Cada página JSP es responsable de procesar los parámetros que le llegan.
• Es difícil mostrar diferentes páginas, dependiendo de los parámetros de entrada de la petición.
• Demasiado código Java en la JSP (Scriptlet)
• Poca claridad, difícil de entender
• Mezcla de código de lógica y presentación, difícil de mantener
• Navegación (lógica de flujo) en la página
• No modular
• Difícil de debugear

3.2 Modelo JSP 2 Capas

En una arquitectura Model-2, los *servlets* y las JSP colaboran para implementar la capa de presentación de las aplicaciones web. Hay un *servlet* "controlador" entre las peticiones de los clientes y las páginas JSP que muestran la salida.

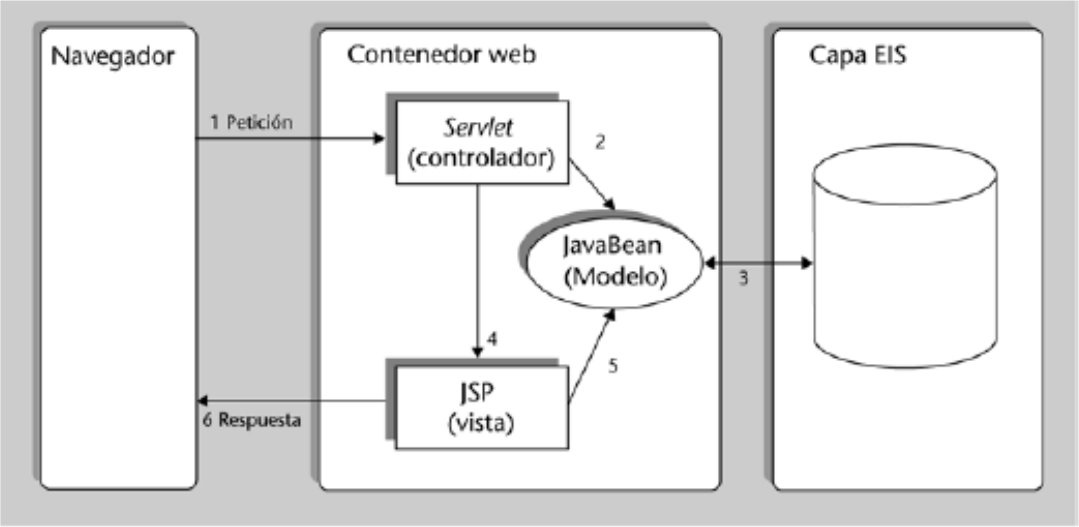


Figura 3-3. Arquitectura Model-2

Este controlador se encarga de centralizar la lógica de selección de la siguiente vista según los parámetros de entrada, el resultado de la invocación a la lógica de negocio y/o el estado de la aplicación. El controlador también se encarga de ejecutar la llamada a la lógica de negocio adecuada a partir de la petición que le llegue.
Las aplicaciones basadas en Model-2 permiten una separación clara de roles en la capa de presentación, son más flexibles, mantenibles y permiten tratamientos centralizados para todas las peticiones (por ejemplo, aplicar seguridad y *logs*). Por todo esto, es recomendable seguir esta arquitectura para la capa de presentación.

3.3 Arquitectura MVC

En la arquitectura MVC existe un proceso de abstracción que permite dividir la aplicación entre los componentes involucrados en mantener los datos y los que presentan los datos. De ésta forma minimizamos la dependencia entre ellos y permitimos que diferentes desarrolladores o equipos de desarrolladores puedan cumplir perfiles diferenciados. Lo más importante de ésta arquitectura es que permite que futuros cambios técnicos o de aparición de estándares tengan el mínimo impacto sobre lo ya desarrollado. De ésta forma se

consigue construir aplicaciones compuestas por componentes débilmente acoplados. Esta circunstancia añade una enorme cantidad de ventajas.

La arquitectura MVC divide los objetos o componentes implicados en una aplicación en 3 tipos: modelo, vista y controlador.

- *Modelo*
  Representa los datos de la aplicación y las reglas de negocio (que en algunos casos gobiernan el acceso y la modificación a estos datos).
  El modelo notifica a las vistas de sus cambios y proporciona los métodos necesarios para que las vistas puedan consultar su modelo asociado y conocer la información que contienen.
- *Vista*
  Muestra el contenido del modelo. Accede a los datos del modelo y especifica de qué manera se muestran estos datos.
  La Vista también se encarga de redireccionar todos los eventos/acciones del usuario hacia el Controlador.
- *Controlador*
  Define el comportamiento de la aplicación, interpreta los eventos del usuario y mapea las acciones a realizar sobre el modelo. Las acciones pueden ser realizar un cambio al estado del modelo o ejecutar funciones de la lógica de negocio del modelo.
  Dependiendo de la acción realizada por el usuario, el Controlador puede seleccionar una nueva vista a mostrar como respuesta a la petición realizada por el usuario.

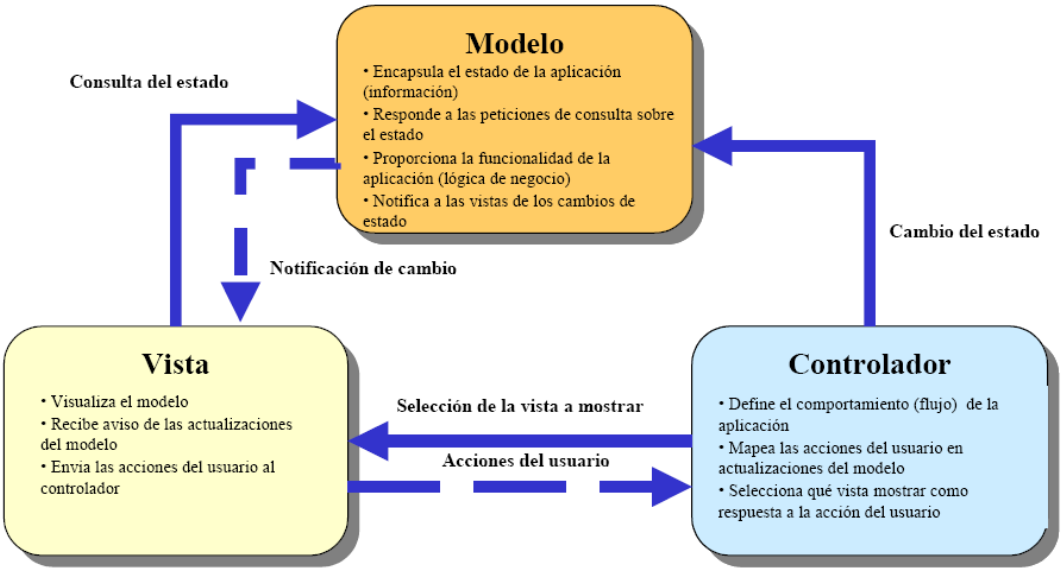


Figura 3-4. Modelo Vista Controlador: Esquema de Participación

Con el uso de una arquitectura MVC se obtiene:

- Diseño claro por separación de roles de las clases
- Se permiten múltiples vistas a los usuarios (diferentes representaciones sin variar la lógica de negocio)
- Simplifica la extensión del sistema, el impacto en el análisis y el despliegue
- Facilidad de mantenimiento debido a que no se duplica el código

Capítulo 4.

4. Frameworks

Si seguimos una arquitectura basada en Modelo 2 o MVC, hay un conjunto de tareas repetitivas que se tienen que implementar para todas las aplicaciones, por ejemplo, recibir los parámetros de entrada de las peticiones y hacer validaciones, llamar a la lógica de negocio, escoger la siguiente vista para mostrar, etc. Estas tareas se pueden implementar en un *framework* o marco de trabajo para la capa web que utilicen todos los desarrollos, de manera que no se deban implementar para cada aplicación.

Los desarrolladores crearán la capa web usando o extendiendo las clases e interfaces del *framework*. El uso de un *framework* para la capa web basado en una arquitectura Modelo 2 o MVC proporciona las ventajas siguientes:

• Desacopla la capa de presentación de la capa de negocio en componentes separados.
• Simplifica y estandariza la validación de los parámetros de entrada.
• Simplifica la gestión del flujo de navegación de la aplicación.
• Proporciona un punto central de control.
• Permite un nivel muy alto de reutilización.
• Impone la misma arquitectura para todos los desarrollos.
• Simplifica muchas tareas repetitivas.

4.1 Definición

Una primera definición de Framework, extraída de Erich Gamma es:

"A framework is a set of classes that embodies an abstract design for solutions to a family related problems"(JohnsonandFoote1988)

Conjunto de clases cooperativas que construyen un diseño reutilizable para un tipo específico de software. Un Framework proporciona la arquitectura partiendo el diseño en clases abstractas y definiendo sus responsabilidades y colaboraciones. Un desarrollador realiza una aplicación haciendo subclases y componiendo instancias a partir de las clases definidas por el Framework.

Un framework es una aplicación semi-completa, de control invertido, así un Framework difiere de una librería de clases:
-En una librería de clases, el control del flujo se encuentra en el código de la aplicación que realiza llamadas a los métodos de la librería de clases
-En un framework, el control del flujo está en código del framework que realiza llamadas al código de la aplicación (control invertido)

4.2 Características

Existen varias diferencias entre una librería de clases y un Framework que se exponen en la siguiente tabla (Extraída del PFC: Framework Gráfico Herramientas Case con instanciación a UML (1997): Xavier Escudero Sabadell y Xavier Arias Losada)

Característica	Frameworks	Librerías de Clases
Modelo de la Aplicación	Los frameworks son la aplicación, de forma que manipulan el flujo de control	El usuario ha de crear el control de la aplicación y la lógica para invocar los componentes
Estructura de la Aplicación	Multiples frameworks cooperantes	Una sola aplicación monolítica consistente en un conjunto de librerías de clases
Obtención de Servicios	Los frameworks son el Servicio	Heredando funciones de las librerías de clases

Creación del Sistema	Los frameworks invocan el código del usuario. Este código puede derivar partes del Framework	Derivando o creando nuevas clases
Granularidad del Control	Media. El usuario solo puede derivar algunas partes del Framework	Alta. El usuario puede derivar cualquier clase de la librería
Abstracción de los Servicios	Alta. Ocultan su complejidad. Automatizan las características estándar y ofrecen un mecanismo de excepciones	El usuario ha de determinar qué métodos ofrece la librería y en qué orden deben ser invocados
Cantidad de nuevo código a implementar	Muy poco	Medio
Coste de mantenimiento	Bajo	Medio
Reducción Complejidad	Mucha. El usuario escribe pequeñas porciones de código	Media. El usuario tiene que crear nuevas clases para cada una de las aplicaciones que desarrolle y desarrollar el programa
Tiempo necesario para desarrollar una aplicación	Bajo	Medio. Depende del grado de reutilización que se pueda hacer de la librería
Reutilización de Código	Muy alto	Alto

4.3 Ventajas de un framework

Como ventajas en la utilización de un framework tenemos las siguientes:

a) Minimiza tiempos de desarrollo / acorta el “Time to market”
. Los proyectos de desarrollo ya no tendrán que resolver los múltiples problemas asociados a las aplicaciones web
. Los frameworks reducen la codificación y sobretodo la puesta en marcha, ya que proporcionan subsistemas que sabemos que ya funcionan. En definitiva, proporcionan código que no se tendrá que mantener ni reescribir.

b) Reduce los riesgos del desarrollo
. Con un modelo de programación complejo como el de J2EE, el riesgo de fallos en los proyectos iniciales es alto. Un framework de desarrollo de aplicaciones reduce significativamente este riesgo al proporcionar una base fiable y suficientemente probada

c) Proporciona una arquitectura consistente entre aplicaciones
. Al usar frameworks todas las aplicaciones generadas comparten una arquitectura común. Esto hace que sea más fácil de aprender, mantener y soportar
. Cualquier programador que trabaje con un framework no deberá invertir gran parte de su tiempo en buscar las clases necesarias, interconectarlas o descubrir los métodos que contienen. Los frameworks ocultan toda esta complejidad dando un alto nivel de abstracción.

En definitiva, menor coste en el diseño, desarrollo, pruebas y mantenimiento de las aplicaciones.

4.4 Desventajas de un Framework

Como desventajas en la utilización de un framework tenemos las siguientes:

a) Limitación de la Flexibilidad

Los componentes que un usuario construya a partir de un Framework deben residir dentro de las restricciones impuestas por la arquitectura del Framework

b) Dificultad de Aprendizaje

El usuario debe estudiar qué proporciona el Framework y cómo debe hacer uso de él. Este aprendizaje (requerido una sola vez) es costoso en el inicio, por partida, el desarrollo será más rápido.

c) Reducción de la Creatividad

Puede suceder que el usuario no pueda cambiar el comportamiento de una clase del Framework si ésta no permite que sus servicios puedan rescribirse o redefinirse

4.5 Opciones en el Mercado

Hay muchos *frameworks* para la capa de presentación de las aplicaciones web. Algunos de éstos son:

- Struts.
- WebWork.
- Java Server Faces.
- Spring (aunque Spring es más genérico, también tiene un *framework* de presentación).

Capítulo 5.

5. Patrones de Diseño

5. 1¿Qué es un Patrón?

Algunas personas definen un patrón como una solución recurrente para un problema en un contexto. Así:

> Un contexto es el entorno, situación, o condiciones interrelacionadas dentro de las cuales existe algo.
> Un problema es una cuestión insatisfecha, algo que se necesita investigar y resolver. Un problema se puede especificar mediante un conjunto de causas y efectos. Normalmente un problema está restringido al contexto en el que ocurre.
> La solución se refiere a la respuesta al problema dentro de un contexto que ayuda a resolver las dificultades.

Además de tener una solución a un problema en un contexto necesitamos asociar la característica de recurrencia con la definición de un patrón. Los patrones deberían comunicar soluciones de diseño a los desarrolladores y arquitectos que los leen y los utilizan.

5.2 Abstracción de Patrones

Un patrón describe, con algún nivel de abstracción, una solución experta a un problema. Normalmente, un patrón está documentado en forma de una plantilla. Aunque es una práctica estándar documentar los patrones en un formato de plantilla especializado, esto no significa que sea la única forma de hacerlo. Además, hay tantos formatos de plantillas como autores de patrones, esto permite la creatividad en la documentación de patrones.
Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que describen soluciones para todo, desde el análisis hasta el diseño y desde la arquitectura hasta la implementación. Además, los patrones existen en diversas áreas de interés y tecnologías. Por ejemplo, hay un patrón que describe como trabajar con un lenguaje de programación específico o un segmento de la industria específico, como la sanidad.

5.3 Identificar un Patrón

Se han manejado muchos proyectos J2EE en Sun Java Center, y, con el tiempo, se ha notado que ocurren problemas similares en todos estos proyectos. También se ha visto que han emergido soluciones similares para todos estos problemas. Aunque las estrategias de implementación variaran, las soluciones generales eran bastante similares.
Cuando se ha visto un problema y una solución recurrentes, se ha intentado identificar y documentar sus características usando la plantilla de patrón. Los candidatos a patrones no se han añadido al catálogo de patrones hasta que no se ha podido observar y documentar su utilización varias veces en diferentes proyectos. También se emprendió el procesamiento de significado de los patrones buscando patrones en soluciones ya implementadas.
Luego, hay una "Regla de Tres", como se la conoce en la comunidad de los patrones. Esta regla es una guía para ascender un patrón candidato al catálogo de patrones. De acuerdo a esta regla, una solución permanece como un patrón candidato hasta que se ha verificado en al menos tres sistemas diferentes. Ciertamente que hay mucho espacio para la interpretación en reglas como ésta, pero ayudan a proporcionar un contexto para la identificación de patrones.
Muchas veces, soluciones similares podrían representar un sólo patrón. Cuando se decide cómo formar un patrón, es importante considerar cual es la mejor forma de comunicar la solución. Algunas veces, un nombre independiente mejora la comunicación entre los desarrolladores. Si es así, es mejor considerar la documentación de dos soluciones similares como dos patrones diferentes. Por otro lado, podría ser mejor comunicar la solución destilando las ideas similares en una combinación de patrón/estrategia.

5. 4 Patrones contra Estrategias

Cuando se empezó a documentar los patrones J2EE, se tomó la decisión de documentarlos con un nivel de abstracción relativamente alto. Al mismo tiempo, cada patrón incluye varias estrategias que proporcionan detalles de su implementación a bajo nivel. A través de las estrategias, cada patrón documenta una solución con varios niveles de abstracción. Se ha reconocido que se podría haber documentado algunas de estas estrategias como patrones por sí mismas. Sin embargo, se ha creído que la estructura de plantilla actual comunica más claramente la relación de las estrategias con la estructura de patrón de alto nivel en que se ha incluido.
Se han anotado algunos de los problemas con respecto a la relación entre las estrategias y los patrones:

- Los patrones existen a un nivel de abstracción más alto que las estrategias.
- Los patrones incluyen implementaciones más comunes o más recomendadas que las estrategias.
- Las estrategias proporcionan un punto de extensibilidad para cada patrón. Los desarrolladores descubren e inventan nuevas formas de implementar patrones, produciendo nuevas estrategias para patrones bien conocidos.
- Las estrategias promueven una mejor comunicación, proporcionando nombres para aspectos de bajo nivel de una solución particular.

El catalogo de Patrones que presenta J2EE es (Core J2EE Patterns):

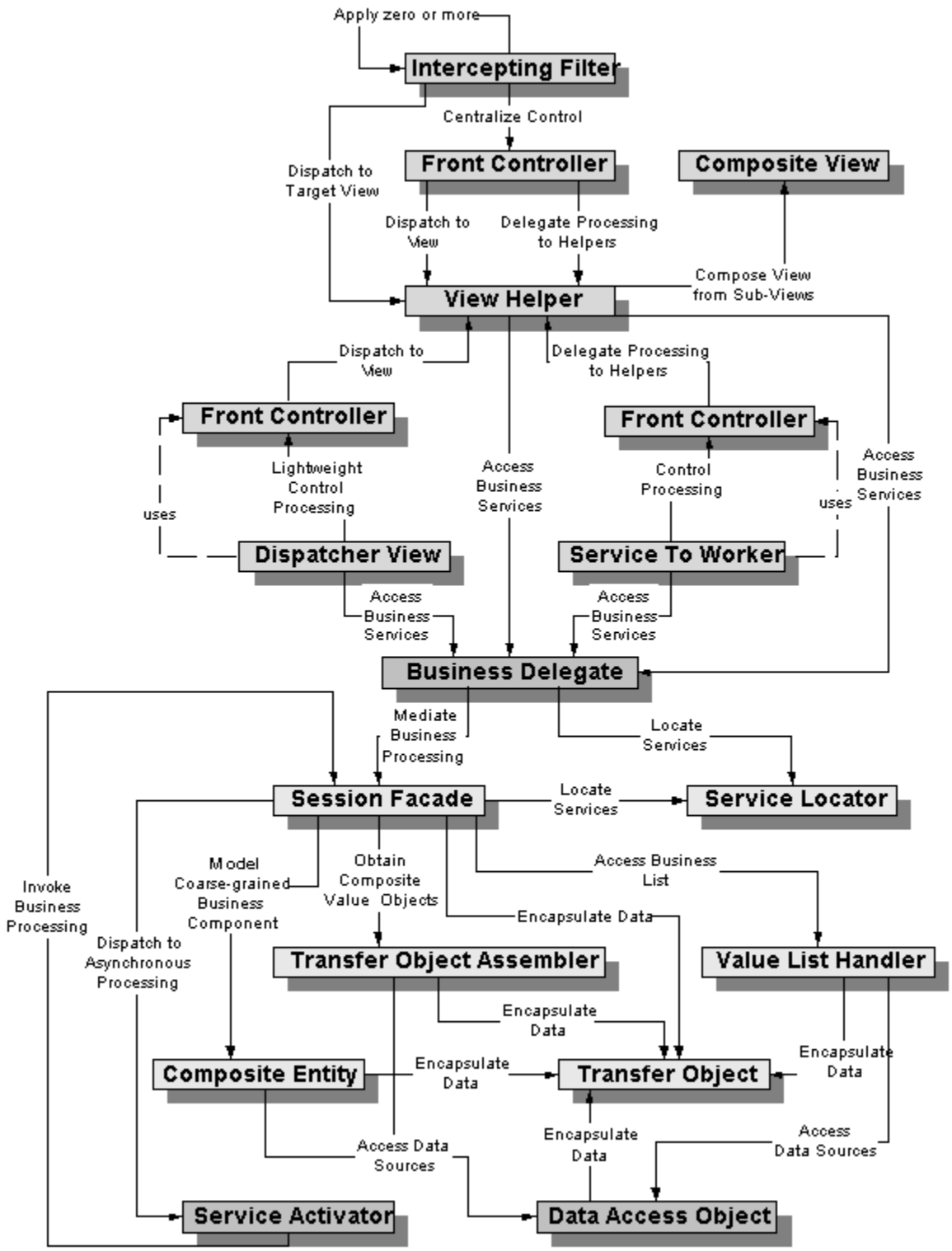


Figura 5-1. Esquema de Patrones J2EE.

El patrón Front Controller, el cual es parte del diseño de patrones de J2EE, describe como un controlador web debe ser implementado. Todas las peticiones y respuesta deben ir a través del controlador, aquí esta el punto centralizado de control para la aplicación web. Esto se tiene que tener en cuenta en cuestiones de mantenimiento y cuando añadimos nueva funcionalidad. El código que normalmente tendría que ser puesto

en cada página JSP debe ser puesto en el controlador del servlet ya que este procesa todas las peticiones. El controlador además ayuda a desacoplar los componentes de presentación (vistas) de las operaciones de negocio, lo que también ayuda al desarrollo.

Capítulo 6.

6. Struts

6.1 Introducción

El proyecto Jakarta Strus se creó para permitir que los desarrolladores pudieran crear aplicaciones web basándose en las tecnologías Java Servlet y JavaServer Pages (JSP). Una aplicación debe tener una base sólida a partir de la que pueda crecer el resto de la estructura. El framework Struts proporciona a los desarrolladores una infraestructura unificada sobre la que se pueden basar las aplicaciones de Internet permitiendo soluciones de calidad comercial basadas en la plataforma Java, en un entono abierto, cooperativo, eficaz y apropiado tanto para desarrolladores independientes como para grandes equipos de de desarrollo.

6.1.1 Capacidades de Struts

Strus nos facilita el desarrollo de aplicaciones ofreciéndonos las siguientes ventajas:

- Implementa el patrón MVC lo que nos provee de un Controlador, al Modelo y la Vista.
- El controlador ya se encuentra implementado por Struts, aunque si fuera necesario se puede heredar y ampliar o modificar
- El workflow de la aplicación se puede programar desde un archivo XML.
- Las acciones que se ejecutarán sobre el modelo de objetos de negocio se implementan basándose en clases predefinidas por el framework
- La generación de interfaz se soporta mediante un conjunto de Tags predefinidos por Struts cuyo objetivo es evitar el uso de Scriplets (los trozos de código Java entre "<%" y "%>"), lo cual genera ventajas de mantenibilidad.
- Logísticamente, separa claramente el desarrollo de interfaz del workflow y lógica de negocio permitiendo desarrollar ambas en paralelo o con personal especializado
- También es evidente que potencia la reutilización, soporte de múltiples interfaces de usuario (Html, sHtml, Wml, Desktop applications, etc.) y de múltiples idiomas, localismos, etc.
- Sin demasiada complejidad (curva de aprendizaje reducida), te permite extenderlo para adaptarlo a tus necesidades, fundamentalmente en empresas grandes. Hay además muchas extensiones estándar.
- Si las piezas que hay que construir son las mismas (acciones, formularios, jsps, tags ...) podemos crear una metodología de análisis y diseño orientado a objeto (el análisis no es Orientado a Objeto ... aunque puede ser amistoso) sacándole el máximo partido a este Framework.

6.2 Visión de Conjunto de una arquitectura

Las aplicaciones J2EE se pueden describir en términos de sus niveles. La funcionalidad de estos niveles se separa en estos niveles para proporcionar una separación de responsabilidad, reutilización, escalabilidad mejorada y muchas otras ventajas. En la siguiente figura se muestra los niveles en los que se puede utilizar una aplicación Struts:

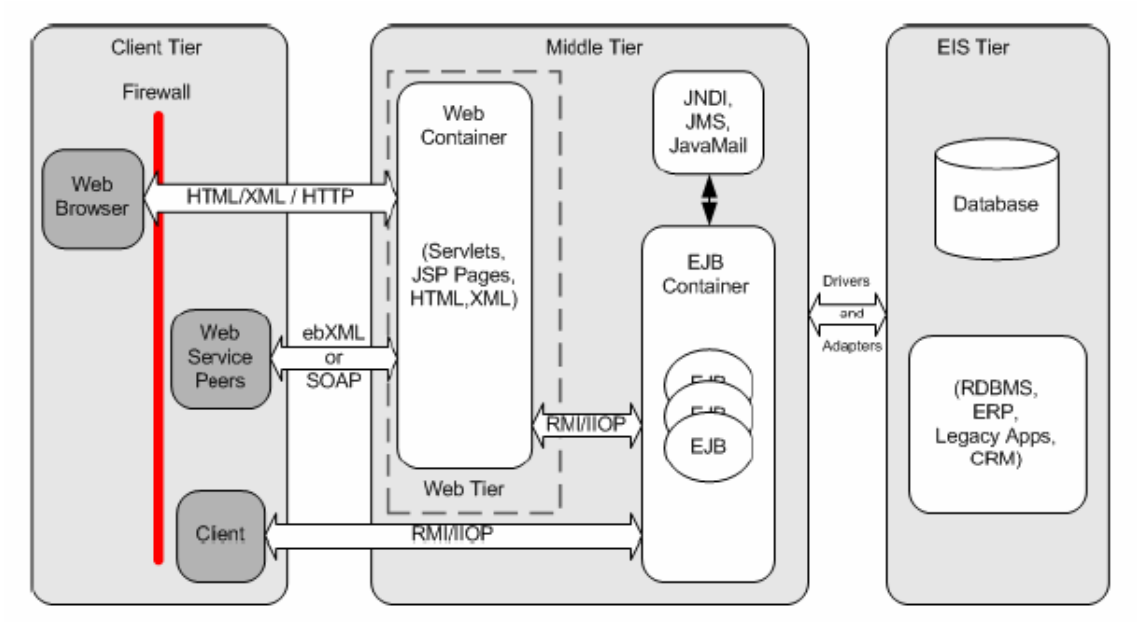


Figura 6-1. Capas funcionales de aplicación

Capa Cliente

Para struts el tipo de cliente básico es un explorador Web. Mediante el explorador el Cliente interactúa enviando y recibiendo algún tipo de respuesta del Middle Tier.

Capa Web

Este nivel permite al cliente comunicarse e interactuar con la lógica de la aplicación que reside en otros niveles. Actúa como un traductor y mapea peticiones http en invocaciones de servicio en el nivel medio.

La capa Web es responsable de gestionar el flujo de pantallas basándose en el estado de la aplicación y el usuario.

Los componentes que residen en el nivel Web permiten que los desarrolladores amplíen la funcionalidad básica de un servicio Web. En el caso de Struts esto se realiza por medio de los componentes del marco de trabajo que se ejecutan en un contador servlet.

Capa Media

No todas las aplicaciones de Struts tiene capa intermedia, algunas que son simples interactúan directamente con la capa de acceso a Datos. Fundamentalmente este nivel sirve para separar las responsabilidades de presentación de las del modelo y reglas de negocio para la aplicación.

Capa EIS (The Enterprise Information System) o de accesso a datos

Proporciona acceso a los recursos de la empresa, tales como BBDD, mainframes, sistemas de planificación de recursos.

La capa media se comunica con este nivel por medio de protocolos específicos de recursos. Por ejemplo, para comunicarse con una base de datos relacional, el nivel medio normalmente utilizara un driver JDBC.

Una vez vista la estructura general podemos ver donde se encuentra Struts:

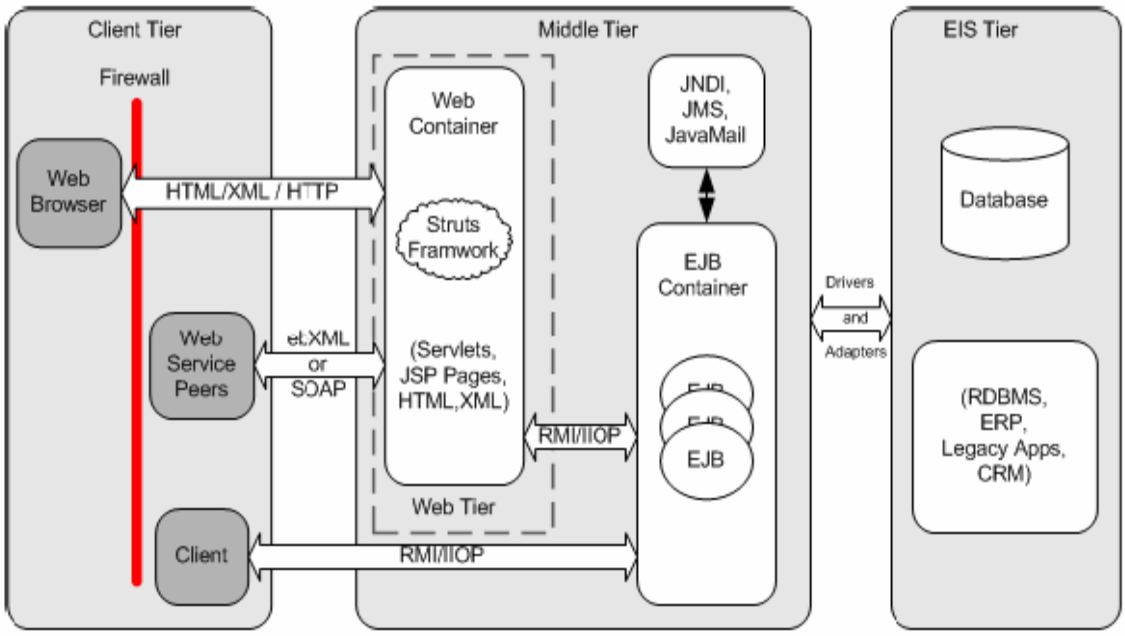


Figura 6-2 El framework Struts esta incrustado dentro de la capa Web.

Que como podemos ver reside en la capa Web.

La aplicación de Struts esta hospedada en un contenedor Web obteniendo los servicios de este, como comunicación por http o HTTPs

HTTP está basado en un modelo petición/respuesta, por lo que existen dos tipos de mensajes HTTP : la petición y la respuesta. El navegador abre una conexión a un servidor y realiza una petición. El servidor procesa la petición del cliente y devuelve la respuesta.

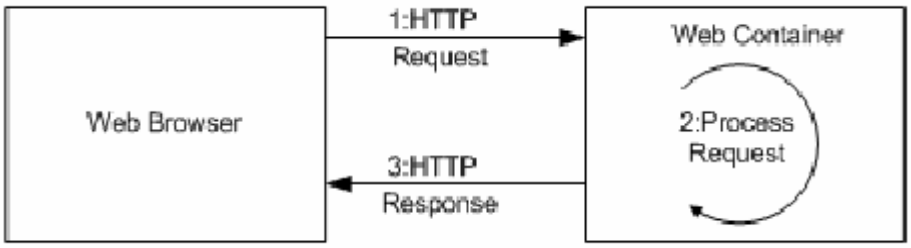


Figura 6-3. El modelo petición / respuesta HTTP.

6.3 Struts y ámbito

El Frameworks Struts utiliza varias áreas de recursos compartidos para almacenar objetos. Las áreas de recursos compartidos tienen una vida útil y una regla de visibilidad que definen el ámbito del recurso.

6.3.1 Ámbito de petición

Cada vez que un cliente lanza una petición HTTP, el servidor crea un objeto que implementa el interface: *javax.servlet.http.HttpServletRequest*

Este objeto contiene una colección de parejas atributo/valor que pueden ser usados para almacenar objetos durante el tiempo de vida de la petición. La clave para cada par es un *String* y el valor puede ser cualquier tipo de *Object*.

Los métodos para almacenar y obtener objetos en un ámbito de petición sesión son:

```
public void setAttribute( String name, Object obj );
public Object getAttribute( String name );
```

Una vez el servidor ha atendido la petición y ha enviando la respuesta al cliente, la petición pasa a ser procesada por el garbage collected de la máquina Virtual, muriendo.

Struts contiene funcionalidades para almacenar JavaBeans dentro de la petición, así pueden ser usados por componentes de presentación como páginas JSP. Esto hace mucho mas sencillo acceder a los datos de un JavaBeans, sin tener que preocuparnos por limpiar o remover los objetos más tarde. Esta tarea la hará el contenedor Web por nosotros.
Los objetos que son almacenados en una petición no son visibles por otra petición de otro cliente.

6.3.2 Ámbito de sesión

El siguiente nivel de visibilidad es el nivel de ámbito de sesión. El contenedor Web crea un objeto que implementa el interfaz *javax.servlet.http.HttpSession* para identificar a un usuario a lo largo de varias peticiones.

La sesión de usuario se va a mantener por un tiempo que se basa en la frecuencia que el usuario realiza peticiones (se puede configurar este parámetro de inactividad), aunque también se puede invalidar invocando al método invalidate() en el objeto de sesión.
En la sesión existe una colección de objetos que son almacenados basados en pares de atributo/valor, similar a los objetos de petición. La única diferencia entre unos objetos y otros es la duración, los objetos que se guardan en sesión permanecen más tiempo vivos que lo que se almacenan en la petición.
El framework de Struts usa atributos de sesión que extiende. Un ejemplo de un objeto que puede ser almacenado en un atributo de sesión es el objeto *Locale* del usuario. Los objetos almacenados en una sesión de usuario no son visibles para usuarios con una sesión diferente.

6.3.3 Ámbito de aplicación

En este nivel, la visibilidad y duración de los objetos almacenados es mayor. Aquí hay objetos que puede ser visibles por todos los clientes y threads de la aplicación web actual. Viven hasta que todos los programas desaparecen o cuando la aplicación termina.

El servidor crea un objeto que implemente el interface *javax.servlet.ServletContext* por cada y para todas las aplicaciones web que están instaladas en el contenedor. El objeto *ServletContext* almacena objetos y los regresa exactamente igual que los objetos que residen en ámbitos de petición y sesión con la diferencia que los objetos son visibles por toda la aplicación y persisten por el tiempo de vida de la aplicación. El frameWork de Struts usa ámbito de aplicación para almejar JavaBeans que necesitan ser visibles para todos los usuarios. Normalmente, los objetos son almacenados en este ámbito durante el inicio de la aplicación y persisten mientras la aplicación existe.

6.3.4 Page Scope

El último nivel tratado es referido a las páginas JSP, y se refiere al ámbito de página. Los objetos con ámbito de pagina son solo accesibles desde la pagina JSP que son creados.
Una vez la respuesta es enviada al cliente o la pagina redirige a otro recurso, el objeto ya no es accesible.
Los objetos son almacenado en el objeto *javax.servlet.jsp.PageContext* por cada página.

Todos las paginas JSP tiene un objeto implícito referenciado llamado pageContext que puede ser usado para almacenar y devolver objetos a nivel de pagina. Están incluidos los mismos métodos getAttribute() y setAttribute() que ofrecen los otros niveles de ámbito y su función es la misma.

6.4 Utilizar parámetros URL

Los parámetros URL son cadenas que se envían al servidor con la petición del cliente. Los parámetros se insertan en el objeto *HttpServletRequest* desde la cadena de consulta URI y los datos que se envían en un método POST. Los parámetros se formatean como pares clave/valor a los que acceden las aplicaciones como parámetros de petición.

El papel que juegan los URL parámetros es importante en todas las aplicaciones web y el Framework de Struts no es una excepción.

6.4.1 Reenviar vs redirigir

Una misma petición a veces es necesario que sea compartida por más de un componente, un servlet puede ser el responsable de autenticar y autorizar a un cliente, mientras que el trabajo de otro servlet diferente es recuperar datos para el usuario. El proceso de compartir una petición se puede realizar de varias formas diferentes.
Existen diferencias importantes entre como un contendor Web procesa la petición de reenvío frente a cómo procesa un redireccionamiento. El servlet de Struts siempre realiza o reenvío o redireccionamiento, pero no ambas a la vez.

6.4.2 Utilizar un redireccionamiento

Cuando se invoca el método sendRedirect(), hace que el contenedor Web devuelva al navegador una respuesta indicando que se debería solicitar una nueva URL. Puesto que el navegador envía una petición completamente nueva, todos los objetos que están almacenados como atributos de petición antes que ocurra el redireccionamiento se pierden. Esta es una de la principales diferencias entre un reenvío y redireccionamiento.

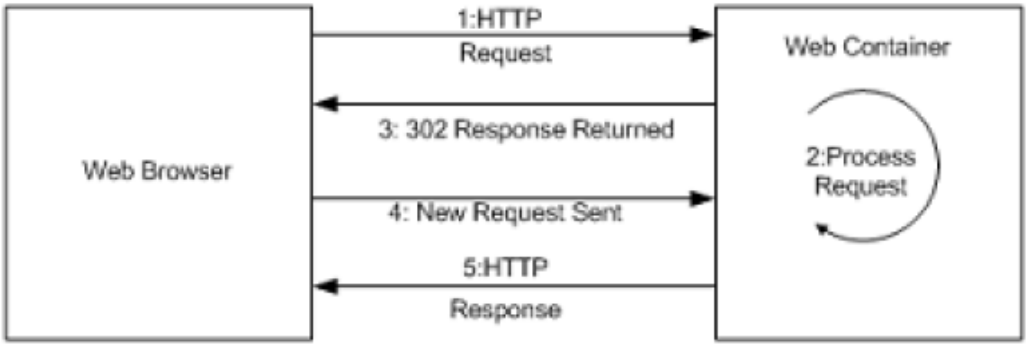


Figura 6-4. Un redireccionamiento ocasiona que el navegador dispare una nueva petición

Debido a esta ida y vuelta adicional que ocurre, un redirecionamiento es más lento que un reenvío.

6.4.3 Utilizar un reenvío

Un reenvío difiere de un redireccionamiento en varias formas. Cuando se invoca un reenvío para una petición, la petición se envía a otro recurso en el servidor sin que el cliente sea informado de que un recurso diferente va a procesar la petición. Este proceso ocurre dentro del contendor web sin visibilidad para el cliente que realiza la petición. A diferencia de con un redireccionamiento los objetos se pueden almacenar en la petición y pasarse al siguiente recurso a utilizar.

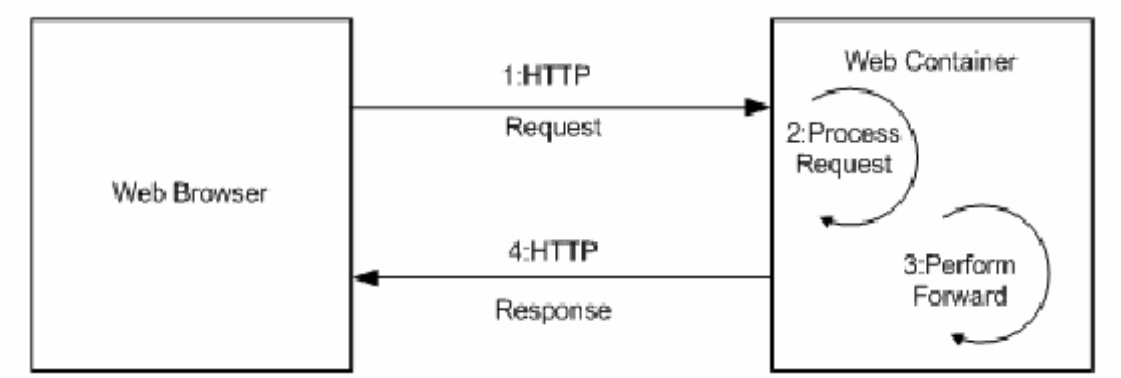


Figura 6-5. Durante un reenvío, la petición es enviada a un recurso secundario del servidor sin que el cliente se entere.

6.4.4 ¿Qué es mejor en Struts Reenviar o redirigir?

La opción por defecto que utiliza Struts es el reenvío, se comparten objetos en el ámbito de petición estando disponibles para los componentes de presentación y son mucho mas eficaces ya que el cliente no realiza una nueva petición. El reenvío plantea situaciones no controladas ante eventos de clic en el botón Actualizar del navegador del cliente, en este caso la opción de redirigir es más útil. Si se utiliza rutas de acceso relativas para imágenes y otros recursos en las páginas JSP, los reenvíos pueden ser problemáticos. Puesto que el navegador no tiene constancia que ha ocurrido un reenvío, las rutas de acceso relativas serán relativas al servlet inicial, no a la JSP a que se reenvía.

Describiremos la estructura de componentes del Framework de Struts desde el punto de vista del camino que sigue una petición por dicho Framework hasta que el servidor devuelve la respuesta. Por esta razón los componentes que realizan la función de controller en el framework serán los que veamos primero.

6.5 Los paquetes de componentes de Struts

El framework de Struts se compone de aproximadamente 300 clases Java divididas en 8 paquetes principales.

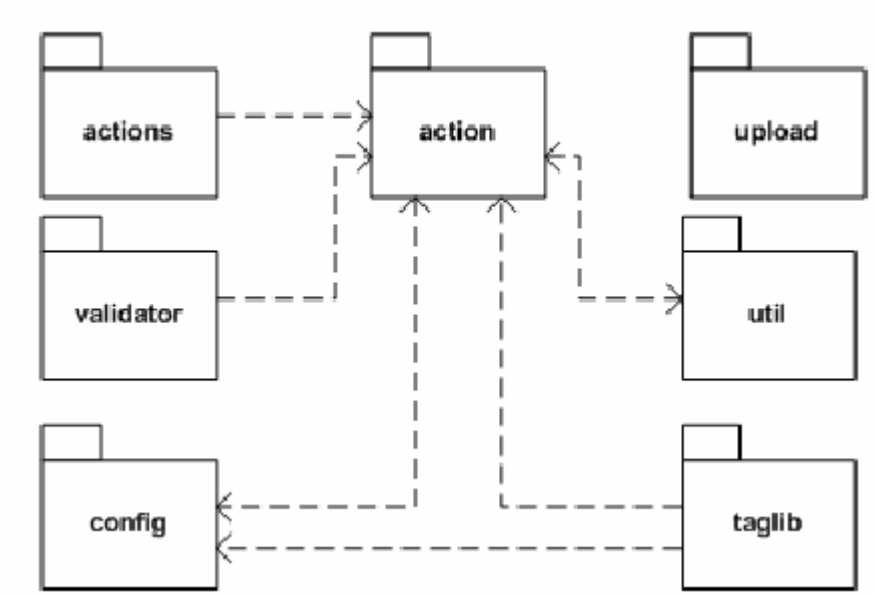


Figura 6-6. El framework Struts se compone de 7 paquetes principales

Entre todos estos paquetes se presenta un MVC. La distribución con respecto a la funcionalidad está distribuidita aleatoria mente. Por ejemplo el paquete *action* contiene clases para el controlador que algunas son usadas por el dominio de la parte de Vista.

Nombre de paquete	Descripción
action	Contiene las clases del controlador, ActionFrom, Action Messagges y algunos otros componentes obligatorios para Struts
Actions	Contiene las clases Action, tales como DispacthAction, que puede ser ampliada.
Config	Incluye las clases de configuración, que son representaciones en memoria del archivo de configuración de Struts.
Tablig	Contiene las clases de manejador de etiquetas para las librerías de etiquetas Struts.
Upload	Contiene las clases utilizadas para enviar o descargar archivos desde el sistema de archivos local utilizando un navegador.
Util	Contiene las clases de utilidades genéricas utilizadas por todo el marco.
Validator	Contiene las clases de ampliación específicas de Struts utlizadas por Struts cuando se despliega el validator. Las clases e interfaces de Validator se encuentran en el paquete commons, separadas de Struts.

6.5.1 Componentes de controlador Struts

La parte de controlador en Struts está implementada por un servlet de Java. Este servlet es el punto de control centralizado para la aplicación Web. Mapea las acciones del usuario en operaciones de negocio y luego ayuda a seleccionar la vista a devolver al cliente basándose en la petición y otra información de estado.

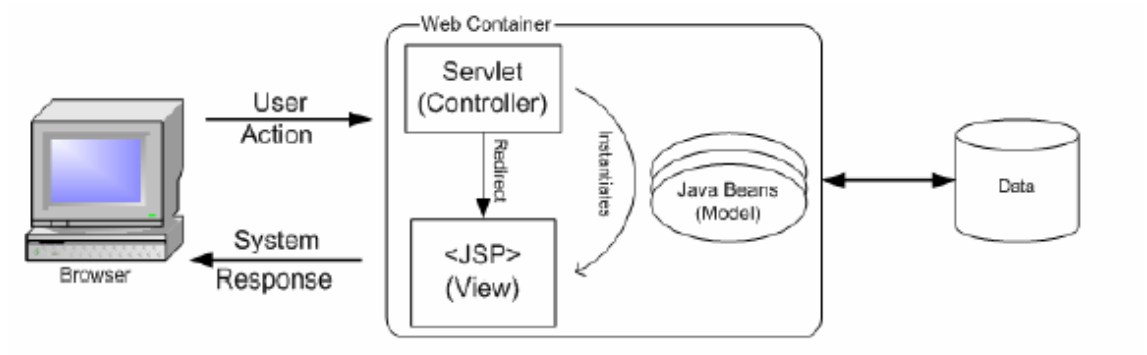


Figura 6-7. El framework de Strus utiliza un servlet de Java para manejar las responsabilidades del controlodor.

En Struts las responsabilidades del controlador están implementadas por varios componentes diferentes, uno de los cuales es una instancia de la clase `org.apache.struts.action.ActionServlet`.

La ActionServlet de Struts

La ActionServlet amplia la clase javax.servlet.http.HttpServlet y es responsable de empaquetar y enrutar tráfico http hasta el manejador adecuado dentro del marco de trabajo. La clase ActionServlet no es abstracta y de esta manera se puede utilizar como controlador concreto para las aplicaciones que se desarrollen. La clase denominada

org.apache.struts.action.RequestProcessor es quien procesa la petición para el controlador. La razón para desacoplar el procesamiento de la petición de la ActionServlet es proporcionarle la flexibilidad necesaria para que RequestProcessor sea una subclase de su propia versión y modificar así como se procesa la petición.
Una vez que el controlador recibe una petición del cliente, delega la gestión de la petición a una clase de ayuda. Esta clase de ayuda sabe como ejecutar la operación de negocio asociada con la acción solicitada. Esta clase de ayuda en Struts es procede de la clase org.apache.struts.action.Action .

Clases Action de Struts

La clase org.apache.struts.action.Action en el framework de Struts framework es una ampliación del componente de controlador. Actúa como paso entre una acción de usuario del lado del cliente y una operación de negocio. La clase Action desacopla la petición del cliente del modelo de negocio. También puede realizar otras funciones como autorización, conexión y validación de sesión antes de invocar la operación de negocio.
El método más importante de la clase Action es el método execute().

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
        throws IOException, ServletException;
```

El método execute() lo invoca el controlador cuando se recibe una petición de un cliente. El controlador crea una instancia de la clase Action si no existe ya alguna.
Solo se creara una instancia de la clase Action por cada aplicación. Se debe crear la propia implementación del método execute.

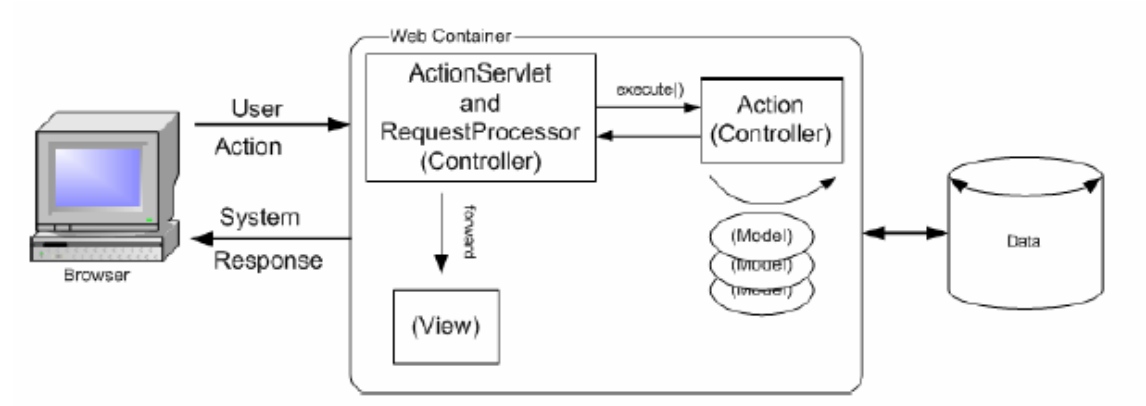


Figura 6-8. El método execute() de la clase Action es llamado por el controlador

Aquí se muestra la implementación de un método execute():

Example 3-3. The LoginAction used by the online banking application

```
package com.oreilly.struts.banking.action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.struts.action.*;
import com.oreilly.struts.banking.IConstants;
import com.oreilly.struts.banking.service.IAuthentication;
import com.oreilly.struts.banking.service.SecurityService;
import com.oreilly.struts.banking.service.InvalidLoginException;
import com.oreilly.struts.banking.view.UserView;
import com.oreilly.struts.banking.form.LoginForm;
/**
* This Action is called by the RequestProcessor when a login
attempt
* is made by the user. The ActionForm should be an instance of
*  a LoginForm and contain the credentials needed by the
SecurityService.
*/
public class LoginAction extends Action {
public ActionForward execute( ActionMapping mapping,
ActionForm form,
HttpServletRequest request,
HttpServletResponse response)
throws Exception {
UserView userView = null;
// Get the credentials from the LoginForm
String accessNbr = ((LoginForm)form).getAccessNumber();
String pinNbr = ((LoginForm)form).getPinNumber();
/*
* In a real application, you would typically get a reference
* to a security service through something like JNDI or a factory.
*/
IAuthentication service = new SecurityService();
// Attempt to login
userView = service.login( accessNbr, pinNbr );
// Since an exception wasn't thrown, login was successful
// Invalidate existing session if it exists
HttpSession session = request.getSession(false);
if(session != null) {
session.invalidate();
}
// Create a new session for this user
session = request.getSession(true);
// Store the UserView into the session and return
session.setAttribute( IConstants.USER_VIEW_KEY, userView );
return mapping.findForward( IConstants.SUCCESS_KEY );
}
}
```

LoginAction obtiene las credenciales del ActionForm que se paso como argumento en el método execute().
Si el usuario se ha logado correctamente se creara nueva HttpSession, pero si falla la autentificación se envía una InvalidLoginException. En el método execute no existe ningún método de captura para InvalidLoginExcepcion en el método execute().
Esto se debe a que Struts permite la gestión declarativa de excepciones. De está manera Struts puede especificar qué excepciones se pueden enviar desde las acciones y qué es lo que quiere que el marco haga con ellas. Esta información de manejo de excepciones se configura en el archivo de configuración de Struts:

```
<global-exceptions>
<exception
key="global.error.invalidlogin"
path="/login.jsp"
scope="request"
type="com.oreilly.struts.banking.service.InvalidLoginException"/>
</global-exceptions>
```

Este fragmento indica que si se produce en cualquier acción un envía al método InvalidLoginExcepcion se enviara el flujo de ejecución a la página login.jsp y creara un mensaje de error utilizando la clave global.error.invalidlogin desde el recurso.

Mapear las acciones

Para unir las acciones que ofrecemos al usuario con los action creados utilizamos lo que se denomina mapeados de acción. Esta información se guarda en un archivo XML de configuración. Está información de configuración se carga en memoria cuando se inicia y se encuentra disponible para el framework en tiempo de ejecución. Cada elemento action esta representado en memoria por una instancia de la clase org.apache.struts.action.ActionMapping
El objeto ActionMapping contiene un atributo path que se hace coincidir con una parte del URI de la petición entrante.

```
<action
path="/login"
type="com.oreilly.struts.banking.action.LoginAction"
scope="request"
name="loginForm"
validate="true"
input="/login.jsp">
<forward       name="Success"       path="/action/getaccountinformation"
redirect="true"/>
<forward name="Failure" path="/login.jsp" redirect="true"/>
</action>
```

El mapeado de acción login mostrado aquí mapea la ruta de acceso "/login" a la clase Action com.oreilly.struts.banking.LoginAction. Siempre que el controlador recibe una petición donde la ruta de acceso en el URI contenga la cadena "/login", el método execute() de la instancia LoginAction se invoca.
Struts también utiliza los mapeados para identificar el recurso al que reenviar al usuario una vez que se ha completado la acción.

Determinar la siguiente vista

Una vez que hemos tratado la petición en el Action, el Action devuelve una clase de tipo org.apache.struts.action.ActionForward. La clase ActionForward representa un destino al que el controlador puede enviar el control una vez que se haya completado una acción.
La página JSP donde nos llevara el flujo no se codifica en el código sino que se asocia declarativamente en un mapeado de reenvío de acción con la JSP y luego reutilizar esa ActionForward en toda la aplicación.
Los reenvíos de acción se especifican en el archivo de configuración, igual que los mapeados de acción:

```
<action
path="/logout"
type="com.oreilly.struts.banking.action.LogoutAction"
scope="request">
<forward name="Success" path="/login.jsp" redirect="true"/>
</action>
```

La acción logout declara un elemento <forward> denominadao "Success" que reenvía al recurso *"/login.jsp"*. El atributo redirect está especificado y establecido en "true". Ahora en lugar de realizar un reenvío utilizando un RequestDispatcher la petición se redimirá.

Los mapeados de reenvío de acción también se pueden especificar en un apartado global, independientes de cualquier mapeado específico de acción. En el caso anterior, solamente el mapeado de acción *logout* podía hacer referencia al reenvío de acción denominado "Success". Sin embargo todos los mapeados de acción pueden hacer referencia a reenvíos, declarados en la parte de sección denominada global-forwards. Aquí tenemos un ejemplo de este tipo de reenvíos:

```
<global-forwards>
<forward name="SystemFailure" path="/systemerror.jsp" />
<forward name="SessionTimeOut" path="/sessiontimeout.jsp" />
</global-forwards>
```

Los reenvíos definidos en este apartado son más generales y no se aplican a mapeados específicos de acción. Los atributos obligatorios son name y path , el atributo redirect es opcional., si no se especifica ningún valor, se utiliza “false” por defecto, por lo que re se realizara un reenvío. El atributo patch en un reenvío de acción también puede especificar otra Action de Struts.

Una vez vistos los componentes del controlador de Struts veremos ahora el Modelo (la M dentro de la especificación MVC)

6.5.2 Componentes de Modelo Struts

Struts no tiene demasiado soporte en la forma de componentes de modelo.
Se debe separar los objetos de negocio de la presentación, así las vistas de presentación deben simplemente recuperar datos de los objetos de transferencia de datos y mostrarlos.

La clase Action es parte de la funcionalidad del controlador y tiene que delegar la gestión de la lógica de negocio a otros servicios. De esta manera mantenemos la lógica de negocio fuera de las clases Action para mantenernos protegidos frente a los cambios.
Para comunicar el Action con la lógica de negocio se utilizan los objetos de transferencia de datos DTO. Por medio de estos objetos Struts ayuda a desacoplar los objetos de negocio de la presentación facilitando el mantenimiento y mejoras futuras.

6.5.3 Componentes de vista de Struts

Es el último de los componentes MVC. Los componentes de vista empleados normalmente en una aplicación Struts son:

- Html
- Objetos de transferencia de datos
- ActionFrom de Struts
- JavaServer Pages
- Etiquetas personalizadas
- Paquetes de recursos de Java

Utilizar ActionForm de Struts

Los objetos ActionForm de Struts se utilizan en el marco para pasar los datos de entrada del cliente entre el usuario y la capa de negocio. El framework recupera automáticamente la entrada de datos de la petición y pasa estos datos a una Action utilizando un form bean, que luego se puede pasar a la capa de negocio. Para mantener la capa de presentación separada de la capa de negocio; en su lugar hay que crear el DTO apropiado utilizando los datos de la ActionForm.

Para toda página HTML donde se publiquen datos de formulario se debe utilizar una ActionForm. La misma ActionForm puede ser utilizada por otras páginas HTML siempre y cuando coincidan los campos HTML y las propiedades ActionForm.

Aquí vemos un ejemplo de ActionForm:

```
package com.oreilly.struts.banking.form;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.*;
import org.apache.struts.util.MessageResources;
/**
* This ActionForm is used by the online banking appliation to validate that the user
has entered an accessNumber and a pinNumber. If one or both of the fields are empty
when validate is called by the ActionServlet, error messages are created.
*/
public class LoginForm extends ActionForm {
               private String pinNumber; // The user's private id number
               private String accessNumber; // The user's access number

/**
* Default Constructor
*/
public LoginForm() {
super();
resetFields();
}
/**
* Called by the framework to validate the user has entered the
* accessNumber and pinNumber fields.
*/
public ActionErrors validate(ActionMapping mapping, HttpServletRequest req ){
ActionErrors errors = new ActionErrors();

// Get access to the message resources for this application
// There's not an easy way to access the resources from an ActionForm
// This assumes our messages are coming from the default MessageResources
MessageResources resources=(MessageResources)req.getAttribute( Action.MESSAGES_KEY);
// Check and see if the access number is missing
if(accessNumber == null || accessNumber.length() == 0) {
       String accessNumberLabel = resources.getMessage( "label.accessnumber" );
// Create an error message for the missing accessNumber value
ActionError newError = new ActionError("global.error.login.requiredfield", "");
errors.add(ActionErrors.GLOBAL_ERROR, newError);
}
// Check and see if the pin number is missing
if(pinNumber == null || pinNumber.length() == 0) {
       String pinNumberLabel = resources.getMessage( "label.pinnumber" );
// Create an error message for the missing pingNumber value
ActionError   newError=   new   ActionError("global.error.login.requiredfield",   "Pin
Number");
errors.add(ActionErrors.GLOBAL_ERROR, newError);
}
// Return the ActionErrors, in any.
return errors;
}
/**
* Called by the framework to reset the fields back to their default values.
*/
public void reset(ActionMapping mapping, HttpServletRequest request) {
// Clear out the access number and pin number fields
resetFields();
}
/**
* Reset the fields back to their defaults.
*/
protected void resetFields() {
this.accessNumber = "";
this.pinNumber = "";
}
public void setAccessNumber(String nbr) {
this.accessNumber = nbr;
}
public String getAccessNumber() {
return this.accessNumber;
}
public String getPinNumber() {
return this.pinNumber;
}
public void setPinNumber(String nbr) {
this.pinNumber = nbr;
}
}
```

La clase ActionForm que proporciona el entorno de desarrollo Struts implemente varios métodos pero los más importantes son reset() y validate (). Ambos se tienen que codificar. El controlador llama al método reset () justo antes de completar la instancia ActionForm con los valores de la petición. El método validate() lo invoca el controlador después que los valores de la petición se hayan insertado en la ActionForm. La validación de lógica de negocio se debería de llevar a cabo en los objetos de negocio, no en la ActionForm, aquí solo se tiene que codificar validación de presentación.

Las clases ActionForm se configuran en la aplicación de Struts, para decir que mapeados de acción deberían utilizar que ActionForm:

```
<form-beans>
<form-bean                                                  name="loginForm"
type="com.oreilly.struts.banking.form.LoginForm"/>
<form-bean
name="accountInformationForm"
dynamic="true"
type="org.apache.struts.action.DynaActionForm">
<form-property name="accounts" type="java.util.ArrayList"/>
</form-bean>
<form-bean
name="accountDetailForm"
dynamic="true"
type="org.apache.struts.action.DynaActionForm">
<form-property
name="view"
type="com.oreilly.struts.banking.view.AccountDetailView"/>
</form-bean>
</form-beans>
```

Después se realiza el mapeo de uno de los nombres form-bean de la sección form-beans en uno o más elementos action.

```
<action
path="/login"
type="com.oreilly.struts.banking.action.LoginAction"
scope="request"
name="loginForm"
validate="true"
input="/login.jsp">
<forward        name="Success"        path="/action/getaccountinformation"
redirect="true"/>
<forward name="Failure" path="/login.jsp" redirect="true"/>
</action>
```

Los componentes de vista pueden utilizar tanto ActionForm como DTO para completar contenido dinámico. Cuando no se configura ninguna ActionForm para un mapeado, puede utilizar los DTO para crear las vistas. Y cuando se define un form bean para el mapeado, existen varias formas de gestionar la extracción de datos desde el bean. Un enfoque es situar un form-bean en torno a uno o mas DTO devueltos desde el nivel de negocio y obligar a los componentes de vista de Struts a acceder a los datos del DTO por medio del form bean. De igual forma cuando un cliente envia una página HTML, Struts invocara los método set de form bean que pueden devolver los datos al DTO después de que el método de validación se haya completado con éxito. Esto proporciona una única interfaz para que las vistas puedan recuperar y enviar datos HTML.

Utilizar JSP para la presentación

Las páginas JSP comprenden casi todo lo que se tiene que crear para los componentes de vista de Struts. Combinadas con librerías de etiquetas personalizadas y HTML, JSP hace más sencillo el hecho de proporcionar un conjunto de vistas para una aplicación. Aunque la utilización de JSP presenta incomodidades como: los desarrolladores pueden incorporar lógica de negocio en las propias JSP, se debe aprender la sintaxis JSP así como programar etiquetas personalizadas y por ultimo, el contenedor debe recompilar las páginas JSP cuando se realiza un cambio en la página.

Además de JSP existen otras tecnologías de presentación que se pueden combinar con el entorno de trabajo de Struts. Una alternativa es la combinación de XMI/XSlT. Este modelo combina el servlet de controlador de Struts con XSLT y beans en serie de los DTO para mostrar vistas.

Utilizar librerías de etiquetas personalizadas

Struts proporciona seis librerías de etiquetas principales para utilizar en las aplicaciones. Cada una de ellas tiene una finalidad diferente. También se puede ampliar las etiquetas Struts o crear sus propias etiquetas personalizadas.

El siguiente ejemplo vemos la utilización de las etiquetas:

```
<%@ page language="java" contentType="text/html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<html:html>
<head>
<html:base/>
<title><bean:message key="title.login"/></title>
<link rel="stylesheet" href="stylesheets/login_style_ie.css" type="text/css">
</head>
<body topmargin="0" leftmargin="5" marginheight="0" marginwidth="0"
bgcolor="#6699FF">
<html:form action="login" focus="accessNumber">
<table border="0" cellpadding="0" cellspacing="0" width="100%" bgcolor="#6699FF">
<tr><td>
<html:img srcKey="image.logo" width="79" height="46"
altKey="image.logo.alt" border="0"/>
</td></tr>
</table>
<table border="0" cellpadding="0" cellspacing="0" width="100%">
<tr><td bgcolor="#000000">
<table border="0" cellpadding="0" cellspacing="0" width="1" height="2"></table>
</td></tr>
</table>
<table border="0" cellpadding="0" cellspacing="0" width="1" height="1">
<tr><td></td></tr>
</table>
<table>
<tr><td></td></tr>
</table>
<table border="0" cellpadding="0" cellspacing="0" width="590">
<tr><td width="15" height="31"></td><td width="12"></td></tr>
<tr>
<td width="15"></td>
<td width="575" bgcolor="#FFFFFF" colspan="2">
<table cellpadding="0" cellspacing="0" border="0" width="575" height="3">
<tr><td></td></tr>
</table>
</td>
</tr>
</table>
<table border="0" cellpadding="0" cellspacing="0" width="590" bgcolor="#ffffff">
<tr>
<td width="15" bgcolor="#6699FF"></td>
<td width="15"></td><td width="379"></td>
<td width="15"></td>
<td width="15"></td>
<td width="15"></td>
</tr>
<tr>
<td bgcolor="#6699FF" width="15"></td>
<td></td>
<td valign="top">
<table border="0" cellpadding="0" cellspacing="0">
<tr class="fieldlabel">
<td><bean:message key="label.accessnumber"/></td>
</tr class="fieldlabel">
<tr>
<td><html:text property="accessNumber" size="9" maxlength="9"/></td>
<td class="error"><html:errors/></td>
</tr>
<tr class="fieldlabel"><td height="10"></td></tr>
<tr class="fieldlabel"><td><bean:message key="label.pinnumber"/></td></tr>
<tr class="fieldlabel">
<td><html:password property="pinNumber" size="4" maxlength="4"/></td>
</tr>
```

```
<tr><td height="10"></td></tr>
<tr><td><html:submit styleClass="fieldlabel" value="Login"/></td></tr>
<tr><td></td></tr>
</table>
</td>
<td width="151" valign="top">
<html:img srcKey="image.strutspower" altKey="image.strutspower.alt"/>
</td>
</tr>
</table>
<%@include file="include/footer.jsp"%>
<br>
</html:form>
</body>
</html:html>
```

Al utilizar etiquetas de HTML y de Struts en las páginas JSP no es necesaria ninguna programación en Java, por lo que los diseñadores HTML pueden trabajar libremente con el diseño de la página sin tener que tener conocimientos de programación.

Utilizar paquetes de recursos de mensaje

El entorno de trabajo de Struts proporciona un conjunto de clases para realizar la lectura de recursos de mensaje de una clase Java o un archivo de propiedades. Estas clases surgen a partir de org.apache.struts.util.MessageResources.
Existe un paquete de recursos para cada idioma dentro de una aplicación Struts.

Aquí se puede observar un archivo de propiedades:

```
# Labels
label.accessnumber=Access Number
label.pinnumber=Pin Number
label.accounts=Accounts
label.balance=Balance
label.totalassets=Total Assets
label.account=Account
label.balance=Available Balance
label.description=Description
label.amount=Amount
label.deposits=Deposits
label.withdrawls=Withdrawls
label.openingbalance=Opening Balance
# Links
link.customeragreement=Customer Agreement
link.privacy=Privacy
link.security=Security
link.viewaccountdetail=View Account Detail
# Page Titles
title.login=Struts Online Banking - Account Login
title.accountinfo=Struts Online Banking - Account Information
title.accountdetail=Struts Online Banking - Account Detail
# Button Labels
label.button.login=Login
# Error messages
global.error.invalidlogin=<li>Invalid Access Number and/or Pin</li>
global.error.login.requiredfield=<li>The {0} is required for login</li>
# Images
image.logo=images/logo.gif
image.logo.alt=Struts Online Banking
image.logout=images/logout.gif
image.logout.alt=Logout
image.strutspower=images/struts-power.gif
image.strutspower.alt=Powered By Struts
image.transfer=images/transfer.gif
image.transfer.alt="Transfer Funds"
image.clear=images/clear.gif
```

Las páginas JSP pueden recuperar valores del paquete de recursos utilizando MessageTag basada en una clave, lo que permite tener centralizados los literales.
Aquí tenemos la línea que carga el literal en el jsp:

```
<title><bean:message key="title.login"/></title>
```

6.5.4 Soporte de múltiples aplicaciones

Antes de la versión 1.1 de Struts, una aplicación hecha con Struts estaba limitada a un solo archivo de configuración, normalmente denominado struts-confing.xml que era el único que estaba especificado en el descriptor de despliegue de la aplicación Web.
Struts permite definir multiples archivos de configuración, permitiendo que los desarrolladores trabajen mejor en paralelo.

Capítulo 7.

7. Java Server Faces

7.1 Introducción

La tecnología JavaServer Faces es un Framework de interfaces de usuario que actúan en el lado de servidor para aplicaciones Web basadas en tecnología Java.

Los principales componentes de la tecnología JavaServer Faces son:

- Un API y una implementación de referencia para: representar componentes de interfaces de usuario (IU, o en ingles UI User Interface) y manejar su estado; manejo de eventos, validación del lado del servidor y conversión de datos; definir la navegación entre páginas; soportar internacionalización y accesibilidad; y proporcionar extensibilidad para todas estas características.
- Una librería de etiquetas JavaServer Pages (JSP) personalizadas para dibujar componentes IU dentro de una página JSP.

Este modelo de programación bien definido y la librería de etiquetas para componentes (IU) facilita de forma significativa la tarea de la construcción y mantenimiento de aplicaciones Web con IUs del lado del servidor. Con un mínimo esfuerzo, podemos:

- Conectar eventos generados en el cliente a código de la aplicación en el lado del servidor.
- Mapear componentes IU a una página de datos del lado del servidor.
- Construir un IU con componentes reutilizables y extensibles.
- Grabar y restaurar el estado del IU más allá de la vida de las peticiones de servidor.

Como se puede apreciar en la siguiente figura, el interface de usuario que creamos con la tecnología JavaServer Faces (representado por myUI en el gráfico) se ejecuta en el servidor y se renderiza en el cliente.

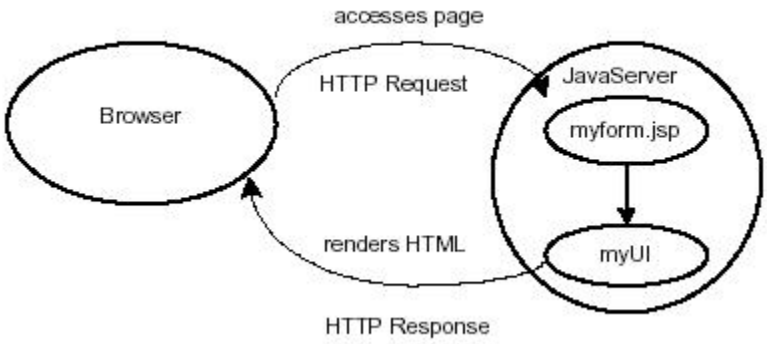


Figura 7-1. Esquema general de envío de una petición en Java Server Faces.

La página JSP, **myform.jsp**, dibuja los componentes del interface de usuario con etiquetas personalizadas definidas por la tecnología JavaServer Faces. El IU de la aplicación Web (representado por **myUI** en la imagen) maneja los objetos referenciados por la página JSP:

- Los objetos componentes que mapean las etiquetas sobre la página JSP.
- Los oyentes de eventos, validadores, y los conversores que está registrados en los componentes.
- Los objetos del modelo que encapsulan los datos y las funcionalidades de los componentes específicos de la aplicación.

7.1.1 Beneficios de la Tecnología JavaServer Faces

Una de las grandes ventajas de la tecnología JavaServer Faces es que ofrece una clara separación entre el comportamiento y la presentación. Las aplicaciones Web construidas con tecnología JSP conseguían parcialmente esta separación. Sin embargo, una aplicación JSP no puede mapear peticiones HTTP al manejador de eventos específicos del componente o manejar elementos IU como objetos con estado en el servidor. La tecnología JavaServer Faces nos permite construir aplicaciones Web que implementan una separación entre el comportamiento y la presentación tradicionalmente ofrecidas por la arquitectura IU del lado del cliente.
La separación de la lógica de la presentación también le permite a cada miembro del equipo de desarrollo de una aplicación Web enfocarse en su parte del proceso de desarrollo, y proporciona un sencillo modelo de programación para enlazar todas las piezas. Por ejemplo, los desarrolladores de páginas sin experiencia en programación pueden usar las etiquetas de componentes IU de la tecnología JavaServer Faces para enlazar código de la aplicación desde dentro de la página Web sin escribir ningún script.
Otro objetivo importante de la tecnología JavaServer Faces es mejorar los conceptos familiares de componente-IU y capa-Web sin limitarnos a una tecnología de script particular o un lenguaje de marcas. Aunque la tecnología JavaServer Faces incluye una librería de etiquetas JSP personalizadas para representar componentes en una página JSP, los APIs de la tecnología JavaServer Faces se han creado directamente sobre el API JavaServlet. Esto nos permite hacer algunas cosas: usar otra tecnología de presentación junto a JSP, crear nuestros propios componentes personalizados directamente desde las clases de componentes, y generar salida para diferentes dispositivos cliente.

Pero lo más importante, la tecnología JavaServer Faces proporciona una rica arquitectura para manejar el estado de los componentes, procesar los datos, validar la entrada del usuario, y manejar eventos.

7.2 ¿Qué es una Aplicación JavaServer Faces?

En su mayoría, las aplicaciones JavaServer Faces son como cualquier otra aplicación Web Java. Se ejecutan en un contenedor Servlet Java, y típicamente contienen:

- Componentes JavaBeans (llamados objetos del modelo en tecnología JavaServer Faces) conteniendo datos y funcionalidades específicas de la aplicación.
- Oyentes de Eventos.
- Páginas, cómo páginas JSP.
- Clases de utilidad del lado del servidor, como beans para acceder a las bases de datos.

Además de estos ítems, una aplicación JavaServer Faces también tiene:

- Una librería de etiquetas personalizadas para dibujar componentes IU en una página.
- Una librería de etiquetas personalizadas para representar manejadores de eventos, validadores, y otras acciones.
- Componentes IU representados como objetos con estado en el servidor.
- Validadores, manejadores de eventos y manejadores de navegación.

Toda aplicación JavaServer Faces debe incluir una librería de etiquetas personalizadas que define las etiquetas que representan componentes IU y una librería de etiquetas para representar otras acciones importantes, como validadores y manejadores de eventos. La implementación de JavaServer Faces proporciona estas dos librerías.

- La librería de etiquetas de componentes elimina la necesidad de codificar componentes IU en HTML u otro lenguaje de marcas, resultando en componentes completamente reutilizables. Y, la librería "core" hace fácil registrar eventos, validadores y otras acciones de los componentes.
- La librería de etiquetas de componentes puede se la librería *html_basic* incluida con la implementación de referencia de la tecnología JavaServer Faces, o podemos definir nuestra propia librería de etiquetas que dibuje componentes personalizados o que dibuje una salida distinta a HTML.

Otra ventaja importante de las aplicaciones JavaServer Faces es que los componentes de IU de la página están representados en el servidor como objetos con estado. Esto permite a la aplicación manipular el estado del componente y conectar los eventos generados por el cliente a código en el lado del servidor.
Finalmente, la tecnología JavaServer Faces nos permite convertir y validar datos sobre componentes individuales y reportar cualquier error antes de que se actualicen los datos en el lado del servidor.

7.3 Una Sencilla Aplicación JavaServer Faces

Describiremos el proceso de desarrollo de una sencilla aplicación JavaServer Faces. Veremos qué características contiene una típica aplicación JavaServer Faces, y qué hace cada rol en el desarrollo de la aplicación.

7.3.1 Pasos del Proceso de Desarrollo

Desarrollar una sencilla aplicación JavaServer Faces requiere la realización de estos pasos:

- Desarrollar los objetos del modelo, los que contendrán los datos.
- Añadir las declaraciones del bean controlado al fichero de configuración de la aplicación.
- Crear las páginas utilizando las etiquetas de componentes de IU y las etiquetas "core".
- Definir la navegación entre las páginas.

Estas tareas se pueden realizar simultáneamente o en cualquier orden. Sin embargo, quien realice las tareas necesitará comunicarse durante el proceso de desarrollo. Por ejemplo, el autor de las páginas necesita saber los nombres de los objetos del modelo para poder acceder a ellos desde la página.
Este ejemplo nos pedirá que adivinemos un número del 0 al 10, ambos inclusive. La segunda página nos dirá si hemos acertado. El ejemplo también chequea la validez de nuestra entrada.

7.3.2 Desarrollar los Objetos del Modelo

Desarrollar los objetos del modelo es responsabilidad del desarrollador de aplicaciones. El autor de las páginas y el desarrollador de aplicaciones podrían necesitar trabajar juntos para asegurarse que las etiquetas de componentes se refieren a las propiedades del objeto apropiado, que las propiedades del objeto son de los tipos apropiados, y para tener cuidado de otros detalles.
Aquí tenemos la clase UserNumberBean.java que contiene los datos introducidos en el campo de texto de la página greeting.jsp:

```
package guessNumber;
import java.util.Random;

public class UserNumberBean {
    Integer userNumber = null;
    Integer randomInt = null;
    String response = null;

    public UserNumberBean () {
        Random randomGR = new Random();
        randomInt = new Integer(randomGR.nextInt(10));
        System.out.println("Duke’s Number: "+randomInt);
    }

    public void setUserNumber(Integer user_number) {
        userNumber = user_number;
        System.out.println("Set userNumber " + userNumber);
    }

    public Integer getUserNumber() {
        System.out.println("get userNumber " + userNumber);
        return userNumber;
    }

    public String getResponse() {
        if(userNumber.compareTo(randomInt) == 0)
            return "Yay! You got it!";
        else
            return "Sorry, "+userNumber+" is incorrect.";
    }
}
```

Como podemos ver, este bean es como cualquier otro componente JavaBeans. Tiene un método set o accesor y un campo privado o propiedad. Esto significa que podemos concebir referenciar beans que ya hayamos escrito desde nuestras páginas JavaServer Faces.
Dependiendo del tipo de componente que referencia una propiedad del objeto del modelo, esta propiedad puede ser de cualquiera de los tipos básicos primitivos y los tipos referencia. Esto incluye cualquiera de los tipos numéricos, String, int, double, y float. La tecnología JavaServer Faces convertirá automáticamente el dato al tipo especificado por la propiedades del objeto del modelo.
También podemos aplicar una conversión a un componente para convertir los valores de los componentes a un tipo que no esté soportado por el componente.
En el UserNumberBean, la propiedad userNumber es del tipo Integer. La implementación de JavaServer Faces puede convertir el String de los parámetros de la solicitud que contiene este valor a un Integer antes de actualizar la propiedad del objeto del modelo cuando utilicemos una etiqueta input_number. Aunque este ejemplo lo convierte a un Integer, en general, deberíamos utilizar tipos nativos en vez de utilizar las clases envoltura (int en lugar de Integer).

7.3.3 Añadir las Declaraciones del Bean Controlado

Después de desarrollar los beans utilizados en la aplicación, necesitamos añadir declaraciones para ellos en el fichero de configuración de la aplicación. Cualquier miembro del equipo de desarrollo puede realizar la tarea de añadir las declaraciones al fichero de configuración de la aplicación. Aquí tenemos la declaración de bean controlado para UserNumberBean:

```
<managed-bean>
    <managed-bean-name>UserNumberBean</managed-bean-name>
        <managed-bean-class>
            guessNumber.UserNumberBean
        </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

La implementación de JavaServer Faces procesa este fichero en el momento de arrancada de la aplicación e inicializa el UserNumberBean y lo almacena en el ámbito de sesión. Entonces el bean estará disponible para todas las páginas de la aplicación. Para aquellos que estén familiarizados con versiones anteriores, esta facilidad de "bean controlado" reemplaza la utilización de la etiqueta jsp:useBean.

7.3.4 Crear las Páginas

La creación de las páginas es responsabilidad del autor de páginas. Esta tarea implica distribuir los componentes IU en las páginas, mapear los componentes a los datos de los objetos del modelo, y añadir otras etiquetas importantes (como etiquetas del validador) a las etiquetas de los componentes. Aquí tenemos la página greeting.jsp con las etiquetas de validador (menos los HTML que lo rodea):

```
<HTML>
<HEAD> <title>Hello</title> </HEAD>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<body bgcolor="white">
<h:graphic_image id="wave_img" url="/wave.med.gif" />
<h2>Hi. My name is Duke.
I'm thinking of a number from 0 to 10.
Can you guess it?</h2>
<f:use_faces>
    <h:form id="helloForm" formName="helloForm" >
        <h:graphic_image id="wave_img" url="/wave.med.gif" />
        <h:input_number id="userNo" numberStyle="NUMBER"
            valueRef="UserNumberBean.userNumber">
            <f:validate_longrange minimum="0" maximum="10" />
        </h:input_number>
        <h:command_button id="submit" action="success"
            label="Submit" commandName="submit" /><p>
        <h:output_errors id="errors1" for="userNo"/>
    </h:form>
</f:use_faces>
```

Esta página demuestra unas cuantas características importantes que utilizaremos en la mayoría de nuestras aplicaciones JavaServer Faces:

- La etiqueta form:

  Esta etiqueta representa un formulario de entrada, que permite al usuario introducir algún dato y enviarlo al servidor, normalmente pulsando un botón. Las etiquetas que representan los componentes que conforman el formulario se anidan dentro de la etiqueta form. Estas etiquetas son h:input_number y h:command_button.

- La etiqueta input_number:

  Esta etiqueta representa un componente que es un campo de texto, dentro del que el usuario introduce un número. Esta etiqueta tiene tres atributos: id, valueRef, y numberStyle. El atributo id es

opcional y corresponde al identificador ID del componente. Si no incluimos uno, la implementación JavaServer Faces generará uno automáticamente.

El atributo valueRef utiliza una expresión de referencia para referirse a la propiedad del objeto del modelo que contiene los datos introducidos en el campo de texto. La parte de la expresión que hay antes del "." debe corresponder con el nombre definido por el elemento managed-bean-name del bean controlado en el fichero de configuración. La parte de la expresión que hay después del "." debe corresponder con el nombre del elemento property-name correspondiente en la declaración del propio bean controlado. En este ejemplo, no se declararon elementos property-name porque no se inicializaron propiedades en la arrancada de la aplicación para este ejemplo.
El atributo numberStyle indica el nombre del patrón de estilo de número definido según la clase java.text.NumberFormat. Los valores válidos son: currency, integer, number, o percent.

- La etiqueta validate_longrange:

  La etiqueta input_number también contiene una etiqueta validate_longrange, que es una del conjunto de etiquetas validadores estándar incluido con la implementación de referencia de JavaServer Faces. Este validador chequea si el valor local de un componente está dentro de un cierto rango. El valor debe ser cualquier cosa que se pueda convertir a long. Esta etiqueta tiene dos atributos, uno que especifica un valor mínimo y otro que específica un valor máximo. Aquí, la etiqueta se utiliza para asegurarnos de que el número introducido en el campo de texto es un número entre el 0 y el 10.

- La etiqueta command_button:

  Esta etiqueta representa el botón utilizado para enviar los datos introducidos en el campo de texto. El atributo action específica una salida que facilita al mecanismo de navegación la decisión de qué página abrir luego.

- La etiqueta output_errors:

  Esta etiqueta mostrará un mensaje de error si el dato introducido en el campo de texto no cumple con las reglas especificadas por el validador. El mensaje de error se muestra en el lugar de la página donde hayamos situado la etiqueta output_errors.

7.3.5 Definir las Navegación por las Páginas

Otra posibilidad que tiene el desarrollador de la aplicación es definir la navegación de páginas por la aplicación.
El desarrollador de la aplicación define la navegación por la aplicación mediante el fichero de configuración, el mismo fichero en el que se declararon los beans manejados. Aquí están las reglas de navegación definidas para el ejemplo guessNumber:

```
<navigation-rule>
    <from-tree-id>/greeting.jsp</from-tree-id>
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-tree-id>/response.jsp</to-tree-id>
    </navigation-case>
</navigation-rule>
<navigation-rule>
    <from-tree-id>/response.jsp</from-tree-id>
        <navigation-case>
        <from-outcome>success</from-outcome>
        <to-tree-id>/greeting.jsp</to-tree-id>
    </navigation-case>
</navigation-rule>
```

Cada regla de navegación define cómo ir de una página (especificada en el elemento from-tree-id) a otras páginas de la aplicación. El elemento navigation-rule puede contener cualquier número de elemento navigation-case, cada uno de los cuales define la página que se abrirá luego (definida por to-tree-id) basándose en una salida lógica (definida mediante from-outcome).
La salida se puede definir mediante el atributo action del componente UICommand que envía el formulario, como en el ejemplo guessNumber:

```
<h:command_button id="submit" action="success" label="Submit" />
```

La salida también puede venir del valor de retorno del método invoke de un objeto Action. Este método realiza algún procesamiento para determinar la salida. Un ejemplo es que el método invoke puede chequear si la password que el usuario ha introducido en la página corresponde con la del fichero. Si es así, el método invoke podría devolver "success"; si no es así, podría devolver "failure". Un salida de "failure" podría resultar en la recarga de la página de logon. Una salida de "success" podría resultar en que se mostrara una página con las actividades de la tarjeta de crédito del usuario, por ejemplo.

7.4 El Ciclo de Vida de una Página JavaServer Faces

El ciclo de vida de una página JavaServer Faces page es similar al de una página JSP: El cliente hace una petición HTTP de la página y el servidor responde con la página traducida a HTML. Sin embargo, debido a las características extras que ofrece la tecnología JavaServer Faces, el ciclo de vida proporciona algunos servicios adicionales mediante la ejecución de algunos pasos extras.
Los pasos del ciclo de vida se ejecutan dependen de si la petición se originó o no desde una aplicación JavaServer Faces y si la respuesta es o no generada con la fase de renderizado del ciclo de vida de JavaServer Faces. Veremos los diferentes escenarios del ciclo de vida y se explicara cada uno de estas fases del ciclo de vida utilizando el ejemplo guessNumber.

7.4.1 Escenarios de Procesamiento del Ciclo de Vida de una Petición

Una aplicación JavaServer Faces soporta dos tipos diferentes de respuestas y dos tipos diferentes de peticiones:

- Respuesta **Faces**:

  Una respuesta servlet que se generó mediante la ejecución de la fase Renderizar la Respuesta del ciclo de vida de procesamiento de la respuesta.

- Respuesta **No-Faces**:

  Una respuesta servlet que no se generó mediante la ejecución de la fase Renderizar la Respuesta. Un ejemplo es una página JSP que no incorpora componentes JavaServer Faces.

- Petición **Faces**:

  Una petición servlet que fue enviada desde una Respuesta Faces préviamente generada. Un ejemplo es un formulario enviado desde un componente de interface de usuario JavaServer Faces, donde la URI de la petición identifica el árbol de componentes JavaServer Faces para usar el procesamiento de petición.

- Petición **No-Faces**:

  Una petición Servlet que fue enviada a un componente de aplicación como un servlet o una página JSP, en vez de directamente a un componente JavaServer Faces.

La combinación de estas peticiones y respuestas resulta en tres posibles escenarios del ciclo de vida que pueden existir en una aplicación JavaServer Faces:

- Escenario 1: Una Petición No-Faces genera una Respuesta Faces:
  Un ejemplo de este escenario es cuando se pulsa un enlace de una página HTML que abre una página que contiene componentes JavaServer Faces. Para dibujar una Respuesta Faces desde una petición No-Faces, una aplicación debe proporcionar un mapeo FacesServlet en la URL de la página que contiene componentes JavaServer Faces. FacesServlet accepta peticiones entrantes y pasa a la implementación del ciclo de vida para su procesamiento.
- Escenario 2: Una Petición Faces genera una Respuesta No-Faces:
  Algunas veces una aplicación JavaServer Faces podría necesitar redirigir la salida a un recurso diferente de la aplicación Web diferente o generar una respuesta que no contiene componentes JavaServer Faces. En estas situaciones, el desarrollador debe saltarse la fase de renderizado (Renderizar la Respuesta) llamando a FacesContext.responseComplete. FacesContext contiene toda la información asociada con una Petición Faces particular. Este método se puede invocar durante las fases Aplicar los Valores de Respuesta, Procesar Validaciones o Actualizar los Valores del Modelo.
- Escenario 3: Una Petición Faces genera una Respuesta Faces:
  Es el escenario más común en el ciclo de vida de una aplicación JavaServer Faces. Este escenario implica componentes JavaServer Faces enviando una petición a una aplicación JavaServer Faces utilizando el FacesServlet. Como la petición ha sido manejada por la implementación JavaServer Faces, la aplicación no necesita pasos adicionales para generar la respuesta. Todos los oyentes, validadores y conversores serán invocados automáticamente durante la fase apropiada del ciclo de vida estándar, como se describe en la siguiente sección.

7.4.2 Ciclo de Vida Estándar de Procesamiento de Peticiones

La mayoría de los usuarios de la tecnología JavaServer Faces no necesitarán conocer a fondo el ciclo de vida de procesamiento de una petición. Sin embargo, conociendo lo que la tecnología JavaServer Faces realiza para procesar una página, un desarrollador de aplicaciones JavaServer Faces no necesitará preocuparse de los problemas de renderizado asociados con otras tecnologías IU. Un ejemplo sería el cambio de estado de los componentes individuales. Si la selección de un componente como un checkbox afecta a la apariencia de otro componente de la página, la tecnología JavaServer Faces manejará este evento de la forma apropiada y no permitirá que se dibuje la página sin reflejar este cambio.
La siguiente figura ilustra los pasos del ciclo de vida petición-respuesta JavaServer Faces

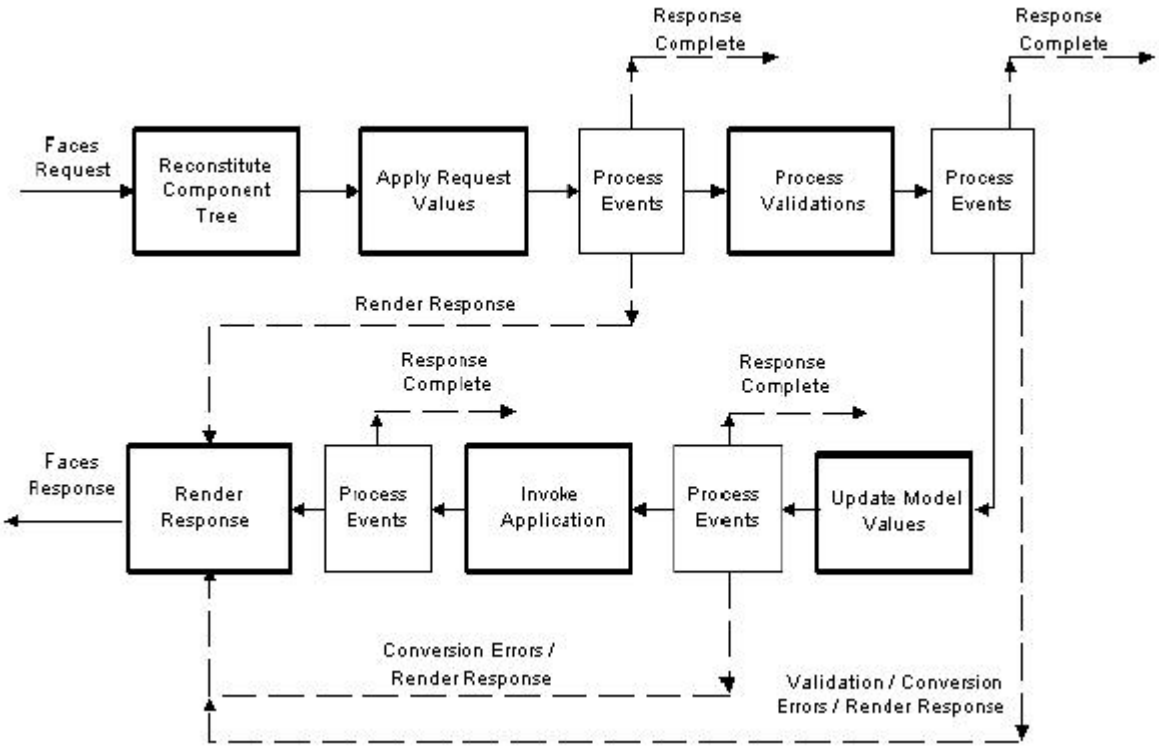


Figura 7-2. Ciclo de vida de una Petición-Respuesta Estándar en Java Server Faces.

7.4.3 Reconstituir el Árbol de Componentes

Cuando se hace una petición para una página JavaServer Faces, como cuando se pulsa sobre un enlace o un botón, la implementación JavaServer Faces comienza el estado Reconstituir el Árbol de Componentes. Durante esta fase, la implementación JavaServer Faces construye el árbol de componentes de la página JavaServer Faces, conecta los manejadores de eventos y los validadores y graba el estado en el FacesContext. El árbol de componentes de la página greeting.jsp del ejemplo guessNumber se parecería a esto:

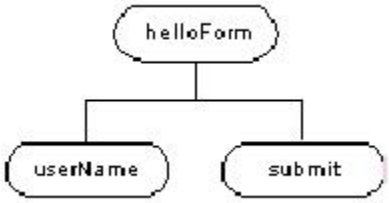


Figura 7-3. Esquema de acciones en una página con Java Server Faces.

7.4.4 Aplicar Valores de la Petición

Una vez construido el árbol de componentes, cada componente del árbol extrae su nuevo valor desde los parámetros de la petición con su método *decode*. Entonces el valor es almacenado localmente en el componente. Si falla la conversión del valor, se genera un mensaje de error asociado con el componente y se pone en la cola de *FacesContext*. Este mensaje se mostrará durante la fase Renderizar la Respuesta, junto con cualquier error de validación resultante de la fase Procesar Validaciones.
Si durante esta fase se produce algún evento, la implementación JavaServer Faces emite los eventos a los oyentes interesados.

En este punto, si la aplicación necesita redirigirse a un recurso de aplicación Web diferente o generar una respuesta que no contenga componentes JavaServer Faces, puede llamar a *FacesContext.responseComplete*.
En el caso del componente *userNumber* de la página greeting.jsp, el valor es cualquier cosa que el usuario introduzca en el campo. Como la propiedad del objeto de modelo unida al componente tiene un tipo Integer, la implementación JavaServer Faces convierte el valor de un String a un Integer.
En este momento, se han puesto los nuevos valores en los componentes y los mensajes y eventos se han puesto en sus colas.

7.4.5 Procesar Validaciones

Durante esta fase, la implementación JavaServer Faces procesa todas las validaciones registradas con los componentes del árbol. Examina los atributos del componente que especifican las reglas de validación y compara esas reglas con el valor local almacenado en el componente. Si el valor local no es válido, la implementación JavaServer Faces añade un mensaje de error al FacesContext y el ciclo de vida avanza directamente hasta la fase Renderizar las Respuesta para que la página sea dibujada de nuevo incluyendo los mensajes de error. Si había errores de conversión de la fase Aplicar los Valores a la Petición, también se mostrarán.
En este momento, si la aplicación necesita redirigirse a un recurso de aplicación Web diferente o generar una respuesta que no contenga componentes JavaServer Faces, puede llamar a *FacesContext.responseComplete*.
Si se han disparado eventos durante esta fase, la implementación JavaServer Faces los emite a los oyentes interesados.
En la página greeting.jsp, la implementación JavaServer Faces procesa el validador sobre la etiqueta input_number de UserNumber. Verifica que el dato introducido por el usuario en el campo de texto es un entero entre 0 y 10. Si el dato no es válido, o ocurrió un error de conversión durante la fase Aplicar los Valores a la Petición, el procesamiento salta a la fase Renderizar las Respuesta, durante la que se dibujará de nuevo la página greeting.jsp mostrando los mensajes de error de conversión o validación en el componente asociado con la etiqueta output_errors.

7.4.6 Actualizar los Valores del Modelo

Una vez que la implementación JavaServer Faces determina que el dato es válido, puede pasar por el árbol de componentes y configurar los valores del objeto de modelo correspondiente con los valores locales de los componentes. Sólo se actualizarán los componentes que tengan expresiones *valueRef*. Si el dato local no se puede convertir a los tipos especificados por las propiedades del objeto del modelo, el ciclo de vida avanza directamente a la fase Renderizar las Respuesta, durante la que se dibujará de nuevo la página mostrando los errores, similar a lo que sucede con los errores de validación.
En este punto, si la aplicación necesita redirigirse a un recurso de aplicación Web diferente o generar una respuesta que no contenga componentes JavaServer Faces, puede llamar a *FacesContext.responseComplete*.
Si se han disparado eventos durante esta fase, la implementación JavaServer Faces los emite a los oyentes interesados.
En esta fase, a la propiedad userNumber del UserNumberBean se le da el valor del componente userNumber.

7.4.7 Invocar Aplicación

Durante esta fase, la implementación JavaServer Faces maneja cualquier evento a nivel de aplicación, como enviar un formulario o enlazar a otra página.
En este momento, si la aplicación necesita redirigirse a un recurso de aplicación Web diferente o generar una respuesta que no contenga componentes JavaServer Faces, puede llamar a *FacesContext.responseComplete.*
La página greeting.jsp del ejemplo guessNumber tiene asociado un evento a nivel de aplicación con el componente Command. Cuando se procesa este evento, una implementación de *ActionListener* por defecto recupera la salida, “success”, desde el atributo action del componente. El oyente pasa la salida al

NavigationHandler por defecto. Y éste contrasta la salida con las reglas de navegación definidas en el fichero de configuración de la aplicación para determinar qué página se debe mostrar luego.
Luego la implementación JavaServer Faces configura el árbol de componentes de la respuesta a esa nueva página. Finalmente, la implementación JavaServer Faces transfiere el control a la fase Renderizar la Respuesta.

7.4.8 Renderizar la Respuesta

Durante esta fase, la implementación JavaServer Faces invoca las propiedades de codificación de los componentes y dibuja los componentes del árbol de componentes grabado en el *FacesContext*.
Si se encontraron errores durante las fases Aplicar los Valores a la Petición, Procesar Validaciones o Actualizar los Valores del Modelo, se dibujará la página original. Si las páginas contienen etiquetas output_errors, cualquier mensaje de error que haya en la cola se mostrará en la página.
Se pueden añadir nuevos componentes en el árbol si la aplicación incluye renderizadores personalizados, que definen cómo renderizar un componente. Después de que se haya renderizado el contenido del árbol, éste se graba para que las siguientes peticiones puedan acceder a él y esté disponible para la fase Reconstituir el Árbol de Componentes de las siguientes llamadas.

7.5 Modelo de Componentes de Interface de Usuario

Los componentes de IU JavaServer Faces son elementos configurables y reutilizables que componen el interface de usuario de las aplicaciones JavaServer Faces. Un componente puede ser simple, como un botón, o compuesto, como una tabla, que puede estar compuesta por varios componentes.
La tecnología JavaServer Faces proporciona una arquitectura de componentes rica y flexible que incluye:

- Un conjunto de clases UIComponent para especificar el estado y comportamiento de componentes de IU.
- Un modelo de renderizado que define cómo renderizar los componentes de diferentes formas.
- Un modelo de eventos y oyentes que define cómo manejar los eventos de los componentes.
- Un modelo de conversión que define cómo conectar conversores de datos a un componente.
- Un modelo de validación que define cómo registrar validadores con un componente.

7.5.1 Las Clases de los Componentes del Interface de Usuario

La tecnología JavaServer Faces proporciona un conjunto de clases de componentes de IU, que especifican toda la funcionalidad del componente, cómo mantener su estado, mantener una referencia a objetos del modelo, y dirigir el manejo de eventos y el renderizado para un conjunto de componentes estándar.
Estas clases son completamente extensibles, lo que significa que podemos extenderlas para crear nuestros propios componentes personalizados.
Todas las clases de componentes IU de JavaServer Faces descienden de la clase UIComponentBase, que define el estado y el comportamiento por defecto de un UIComponent. El conjunto de clases de componentes UI incluido en la última versión de JavaServer Faces es:
-UICommand: Representa un control que dispara actions cuando se activa.
-UIForm: Encapsula un grupo de controles que envían datos de la aplicación. Este componente es análogo a la etiqueta form de HTML.
-UIGraphic: Muestra una imagen.
-UIInput: Toma datos de entrada del usuario. Esta clase es una subclase de UIOutput.
-UIOutput: Muestra la salida de datos en un página.
-UIPanel: Muestra una tabla.
-UIParameter: Representa la sustitución de parámetros.
-UISelectItem: Representa un sólo ítem de un conjunto de ítems.
-UISelectItems: Representa un conjunto completo de ítems.
-UISelectBoolean: Permite a un usuario seleccionar un valor booleano en un control, selececcionándolo o deseleccionándolo. Esta clase es una subclase de UIInput.
-UISelectMany: Permite al usuario seleccionar varios ítems de un grupo de ítems. Esta clase es una subclase de UIInput.

-UISelectOne: Permite al usuario seleccionar un ítem de un grupo de ítems. Esta clase es una subclase de UIInput.

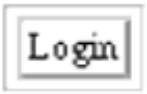
La mayoría de los autores de páginas y de los desarrolladores de aplicaciones no tendrán que utilizar estas clases directamente. En su lugar, incluirán los componentes en una página usando la etiqueta correspondiente al componente. La mayoría de estos componentes se pueden renderizar de formas diferentes. Por ejemplo, un UICommand se puede renderizar como un botón o como un hiper-enlace.

7.5.2 El Modelo de Renderizado de Componentes

La arquitectura de componentes JavaServer Faces está diseñada para que la funcionalidad de los componentes se defina mediante las clases de componentes, mientras que el renderizado de los componentes se puede definir mediante un renderizador separado. Este diseño tiene varios beneficios:

- Los escritores de componentes pueden definir sólo una vez el comportamiento de un componente, pero pueden crear varios renderizadores, cada uno de los cuales define una forma diferente de dibujar el componente para el mismo cliente o para diferentes clientes.
- Los autores de páginas y los desarrolladores de aplicaciones pueden modificar la apariencia de un componente de la página seleccionando la etiqueta que representa la combinación componente/renderizador apropiada.

Un kit renderizador define como se mapean las clases de los componentes a las etiquetas de componentes apropiadas para un cliente particular. La implementación JavaServer Faces incluye un RenderKit estándar para renderizar a un cliente HTML.
Por cada componente IU que soporte un RenderKit, éste define un conjunto de objetos *Renderer*. Cada objeto *Renderer* define una forma diferente de dibujar el componente particular en la salida definida por el RenderKit. Por ejemplo, un componente *UISelectOne* tiene tres renderizadores diferentes. Uno de ellos dibuja el componente como un conjunto de botones de *radio*. Otro dibuja el componente como un *ComboBox*. Y el tercero dibuja el componente como un *ListBox*.
Cada etiqueta JSP personalizada en el RenderKit de HTML, está compuesta por la funcionalidad del componetne, definida en la clase UIComponent, y los atributos de renderizado, definidos por el *Renderer*. Por ejemplo, las dos etiquetas que podemos ver en la siguiente tabla representan un componente UICommand, renderizado de dos formas diferentes:

Etiqueta	Se renderiza como...
command_button	
command_hyperlink	

La parte command de las etiquetas corresponde con la clase UICommand, y especifica la funcionalidad, que es disparar un action. Las partes del botón y el hiperenlace de las etiquetas corresponden a un renderizador independiente, que define cómo dibujar el componente.
La implementación de referencia de JavaServer Faces proporciona una librería de etiquetas personalizadas para renderizar componentes en HTML. Soporta todos los componentes listados en la siguiente tabla:

Etiqueta	Funciones	Se renderiza como...	Apariencia
command_button	Enviar un formulario a la aplicación	Un elemento <input type=type> HTML, donde el valor del tipo puede ser submit, reset, o image.	Un botón
command_hyperlink	Enlaza a otra página o localización en otra págona	Un elemento <a href> HTML.	Un Hiperenlace
form	Representa un formulario de entrada. Las etiquetas internas del formulario reciben los datos que serán enviados con el formulario.	Un elemento <form> HTML.	No tiene apariencia.

graphic_image	Muestra una imagen	Un elemento <img> HTML.	Una imagen.
input_date	Permite al usuario introducir una fecha	Un elemento <input type=text> HTML.	Un string de texto, formateado con un ejemplar de java.text.DateFormat
input_datetime	Permite al usuario introducir una fecha y una hora.	Un elemento <input type=text> HTML.	Un string de texto, formateado con un ejemplar de java.text.SimpleDateFormat
input_hidden	Permite introducir una variable oculta en una página.	Un elemento <input type=hidden> HTML.	Sin apariencia
input_number	Permite al usuario introducir un número.	Un elemento <input type=text> HTML.	Un string de texto, formateado con un ejemplar de java.text.NumberFormat
input_secret	Permite al usuario introducir un string sin que aparezca el string real en el campo.	Un elemento <input type=password> HTML.	Un campo de texto, que muestra una fila de caracteres en vez del texto real introducido.
input_text	Permite al usuario introducir un string.	Un elemento <input type=text> HTML.	Un campo de texto.
input_textarea	Permite al usuario introducir un texto multi-líneas.	Un elemento <textarea> HTML.	Un campo de texto multi-línea.
input_time	Permite al usuario introducir una hora.	Un elemento <input type=text> HTML.	Un string de texto, formateado con un ejemplar de java.text.DateFormat
output_date	Muestra una fecha formateada.	Texto normal.	Un string de texto, formateado con un ejemplar de java.text.DateFormat
output_datetime	Muestra una fecha y hora formateados.	Texto normal.	Un string de texto, formateado con un ejemplar de java.text.SimpleDateFormat
output_errors	Muestra mensajes de error.	Texto normal	Texto normal
output_label	Muestra un componente anidado como una etiqueta para un campo de texto especificado.	Un elemento <label> HTML.	Texto normal.
output_message	Muestra un mensaje localizado (internacionalizado).	Texto normal.	Texto normal.
output_number	Muestra un número formateado.	Texto normal.	Un string de texto, formateado con un ejemplar de java.text.NumberFormat
output_text	Muestra una línea de texto.	Texto normal.	Texto normal.
output_time	Muestra una hora formateada.	Texto normal.	Un string de texto, formateado con un ejemplar de java.text.DateFormat
panel_data	Itera sobre una colección de datos.		Un conjunto de filas en una tabla
panel_grid	Muestra una tabla.	Un elemento <label> HTML. con elementos <tr> y %lt;td>	Una tabla.
panel_group	Agrupa un conjunto de paneles bajo un padre.		Una fila en una tabla
panel_list	Muestra una tabla de datos que vienen de una collection, un array, un iterator, o un map.	Un elemento <table> HTML. con elementos <tr> y %lt;td>	Una tabla.
selectboolean_checkbox	Permite al usuario cambiar el valor de una elección booleana.	Un elemento <input type=checkbox> HTML.	Un checkBox.
selectitem	Representa un ítem de una lista de ítems en un componente UISelectOne.	Un elemento <option> HTML.	Sin apariencia.
selectitems	Representa una lista de ítems en un componente UISelectOne.	Un elemento <option> HTML.	Sin apariencia.

selectmany_checkboxlist	Muestra un conjunto de checkbox, en los que el usuario puede seleccionar varios.	Un conjunto de elementos <input> HTML.	Un conjunto de CheckBox.
selectmany_listbox	Permite a un usuario seleccionar varios ítems de un conjunto de ítems, todos mostrados a la vez.	Un conjunto de elementos <select> HTML.	Un ListBox
selectmany_menu	Permite al usuario selecionar varios ítems de un grupo de ítems.	Un conjunto de elementos <select> HTML.	Un comboBox.
selectone_listbox	Permite al usuario seleccionar un ítem de un grupo de ítems.	Un conjunto de elementos <select> HTML.	Un listBox.
selectone_menu	Permite al usuario seleccionar un ítem de un grupo de ítems.	Un conjunto de elementos <select> HTML.	Un comboBox.
selectone_radio	Permite al usuario seleccionar un ítem de un grupo de ítems.	Un conjunto de elementos <input type=radio> HTML.	Un conjunto de botones de radio.

7.5.3 Modelo de Conversión

Una aplicación JavaServer Faces opcionalmente puede asociar un componente con datos del objeto del modelo del lado del servidor. Este objeto del modelo es un componente JavaBeans que encapsula los datos de un conjunto de componentes. Una aplicación obtiene y configura los datos del objeto modelo para un componente llamando a las propiedades apropiadas del objeto modelo para ese componente.
Cuando un componente se une a un objeto modelo, la aplicación tiene dos vistas de los datos del componente: la vista modelo y la vista presentación, que representa los datos de un forma que el usuario pueda verlos y modificarlos.
Una aplicación JavaServer Faces debe asegurarse que los datos del componente puedan ser convertidos entre la vista del modelo y la vista de presentación. Esta conversión normalmente la realiza automáticamente el renderizador del componente.
En algunas situaciones, podríamos querer convertir un dato de un componente a un tipo no soportado por el renderizador del componente. Para facilitar esto, la tecnología JavaServer Faces incluye un conjunto de implementaciones estándar de Converter que nos permite crear nuestros conversores personalizados. La implementación de Converter convierte los datos del componente entre las dos vistas.

7.5.4 Modelo de Eventos y Oyentes

Un objetivo de la especificación JavaServer Faces es mejorar los modelos y paradigmas existentes para que los desarrolladores se puedan familiarizar rápidamente con el uso de JavaServer Faces en sus aplicaciones. En este espíritu, el modelo de eventos y oyentes de JavaServer Faces mejora el diseño del modelo de eventos de JavaBeans, que es familiar para los desarrolladores de GUI y de aplicaciones Web.
Al igual que la arquitectura de componentes JavaBeans, la tecnología JavaServer Faces define las clases Listener y Event que una aplicación puede utilizar para manejar eventos generados por componentes de IU. Un objeto Event identifica al componente que lo generó y almacena información sobre el propio evento. Para ser notificado de un evento, una aplicación debe proporcionar una implementación de la clase Listener y registrarla con el componente que genera el evento. Cuando el usuario activa un componente, como cuando pulsa un botón, se dispara un evento. Esto hace que la implementación de JavaServer Faces invoque al método oyente que procesa el evento. JavaServer Faces soporta dos tipos de eventos: eventos value-changed y eventos action.
Un evento value-changed ocurre cuando el usuario cambia el valor de un componente. Un ejemplo es seleccionar un checkbox, que resulta en que el valor del componente ha cambiado a true. Los tipos de componentes que generan estos eventos son los componentes UIInput, UISelectOne, UISelectMany, y UISelectBoolean. Este tipo de eventos sólo se dispara si no se detecta un error de validación.

Un evento action ocurre cuando el usuario pulsa un botón o un hiperenlace. El componente UICommand genera este evento.

7.5.5 Modelo de Validación

La tecnología JavaServer Faces soporta un mecanismo para validar el dato local de un componente durante la fase del Proceso de Validación, antes de actualizar los datos del objeto modelo.
Al igual que el modelo de conversión, el modelo de validación define un conjunto de clases estándar para realizar chequeos de validación comunes. La librería de etiquetas jsf-core también define un conjunto de etiquetas que corresponden con las implementaciones estándar de Validator.
La mayoría de las etiquetas tienen un conjunto de atributos para configurar las propiedades del validador, como los valores máximo y mínimo permitidos para el dato del componente. El autor de la página registra el validador con un componente anidando la etiqueta del validador dentro de la etiqueta del componente.
Al igual que el modelo de conversión, el modelo de validación nos permite crear nuestras propias implementaciones de Validator y la etiqueta correspondiente para realizar validaciones personalizadas.

7.6 Modelo de Navegación

Virtualmente todas las aplicaciones Web están hechas de un conjunto de páginas. Uno de los principales problemas de un desarrollador de aplicaciones Web es manejar la navegación entre esas páginas.
El nuevo modelo de navegación de JavaServer Faces facilita la definición de la navegación de páginas y el manejo de cualquier procesamiento adicional necesario para elegir la secuencia en se que cargan las páginas. En muchos casos, no se requiere código para definir la navegación. En su lugar, la navegación se puede definir completamente en el fichero de configuración de la aplicación usando un pequeño conjunto de elementos XML. La única situación en que necesitaremos proporcionar algo de código es si necesitamos algún procesamiento adicional para determinar qué página mostrar luego.
Para cargar la siguiente página en una aplicación web, el usuario normalmente pulsa un botón. Como vimos anteriormente, una pulsación de botón genera un evento action. La implementación de JavaServer Faces proporciona un nuevo oyente de eventos action por defecto para manejar este evento. Este oyente determina la salida del evento action, como success o failure. Esta salida se puede definir como una propiedad String del componente que generó el evento o como el resultado de un procesamientro extra realizado en un objeto Action asociado con el componente. Después de determinar la salida, el oyente la pasa al ejemplar de NavigationHandler asociado con la aplicación. Basándose en la salida devuelta, el NavigationHandler selecciona la página apropiada consultando el fichero de configuración de la aplicación.

7.7 Creación del Bean Controlado

Otra función crítica de las aplicaciones Web es el manejo apropiado de los recursos. Esto incluye la separación de la definición de objetos componentes IU de los objetos de datos y almacenar y manejar estos ejemplares de objetos en el ámbito apropiado. Las versiones anteriores de la tecnología JavaServer Faces nos permitían crear objetos del modelo que encapsulaban los datos y la lógica del negocio separadamente de los objetos de componentes IU y almacenarlos en un ámbito particular. La nueva versión especifica completamente cómo se crean y se manejan estos objetos.
Esta versión presenta APIs para:

- Evaluar una expresión que se refiere a un objeto del modelo, una propiedad de un objeto del modelo, u otro tipo de datos primitivo o estructura de datos. Esto se hace con el API Value-Binding.
- Recuperar un objeto desde el ámbito. Esto se hace con el API VariableResolver.
- Crear un objeto y almacenarlo en un ámbito si no está ya allí. Esto se hace con el VariableResolver por defecto, llamada la Facilidad Bean Controlado, que se configura con el fichero de configuración de la aplicación descrito en la siguiente página.

7.8 Configuración de la Aplicación

En las páginas anteriores hemos visto los distintos recursos disponibles para una aplicación JavaServer Faces. Esto incluye, conversores, validadores, componentes, objetos del modelo, acciones, manejadores de navegación y otros. En las versiones anteriores, estos recursos tenían que ser configurados mediante

programación. Se requería un ApplicationHandler para definir la navegación de páginas, y un ServletContextListener para registrar conversores, validadores renderizadores, etc.
Esta versión presenta un formato de configuración de recursos configurable (como un documento XML) para configurar los recursos requeridos por una aplicación en el momento de arrancar la aplicación. Esta nueva característica elimina la necesidad de un ApplicatonHandler y un ServletContextListener.
En las versiones anteriores, una vez que se creaban los recursos, la información utilizada por algunos de ellos se almacenaba en el FacesContext, que representa información contextual para una petición dada. Estos recursos normalmente están disponibles durante el ciclo de vida de la aplicación. Por lo tanto, es más apropiado recuperar la información de estos recursos desde un sólo objeto que es ejemplarizado para cada aplicación. Esta versión de JavaServer Faces presenta la clase Application, que se crea automáticamente por cada aplicación.
La clase Application actúa como factoría centralizada de recursos como conversores y recursos de mensajes que están definidos en el fichero XML. Cuando una aplicación necesita acceder a alguna información sobre uno de los recursos definidos en el fichero XML, primero recupera un ejemplar de Application de un ApplicationFactory y luego recupera el ejemplar del recurso de la aplicación.

Capítulo 8.

8. Spring Web MVC Framework

8.1 Introduccion

El framework MCV Web de Spring está construido en base a DispatcherServlet que procesa las peticiones hacia los manejadores, con mapeos configurables hacia los manejadores, vista de resultados, y utilidades como subida y descarga de ficheros. Por defecto el manejador es un interfaz simple denominado Controler, que ofrece un método de Modelo y vista denominado handleRequest (request, response). Este método puede ser usado como controlador de las aplicaciones pero es preferible incluir una implementación jerárquica, consiste por ejemplo en AbstractController, AbstractCommandController y SimpleFormController. El controlador de la aplicación será típicamente una subclase de estas. De esta manera podemos escoger cual va a ser nuestra clase base. Si no necesitamos un form, no tenemos necesidad de un controlador para ese form. Está es una de las mayores diferencias con Struts.

En Spring Web MCV se puede usar cualquier objeto como comando o como objeto de formulario, no es necesario implementar un interface especifico en el framework o una clase base. La unión de datos en Spring es altamente flexible: por ejemplo, los errores de violaciones de tipos de datos pueden ser evaluados por la aplicación, y no elaborando un sistema de errores. De esta manera no necesitamos duplicar las propiedades de nuestros objetos de negocio, es más simple, tipos Strings inválidos en los objetos de formulario se elevaran a manejadores locales, o convirtiendo al tipo String correcto. Esta es otra de las mayores diferencias con Strus el cual está construido alrededor de clases bases como Action y ActionForm.

Comparado con WebWork, Spring tiene mas diferenciado el concepto de los roles para los objetos. Soporta la noción de un Controller, un comando opcional u objeto de formulario, y un modelo que consigue pasar hacia la parte de la vista. La parte de modelo normalmente incluye el comando o objeto de formulario pero con referencia arbitraria al dato; por el contrario, un Action de WebWork combina todos los roles dentro de un único objeto. WebWork obliga a usar extensos objetos de negocio como parte del formulario, pero solo los trata en las propiedades del bean que están en la clase Action. Finalmente, la misma instancia Action que maneja la petición es usada para evaluar los objetos del formulario en la vista. De este modo, las referencias a los datos necesitan ser modeladas como propiedades del bean del Action. Esto provoca demasiados roles para un único objeto.

La parte de vista de String es extremadamente flexible. Una implementación del Controller puede ser escrita directamente como una respuesta hacia la parte de vista (retornando null para el ModelAndView). En un caso normal, una instancia de ModelAndView consiste de una nombre de vista y un modelo denominado Map, el cual contiene nombres de bean y sus correspondientes objetos (como un comando o formulario, contiene datos referenciados). La parte de resolución de vista de nombre es altamente configurable, cualquier vía nombres de bean, ficheros de propiedades, o nuestro propia implementación de ViewResolver. El hecho que el modelo (la M in MVC) este basado en un inteface de Map ayuda a completar la abstracción de la tecnología en que se sustenta la parte de vista. Cualquier redirección puede ser integrada directamente, tanto si es JSP, Velocity, o cualquier otra tecnología de redirección. El modelo basado en Map se transforma fácilmente al formato apropiado, en atributos de respuesta para JSP o en un modelo de plantilla de Velocity.

8.1.1Capacidades de Spring Web MVC

El modulo Web de Spring proporciona un a riqueza de elementos web incluyendo:

- Clara separación de roles – controlador, validador, objetos de comando, objetos de formulario, objetos de modelado, DispatcherServlet, mapeos hacia los manejadores, vista de resultados, etc. Cada rol puede ser implementado por un objeto especializado.
- Fuerte y sencilla configuración del propio framework y las clases de aplicación como JavaBeans, incluyendo un fácil referenciado hacia los contextos, así como desde los controlodares web a objetos de negocio y validadores.
- Adaptabilidad, y no infiltraciones. Se puede usar cualquier subclase controladora que necesitemos en lugar de que tenga que proceder todo de un único controlador.

- Capacidad de reutilizar el código de negocio, no es necesario duplicarlo. Se pueden usar objetos de negocio existentes como comando o como objetos de formulario en lugar de duplicarlos para lograr extender una clase base particular clase del framework.
- Uniones y validaciones mejoradas – los errores de tipo de aplicación – nivel de validación de errores que guardan el valor ofrecido, fechas locales y formateadores de números etc, en lugar de esto tendremos solo objetos de formulario String con parseo y conversiones manuales hacia objetos de negocio.
- Óptimos mapeos hacia los manejadores y hacia la vista de resolución – mapeos a manejadores y vista de resolución desde estrategias encadenadas hacia una simple URL como base de la configuración, a sofisticados, construidos por objetivo estrategias de resolución . Esto es más flexible que otros framework MCV los cuales obligan a utilizar una técnica en particular.
- Modelo de transferencias flexible- modelos de transferencias del Map vía nombre/valor, soportando una fácil integración con cualquier tecnología para la Vista.
- Optimo integración para desarrollos en el lado del cliente, soporta JSP, con o sin tag de librerías de Spring, soporta JSTL, soporta Velocity sin la necesidad de conversores extras etc.
- Un poderoso tag de librerías para de JSP que hace más sencillo el desarrollo de páginas JSP.
- Beans cuyo ciclo de vida tienen un ámbito en la petición HTTP actual o HTTP Session. No es específico del modelo Spring MVC, pero es lo que el contendor WebApplicationContext que usa el MVC de Spring utiliza.

8.2 El DispatcherServlet

El framework web MVC de Spring, es como otros framework web MVC (conductores de peticiones) esta diseñado alrededor de un servlet central que maneja las peticiones hacia los controladores y ofrece otras funcionalidades para el desarrollo de aplicaciones Web. El DispatcherServlet de Spring sin embargo, hace más que eso. Está completamente integrado con el contenedor IoC de Spring y de esta manera obtiene todas las capacidades que ofrece la tecnología Spring.

El workflow de procesamiento que sigue una petición en el DispatcherServlet de MVC Web de Spring se muestra en el siguiente diagrama. El DispactherServlet es una implementación del diseño ofrecido en el patrón “Front Controller”.

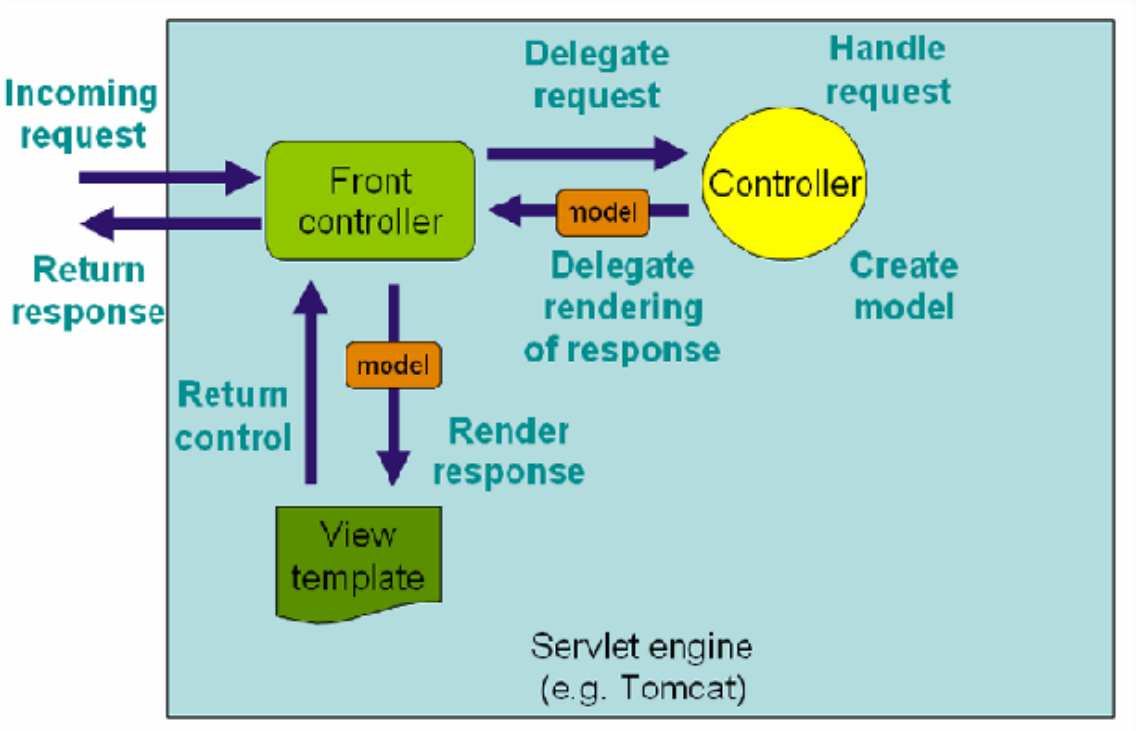


Figura 8-1. Flujo de trabajo (a alto nivel), en el proceso de una petición en el Spring Web MVC

El DispatcherServlet es el actual Servlet (esta basado en la clase base HttpServlet), y como tal esta declarado en el web.xml de la aplicación web. Las peticiones que queramos que maneje DispatcherServlet tienen que ser mapeadas usando la URL que está en el mismo fichero web.xml. Sigue el estándar de la configuración de servlet de J2EE. Un ejemplo de la declaración y mapeo de un DispatcherServlet lo encontramos en el siguiente cuadro:

```
<web-app>
   <servlet>
      <servlet-name>example</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>example</servlet-name>
      <url-pattern>*.form</url-pattern>
   </servlet-mapping>
</web-app>
```

Se puede observar que todas las peticiones con .form serán manejadas por el 'example' DispatcherServlet. De esta manera los beans que use el framewok Web MVC de Spring ahora necesitan ser configurados.

Las instancias ApplicationContext en Spring pueden ser almacenadas. En el framework web MVC, cada DispatcherServlet tiene su propio WebApplicationContext, el cual tiene de forma inherente todos los bean definidos en el raíz de WebApplicationContext. Estos bean definidos que están inheridos pueden ser reconducidos y almacenados en el ámbito del servlet específico, y un nuevo ámbito local específico del bean puede ser definido a la instancia del servlet.

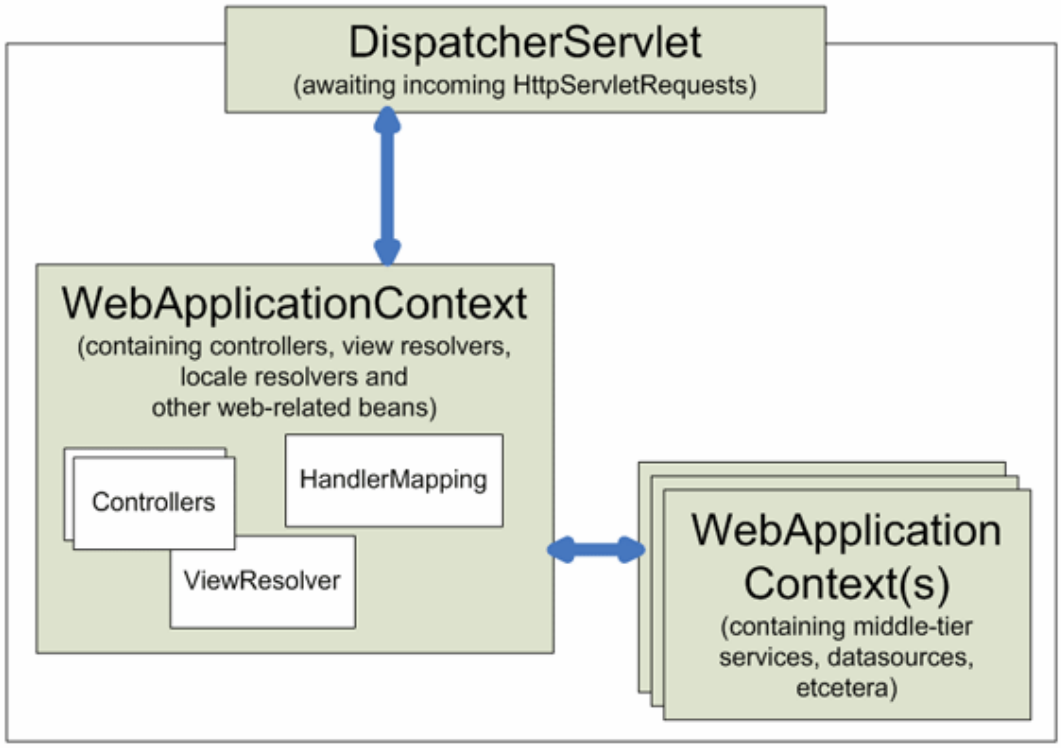


Figura 8-2. Jeraquía del Contexto en el Spring Web MVC

El framework, una vez inicializado desde un DispatcherServlet, mirara en el fichero con nombre [servlet-name]-servlet.xml en el directorio WEB-INF de nuestra aplicación web y creara los bean definidos allí.
Si observamos el siguiente DispatcherServlet servlet configurado (en el 'web.xml' file):

```
<web-app>
   ...
   <servlet>
      <servlet-name>golfing</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>golfing</servlet-name>
      <url-pattern>*.do</url-pattern>
   </servlet-mapping>
</web-app>
```

Con este servlet configurado, es necesario un fichero '/WEB-INF/golfing-servlet.xml' en la aplicación. Este fichero contendrá todos los componentes específicos (beans) de nuestro Spring Web MCV.

El WebApplicationContext es una extensión de ApplicationContext con algunas capacidades extras necesarias para las aplicaciones web. Difiere de un ApplicationContext normal en que es capaz de resolver temas, y que sabe que servlet esta asociado con él. El WebApplicationContext esta limitado en el ServletContext, y por usar métodos estáticos en la clase RequestContextUtils. Siempre se puede mirar el WebApplicationContext en caso que sea necesario acceder a él.

El Spring DispatcherServlet tiene una pareja de beans especiales que usa para ser capaz de procesar las peticiones y las redirecciones a la vista apropiada. Estos beans están incluidos en el Framework de Srping y pueden ser configurados en el WebApplicationContext, como cualquier otro bean. Cada uno de estos beans está descrito en la siguiente tabla.

Table. Beans especiales en el WebApplicationContext

Bean type	**Explanation**
Controladores /Controllers	Controladores: son todos los componentes que forman 'C' en el MVC.
Mapeos Manejadores/ Handler mappings	Mapeos de manejadores: manejar la ejecución de una lista pre y post procesos y controladroes que ejecutaran si encajan con ciertos criterios (por ejemplo machear una URL especifica con el controlador).
Resolutotes de Vista /View resolvers	Resolutotes de vista: son componentes capaces de resolver los nombres de vista hacia las propias vistas.
Resolutotes Locales /Locale resolver	Un resolutor loca es un componente capaz de resolver la configuración local que un cliente está usando, se usara para poder ofrecer vistas internacionalizadas.
Resolutotes de Temas	Un resolutor de tema es capaz de resolver los temas referentes al uso de nuestra aplicación web, por ejemplo ofrecer layouts

Bean type	Explanation
/Theme resolver	personalizados.
Resolutotes de ficheros multiparte /Multipart file resolver	Un resolutor de fichero en varias partes ofrece la funcionlaidad para procesar cargas de ficheros desde formulario HTML
Resolutotes de manejadores de Exepciones /Handler exception resolver(s)	Los manejadores de excepciones ofrecen funcionalidad para mapear excepciones hacia la parte de vista o para que implementemos nuestros propios manejadores de excepciones.

Después de presentar brevemente estos beans, continuamos con el DispatcherServlet. Cuando un DispatcherServlet es disparado para procesar una petición que va dirigido a este DispatcherServlet, empieza el proceso de la petición. La siguiente lista describe el proceso completo de una petición cuando es manejada por un DispatcherServlet:

- El WebApplicationContext es buscado para procesar la petición, dentro de un atributo. Esta cadena por defecto se encuentra bajo el valor siguiente DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUTE.
- El resolutor local se une a la petición para dejar los elementos en el proceso de resolución local para poder acceder a ellos cuando se está procesando la petición (redirreccionar la vista, preparar datos etc). Si no se usa el resolutor, no afecta a nada, con lo que si no se requiere resolución local, no necesita usarse.
- El tema resolutor asociado es unido a la petición para dejar preparados los elementos en la parte de vista determinada, conforme al tema que usen. Si no se usa el tema resolutor no afecta en nada, si no se necesita se puede ignorar.
- Si ha sido especificada una resolución en varias partes, la petición es preparada para varias partes , si las varias partes son encontradas, la petición es mapeada en una MultipartHttpServletRequest para que pueda se procesada por otros elementos en el proceso.
- El manejador apropiado es buscado para procesar la petición. Si el manejador es encontrado, la ejecución principal se asocia con el manejador (pre-procesos, post-procesos, y controladores) que será ejecutado para preparar un modelo (para redireccionar).
- Si un modelo es regresado, la vista es redireccionada. Si no se ha regresado un modelo (el pre o post proceso podría haber interceptado la petición, por ejemplo por razones de seguridad), no se redirige hacia la parte de vista, desde la petición podríamos haber tenido todo mal relleno.

Las excepciones que ocurran en el procesamiento de la petición serán resueltas por un manejador de excepción que es declarado en el WebApplicationContext.

El DispatcherServlet de Spring además es capaz de devolver la última fecha modificada. El proceso de determinar la última fecha de modificación por petición específica es sencillo: el DispatcherServlet primero mirara el manejador apropiado y testara si el manejador que ha encontrado implemente el interface LastModified. Si es así el valor de tiempo que devuelva el método getLastModified(request) del interface LastModified es regresado al cliente.

Se puede optimizar el DispatcherServlet de Spring añadiendo parámetros en el fichero web.xml o inicializando parámetros en el servlet.

8.3 Controladores

La noción de controlador es parte del diseño de patrones en el modelo MVC (más específicamente es la ‘C’ en MVC). El controlador proporciona acceso a la aplicación, esta definido como un servicio de interface. El controlador interpreta la entrada de usuario y transforma la entrada a un modelo el cual es representado para el usuario mediante una vista. Spring ha implementado la noción de controlador en un camino muy abstracto habilitando una variedad de diferentes tipos de controladores que pueden ser creados. Spring contiene controladores con una forma especifica, controladores de comando base, y controladores que ejecutan un estilo lógico novedoso por nombrar algunos.

El controlador básico de Spring's es el interface org.springframework.web.servlet.mvc.Controller :

```
public interface Controller {

    /**
     * Process the request and return a ModelAndView object which the DispatcherServlet
     * will render.
     */
    ModelAndView handleRequest(
        HttpServletRequest request,
        HttpServletResponse response) throws Exception;

}
```

El interface Controller define un único método que es responsable de manejar una petición y devolver un apropiado modelo y vista (‘M’ y ‘ V’ del MVC). Mientras el interface Controller es abstracto Spring ofrece muchas implementaciones del Controller que contienen mucha funcionalidad dependiendo de la necesidad. El interface Controller define la responsabilidad más básica requerida por un controlador: nombrar el manejador de la petición y regresar un modelo y una vista.

8.3.1 AbstractController y WebContentGenerator

Para proporcionar una infraestructura básica, todos los diferentes controladores de Spring extienden de AbstractController, una clase que ofrece elementos de cache, o por ejemplo, configuración de mimetype.

Cuando se usa el AbstractController como clase base de un controlador solo se puede sobrescribir el método handleRequestInternal(HttpServletRequest, HttpServletResponse), que implementara la lógica y regresara un objeto ModelAndView. Aquí se muestra un ejemplo:

```
package samples;

public class SampleController extends AbstractController {

    public ModelAndView handleRequestInternal(
        HttpServletRequest request,
        HttpServletResponse response) throws Exception {

        ModelAndView mav = new ModelAndView("hello");
        mav.addObject("message", "Hello World!");
        return mav;
    }
}
```

```
<bean id="sampleController" class="samples.SampleController">
    <property name="cacheSeconds" value="120"/>
</bean>
```

La clase de arriba y la declaración en el contexto de la aplicación es todo lo que se necesita para establecer un mapeo de controlador para conseguir un controlador simple para trabajar. Este controlador generara una directiva de cache que le dirá al cliente que tendrá 2 minutos antes de volver a chequearse si la página ha sido modificada para volverla a enviar. Este controlador además regresa una vista con un código de respuesta

definido en el propia clase es decir hard-codeado, hecho que es considerado una mala práctica en programación.

8.3.2 Otros controladores simples

Aunque se puede extender el AbstractController, Spring provee un número concreto de implementaciones los cuales ofrecen la funcionalidad común que se usa en aplicaciones simples de tipo MVC. El ParameterizableViewController es básicamente el mismo que en ejemplo anterior, excepto por el hecho que se puede especificar el nombre de vista que se regresara en el contexto de aplicación web (y de esta manera se puede omitir el hard-code del nombre de vista en la clase Java).

El UrlFilenameViewController revisa la URL y manda el nombre del fichero solicitado, usándolo como un nombre de vista. Por ejemplo, el nombre de fichero solicitado http://www.springframework.org/index.html en la petición es el nombre de vista index.

8.3.3 El MultiActionController

Spring ofrece un controlador multiaccion con el cual se puede agregar múltiples acciones en un solo controlador, así se realiza una agrupación de la funcionalidad. El controlador multiacción vive en un paquete separado - org.springframework.web.servlet.mvc.multiaction – y es capaz de mapear peticiones a nombres de método y después invocar correctamente al nombre del método concreto. Usar el controlador multi-acción es especialmente necesario cuando se tiene mucha funcionalidad en un solo controlador, pero requiere tener múltiples puntos de entrada para el controlador, haciendo más difícil de seguir su comportamiento.

8.3.4 Controladores de comando

Los controladores de comando de Spring son una parte fundamental del paquete Web MVC de Spring. Los controladores de comando proveen de un camino para interactuar con datos y unir dinámicamente los parámetros del HttpServletRequest para los objetos de datos especificados. Son parecidos al rol que desempeña el ActionForm de Struts, pero en Spring, los objetos de datos no tienen la interfaz implementada de un framework especifico. Aquí tenemos los siguientes controladores de comando:

- AbstractCommandController – es un controlador de comando que se puede usar para crear nuestro propio controlador de comando, capaz de unir los parámetros de petición al objeto de datos especificado. Esta clase no ofrece funcionalidad de formulario, sin embargo ofrece capacidades de validación y deja que podamos especificar en el controlador mismo que hacer con el objeto de comando que ha sido rellenado con los valores de los parámetros en la petición.

- AbstractFormController – un controlador de comando que ofrece apoyo en los fomularios submiteados (enviados en la petición). Usando este controlador se pueden modelar formularios y rellenarlos usando objetos de comando que regresaremos en el controlador. Después que un usuario ha rellenado un formulario, el AbstractFormController enlaza los campos, los valida con un objeto de comando y regresa el objeto al controlador que realizara la apropiada acción. Soporta capacidades como: invalidar formularios submiteados (re-submitear), validaciones, y workflow de formulario normal. Se pueden implementar métodos para determinar que vistas son usadas por los formularios en presentación. Este controlador se puede usar si necesitamos formularios, pero no se puede especificar que vistas se quiere mostrar al usuario en el contexto de aplicación.

- SimpleFormController – un controlador de formulario que provee incluso mas soporte cuando se crea un formulario con una correspondencia con objetos de comando. El SimpleFormController nos deja especificar un objeto de comando, una vista de nombre para el formulario, una vista de nombre para la página que queremos mostrar al usuario cuando un formulario ha sido enviado en la petición, y muchas mas cosas.

- AbstractWizardFormController – el nombre de la clase sugiere que es una clase abstracta – el controlador que creemos debe extender esta clase. Esto significa que se debe implementar los métodos validatePage(), processFinish() y processCancel().

8.4 Mapeos manejadores

Usando mapeos manejadores se puede mapear peticiones web entrantes hacia los apropiados controladores. Hay algunos mapeos manejadores listos para usarse como por ejemplo SimpleUrlHandlerMapping o el BeanNameUrlHandlerMapping, pero primeramente vamos a examinar el concepto general de HandlerMapping.

La funcionalidad básica de un proveedor de HandlerMapping es la liberación de un HandlerExecutionChain, el cual deberá contener el controlador que machera la petición entrante, y además contendrá una lista de manejadores interceptadores que serán aplicados a la petición.Cuando una petición viene, el DispatcherServlet da la petición al mapeo controlador para conseguir inspeccionar la petición y levantar el HandlerExecutionChain apropiado. Entonces el DispatcherServlet ejecutara el manejador y los interceptadores de la cadena de ejecución (si existe alguno).

El concepto de mapeos controladores configurables que pueden opcionalmente contener interdeptadores (ejecutar antes y después que sea ejecutado el manejador en curso) es extremadamente poderoso. Un conjunto de funcionalidad de soporte puede ser construido dentro de los HandlerMappings.

Describiremos dos de los más comunes mapeos controladores de Spring. Ambos proceden del AbstractHandlerMapping y comparten las siguientes propiedades:

- interceptors: lista de interceptores listos para usarse.HandlerInterceptors.
- defaultHandler: el controlador por defecto para usar, cuando se utiliza este mapeo controlador no hay resultado en el manejador correspondiente.
- order: basado en el valor de propiedad ordenada, Spring ordena todos los mapeos controladores disponibles en el contexto y aplica el primer manejador correspondiente.
- alwaysUseFullPath: si esta propiedad esta activada, Spring usara el path absoluto dentro del servlet en ejecución para encontrar el manejador apropiado.Si esta propiedad esta desactivada , (el defecto), se utilizara el path relativo para encontrar el manejador apropiado.
- urlPathHelper: usando esta propiedad se puede retocar el UrlPathHelper en uso cuando inspeccionamos URLs. Normalmente, no se debe cambiar el valor por defecto.
- urlDecode: el valor por defecto de esta propiedad es falso. El HttpServletRequest regresa la URL que ha solicitado la petición y la URIs que no ha decodificado. Si no se quiere que la URL y la URI sean decodificadas antes de que un HandlerMapping las use para encontrar un controlador apropiado, se tiene que poner a true. El método decodificador usa la decodificación especificada para las peticiones en el esquema de decoficación de ISO-8859-1.

8.4.1 BeanNameUrlHandlerMapping

Un simple pero poderoso mapeo controlador es el BeanNameUrlHandlerMapping, el cual mapea peticiones entrantes HTTP a nombres de beans definidos en el contexto de aplicación web. Digamos que queremos ofrecer al usuario una inserción en una cuenta y nosotros queremos proveer un apropiado controlador y una vista JSP que envía el formulario. Cuando usamos el BeanNameUrlHandlerMapping, nosotros podemos mapear la petición HTTP con la URL http://samples.com/editaccount.form al apropiado Controller de esta manera:

```
<beans>
  <bean                                                                    id="handlerMapping"
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

  <bean                                                              name="/editaccount.form"
class="org.springframework.web.servlet.mvc.SimpleFormController">
    <property name="formView" value="account"/>
    <property name="successView" value="account-created"/>
    <property name="commandName" value="account"/>
    <property name="commandClass" value="samples.Account"/>
  </bean>
<beans>
```

Todas las peticiones que entren a través de la URL /editaccount.form ahora serán manejadas por el form Controller que hemos definido. Por supuesto es necesario definir un servle-mapping en el web.xml para procesar todas las peticiones que termina con.form.

```
<web-app>
    ...
    <servlet>
        <servlet-name>sample</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

   <!-- maps the sample dispatcher to *.form -->
    <servlet-mapping>
        <servlet-name>sample</servlet-name>
        <url-pattern>*.form</url-pattern>
    </servlet-mapping>
    ...
</web-app>
```

Si se quiere utilizar el BeanNameUrlHandlerMapping, no es necesario tenerlo definido en el contexto de aplicación (como se indica en las líneas anteriores). Por defecto, si no puede ser encontrado un mapeo controlador el DispatcherServlet creates crea una BeanNameUrlHandlerMapping por nosotros.

8.5 Vistas y su resolución

Todos los frameworks MVC para aplicaciones web siempre encaminan el resultado hacia una dirección o modelo de vista. Spring proporciona vistas con los resultados, los cuales son modelos renderizados hacia un explorador sin tener en cuanta una tecnología especifica. Fuera de las funcionalidades que proporciona Spring, Spring es capaz de usar JSP, Velocity templates and XSLT views por ejemplo.

Los dos interfaces mas importante que ofrece Spirng hacia los manejadores para la parte de vista son ViewResolver y View. El ViewResolver establece un mapeo entre el nombre de vista y las vistas actuales. El interfaz View dirige la preparación de la petición y da la petición a una de las tecnologías de la parte de vista.

8.5.1 Vistas de resolución – el interfaz ViewResolver

Como ya hemos visto en la parte de Controladores, todos los controladores del framework Web MVC de Spring regresan una instancia de un ModelAndView . Las vistas en Spring son dirigidas por un nombre de vista y resueltas por un resolutor de vistas. Spring tiene una serie de resolutores de vista. Aquí mostramos los siguientes:

ViewResolver	Descripción

AbstractCachingViewResolver	Un resolutor de vista abstracto que realiza la parte de cacheado en la vista. Algunas vistas necesitan una preparación antes de ser usas, extendiendo esta vista ofrecemos resolución de cacheado a las vistas.
XmlViewResolver	Una implementación de ViewResolver que acepta un fichero de configuración escrito en XML con el mismo DTD de la factoría de beans de Spring. El fichero de configuración por defecto es /WEB-INF/views.xml.
ResourceBundleViewResolver	Una implementación de ViewResolver que usa definiciones de bean en un ResourceBundle, especificado por el nombre base del bundle. El bundle es típicamente definido en un fichero properties file, localizado en el classpath. El nombre por defecto del fichero es views.properties.
UrlBasedViewResolver	Una implementación simple del interfaz ViewResolver que realiza la resolución directa de nombre de vista simbólicos para URLs, sin una definición explicita del mapeo. Esto es apropiado si los nombres simbólicos son resueltos hacia recursos de vista de una forma sencilla sin la necesidad de mapeos arbitrarios.
InternalResourceViewResolver	Conjunto de susclases de UrlBasedViewResolver que ofrecen InternalResourceView (i.e. Servlets y JSPs), y subclases tales como JstlView y TilesView. Las clases vista para todas las vistas generadas por este resolutor pueden ser especificadas por medio de setViewClass(..).
VelocityViewResolver / FreeMarkerViewResolver	Conjunto de subclases de UrlBasedViewResolver que ofrecen VelocityView (i.e. Velocity templates) or FreeMarkerView respectivamente y customizan susbclases de estos.

Si usamos JSP como tecnología de la parte de vista se puede usar UrlBasedViewResolver. Esta vista resuelve la transformación de un nombre de vista a una URL y une la petición hecha al RequestDispatcher a la vista a renderizar.

```
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.UrlBasedViewResolver">
  <property name="prefix" value="/WEB-INF/jsp/"/>
  <property name="suffix" value=".jsp"/>
</bean>
```

Cuando es regresado test como viewname, esta vista se resuelve uniendo la petición al RequestDispatcher que enviara la petición a /WEB-INF/jsp/test.jsp.

Cuando utilizamos diferentes tecnologías para la parte de vista en una aplicación web, se puede usar ResourceBundleViewResolver:

```
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
  <property name="basename" value="views"/>
  <property name="defaultParentView" value="parentView"/>
</bean>
```

El ResourceBundleViewResolver inspecciona el ResourceBundle identificado por el nombre base y por cada vista que es capaz de asociar, usa el valor de la propiedad [viewname].class como la clase vista y el valor de la propiedad [viewname].url como url de la vista. Como se puede ver, se puede identificar una vista padre, del cual todas las vistas en el fichero de propiedades extienden de ella. En esta línea por ejemplo se puede especificar una clase vista por defecto como una url.

8.5.2 Encadenar ViewResolvers

Spirng contiene más que una única vista de resolución. Esto significa que se puede cambiar el resolutor, y por ejemplo ignorar vistas específicas en ciertas circunstancias. Optimizar los resultores de vistas es bastante sencillo, simplemente añadimos el resolutor al contexto de aplicación y si es necesario establecemos la propiedad de orden, para especificar un orden. Cuanto más alto es el nivel de prioridad configurado mas tarde se resolverá dentro de la cadena de resolución de prioridades.

En el siguiente ejemplo, la cadena de resolutores de vista consiste en dos resolutores, un InternalResourceViewResolver (el cual es automáticamente posicionado como último resolutor en la cadena) y un XmlViewResolver para especificar una vista Excel (lo cual no es soportado por el InternalResourceViewResolver):

```
<bean id="jspViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
  <property name="prefix" value="/WEB-INF/jsp/"/>
  <property name="suffix" value=".jsp"/>
</bean>

<bean id="excelViewResolver" class="org.springframework.web.servlet.view.XmlViewResolver">
  <property name="order" value="1"/>
  <property name="location" value="/WEB-INF/views.xml"/>
</bean>

<!-- in views.xml -->

<beans>
  <bean name="report" class="org.springframework.example.ReportExcelView"/>
</beans>
```

Si un resolutor especifico de vista no consigue resultado en una vista, Spring inspecciona el contexto para ver si otros resultores de vista están configurados. Si existen los examinara y si no lanzara una Exception.

8.5.3 Redireccionar a vistas

Un controlador normalmente regresa un nombre de vista lógico, el cual un resolutor de vista resuelve a una tecnología de vista concreta. Para tecnologías de vista como JSPs se procesa mediante la vía del Servlet/JSP , este es lo que se usaba en los controladores InternalResourceViewResolver / InternalResourceView los cuales al final acababan usando una redirección o inclusión, usando el Servlet API's RequestDispatcher.forward(..) o RequestDispatcher.include(). Para otras tecnologías de vista, tal como Velocity, XSLT, etc., la vista por si misma produce el contenido sobre la cadena de respuesta.

A veces es deseable usar una redirección HTTP como respuesta al cliente, antes que la vista sea renderizada. Esto es deseable cuando por ejemplo un controlador ha sido llamado con POSTed data y la respuesta es delegada en otro controlador (por ejemplo en un envió de formulario exitoso). En este caso, una redirección interna normal significara que otro controlador vera el mismo POST data, lo cual es potencialmente problemático si puede confundir con otros datos esperados. Otra razón para redirecionar antes de ver los resultados es que elimina la posibilidad al usuario de hacer un envío doble de un formulario de datos. El navegador enviara inicialmente un POST, nosotros enviaremos una redirección y haremos un posterior GET porque de echo, y por lo que se esta tratando, la actual pagina no refleja el resultado de un POST, sino mas bien de un GET, así no hay posibilidad para que el usuario puede accidentalmente re-POSTear el mismo dato haciendo un refresco por ejemplo. El refresco forzaría un GET de la página resultado, no reenviar el POST con los datos iniciales.

RedirectView

Una manera de forzar una redirección es que el resultado de un controlador cree y regrese una instancia de RedirectView de Spring. En este caso DispatcherServlet no usara el mecanismo normal de resolución de vista, más bien le han facilitado la propia vista (redirect).

El RedirectView simplemente termina haciendo uso de una llamada a HttpServletResponse.sendRedirect(), el cual regresa al explorador del cliente como una redirección HTTP.

8.6 Utilización de los valores locales del cliente

La mayoría de la parte de la arquitectura de Spring soportan internacionalización, y como consecuencia el framework Web MVC de Spring también. DispatcherServlet habilita de forma automática resolución de mensajes usando la configuración local del cliente. Esto se hace con los objetos LocaleResolver.

Cuando una petición se desencadena, DispatcherServlet mira el resolutor local y si encuentra uno intenta usar las propiedades locales. Usando el método RequestContext.getLocale() se puede siempre obtener el objeto local que ha sido resuelto por el resolutor local.

A pesar que la resolución local es automática, se puede recibir la petición y interceptar el mapeado controlador para cambiar los parámetros locales en circunstancias específicas, por ejemplo, basándonos en parámetros de la petición.
Los resolutores y los interceptores están definidos en el paquete org.springframework.web.servlet.i18n y son configurados en contexto de aplicación.
Aquí tenemos una selección de resolutores locales incluidos en Spring:

Resolutores Locales	Descripción
AcceptHeaderLocaleResolver	Inspecciona en la cabecera de la petición el lenguaje aceptado que es enviado por el buscador del cliente
CookieLocaleResolver	inspecciona si una Cookie podría existir en el cliente
SessionLocaleResolver	regresa parámetros locales de la sesión que podrían ser asociados a la petición del cliente
LocaleChangeInterceptor	cambia los parámetros locales de una petición

8.7 Usando Temas de Spring

8.7.1 Introducción

Un tema es básicamente una colección estática de recursos que afectan al estilo visual de la aplicación, típicamente hojas de estilos e imágenes. El uso de temas permite al usuario percibir un estilo global en la aplicación que maneja.

8.7.2 Definiendo temas

Cuando se quieren utilizar temas en la aplicación web se tiene que levantar un org.springframework.ui.context.ThemeSource. El inteface WebApplicationContext extiende ThemeSource pero delega las responsabilidades para una implementación concreta. Por defecto esta delegación será en un org.springframework.ui.context.support.ResourceBundleThemeSource que cargara los ficheros de propiedades desde el directorio raíz (root) del classpath. Si se quiere utilizar una implementación propia de ThemeSource o si es necesario configurar prefijos de nombre base del ResourceBundleThemeSource, se tiene que registrar un bean en el contexto de aplicación con la palabra reservada "themeSource". El contexto de aplicación Web automáticamente detectara ese bean y lo ejecutara para su uso.

Cuando usamos el ResourceBundleThemeSource, un tema se ha definido en un fichero simple de propiedades. La lista de propiedades del fichero con los recursos crea el tema. Aquí hay un ejemplo:

```
styleSheet=/themes/cool/style.css
background=/themes/cool/img/coolBg.jpg
```

La manera de identificar las propiedades se realiza mediante los nombres usados para referir los elementos del tema desde el código de parte de vista. Para un JSP típicamente se utilizara el tag customizado srping:theme, el cual es muy similar al tag spring:message. El siguiente fragmento de JSP usa el tema definido anteriormente:

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<html>
  <head>
    <link rel="stylesheet" href="<spring:theme code="styleSheet"/>" type="text/css"/>
  </head>
  <body background="<spring:theme code="background"/>">
    ...
  </body>
</html>
```

Por defecto, el ResourceBundleThemeSource usa un nombre base de prefijo vació. Como resultado el fichero de propiedades será cargado desde el directorio raíz del classpatch, así para nuestro tema cool.properties definiremos en un directorio del classpath e.g. in /WEB-INF/classes. ResourceBundleThemeSource usa el mecanismos de carga de recursos estándar de Java, teniendo en cuenta los temas de internacionalización por ejemplo nosotros podemos tener un /WEB-INF/classes/cool_nl.properties para referenciar una configuración especial para un país en concreto.

8.7.3 Resolutores de Temas

A la hora de decidir que temas son cargados el DispatcherServlet buscara un bean llamada “themeResolver” para encontrar el ThemeResolver que usara para implementarlo. Es posible detectar que tema va a ser usado y también se puede alterar el tema.

8.8 Soporte en varias partes (multipartes) de Spring

Srping ha construido soporte en varias partes para manejar carga de ficheros en aplicaciones web. El diseño para el soporte en varias partes se ha realizado con los pluggin de objetos MultipartResolver objects, definidos en el paquete org.springframework.web.multipart package. Además Spring ofrece servicios de MultipartResolvers para usar con Commons FileUpload (http://jakarta.apache.org/commons/fileupload) y COS FileUpload (http://www.servlets.com/cos).

Por defecto, el manejador de varias partes esta hecho por Spring, algunos desarrolladores quieren implementar ellos mismos los manejadores de varias partes. Se puede habilitar nuestro propio manejador añadiendo un resolutor multiparte en el contexto web de la aplicación. Después de que esto se haya hecho cada petición será inspeccionada para ver si esta compuesta de varias partes. Si no es en varias partes, la petición continua un flujo normal. Sin embargo, si es una petición en varias partes, el MultipartResolver que es declarado en el contexto será el que se use. A continuación el atributo de varias partes de la petición será tratado como cualquier otro atributo.

8.9 Usando la librería de tag para formularios de Spring

Desde la versión 2.0, Spring provee de un conjunto de tags para manejar elementos en un formulario cuando usamos JSP y Spring Web MVC. Cada tag provee de un conjunto de atributos con su correspondencia hacia un tag de HTML haciendo los tags familiares e intuitivos para su uso. El Tag generado HTML es un HTML 4.01/XHTML 1.0 compilado.

La librería de tag de formulario de Spring esta integrada en el Spring Web MVC, dando acceso a los tags desde objetos de comando y con datos referenciados desde el controlador. Los tags de formulario hacen mas sencillo el desarrollo, lectura y mantenimiento de JSP’s.

La librería de tag de formulario viene en el jar spring.jar. El descriptor de librería se llama srping-from.tld.
Para usar los tag de librería se debe añadir la siguiente directiva en la parte de cabecera del jsp:

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

Aquí tenemos la lista de tags: form tag,input tag,checkbox tag,radiobutton tag,password tag,select tag,option tag,options tag,textarea tag,hidden tag ,errors tag

8.10 Manejar excepciones

Spring provee HandlerExceptionResolvers para manejar aquellas excepciones no esperadas que ocurren mientras una petición esta siendo manejada por un controlador que esta tratando la petición. HandlerExceptionResolvers reensambla los mapeos de excepción que podemos definir en el descriptor de la aplicación web web.xml. Sin embargo, este manejador provee un camino más flexible para manejar las excepciones. Provee información sobre que manejador ha sido ejecutado cuando la excepción se ha producido. However, they provide a more flexible way to handle exceptions. Además, un posibilidad de programación para manejar la excepción da mas opciones para responder de una forma apropiada antes que la petición sean enviada hacia otra URL (el resultado final es el mismo que cuando usamos el servlet de mapeos de excepción específicos).

Además de la implementación del interfaz HandlerExceptionResolver, el cual es solo un asunto de implementación del método resolveException(Exception, Handler) y regresar un ModelAndView, podríamos utilizar el SimpleMappingExceptionResolver. Este resolutor hablita que tomemos el nombre de clase de cualquier excepción que podamos capturar y mapearla hacia un nombre de vista. Esta es una funcionalidad equivalente a las ventajas de mapeos de excepción del Servlet API, pero además es posible implementar un grano mas fino de mapeos de excepción para manejadores diferentes.

Capítulo 9.

9. Comparación de los tres Frameworks.

El itinerario que se ha seguido en la presentación de los distintos Frameworks se ha regido por las distintas fechas de aparición en la escena de los mismos en los desarrollos Web. A lo largo de las distintas descripciones, se ha podido ver como cada Framework presenta su solución a problemas comunes como internacionalización, elementos para facilitar la construcción de las páginas (librerías de Tag), como han acometido la función de recibir las peticiones y procesarlas etc.

Así Struts ha sido el primero en ser presentado por ser el primero que surgió. Como características relevantes nos encontramos las siguientes:

- Validación en cliente (JavaScript)
- Soporte de peticiones “multipart”
- Excelente sistema de mensajes de aplicación.
- Tecnología madura. Conocimiento muy extendido y actualmente muy usado.
- Herramientas de configuración ya maduras:
    - StrutsGUI (Plug-in de Visio)
    - Struts Console

Por su parte Java Server Faces, presenta las siguientes características relevantes:

- Posibilita herramientas visuales de diseño.
- Más fácil de reutilizar. (al basarse en componentes, renderers y conversores)
- Estándar <-> Abre mercado de componentes.
- Posibilidad de personalizar casi todo. (navegación, resolución de variables y propiedades, …)
- Herramientas de desarrollo:
    - Faces Console James Holmes
    - Sun Java Studio Creator (Project Rave)
    - JSF FormBuilder

Y en último lugar se ha presentado la parte que ofrece Spring como MVC Web. En el propio texto incluido ya se han detallado las principales diferencias que ofrece este FrameWok con respecto a Struts. Es un Framework robusto que mejora las capacidades de Struts, pero presenta una curva de aprendizaje alta.

La opción más atractiva para la implementación de nuestro Framework la vamos a basar en Struts. Partiremos de su base y la extenderemos. Struts proporciona una serie de facilidades que permiten poder concentrarnos en la lógica y el control del aplicativo y dejar de lado otros aspectos más mecánicos que ya tiene implementados. Es una tecnología asentada en el mercado, presente durante muchos años, lo que nos ofrece un entorno de desarrollo estable, con una rica documentación, y el conjunto de posibilidades necesarias para construir nuestro propio Framework sin demasiada complejidad tecnológica. Así podremos dedicar la mayor parte del tiempo en la construcción de los elementos de presentación que ofrecerá nuestro FrameWork, ahondado en la usabilidad de los mismos para ofrecer un Framework con el que podamos crear interfaces de gestión en intranets de una forma optima, usable y sencilla.

Capítulo 10.

10. Usabilidad

10.1 Introducción

Diseñar una aplicación no es una tarea fácil. Durante el desarrollo de una aplicación, al enfrentarse al diseño del interfaz de la misma, todo desarrollador se hace preguntas como:
- ¿Qué colores utilizo para diseñar las pantallas?
- ¿Dónde pongo el logotipo de la Empresa?
- ¿Dónde pongo el nombre de la aplicación?
- ¿Cómo puedo estructurar las funcionalidades?
- ¿Cómo navegará el usuario entre las diferentes funcionalidades?
- ¿Cómo funcionan las búsquedas? ¿Qué datos presentan como resultado?
- ¿Qué información es necesaria en cada pantalla? ¿En qué formato se presenta?
- ¿Puedo usar iconos al diseñar la aplicación? ¿Dónde los ubico?
- ¿Cómo comunico al usuario la existencia de errores en su trabajo?

Generalmente, tras un proceso de trabajo más o menos debatido, que consume plazos de desarrollo, termina por llegar a una solución que aplica, reutiliza y adapta en posteriores desarrollos.

El trabajo en diferentes equipos de desarrollo y la falta de criterios uniformes y pautas para aplicar en el desarrollo de aplicaciones conlleva que cada equipo se plantee estas mismas preguntas dando lugar a soluciones diferentes para un mismo problema.

Esta situación deriva en problemas como:
a) Ineficiencia derivada de no reaprovechar soluciones probadas.
b) Aumento de costes de desarrollo y mantenimiento.
c) Cada aplicación tiene una estructura, aspecto y funcionamiento diferente.

En definitiva, una inconsistencia y dispersión de criterios que puede transformarse en un entorno de trabajo incómodo y difícil para el usuario, lo que conlleva:
- Necesidad de reaprendizaje continúo para el usuario.
- Mayor probabilidad de cometer errores.
- Uso insatisfactorio que deriva en una pérdida de productividad.

10.2 Objetivos y proceso de creación

10.2.1 La Guía de Estilos en las aplicaciones

Para poner solución a estos problemas, un primer paso es la creación unas normas de Estilo que se centrará en aspectos clave del diseño y desarrollen el interfaz de usuario para nuestras aplicaciones lo que nos proporcionara:
• Estandarización
• Usabilidad
• Adaptación a la identidad del Framework.

Además, para favorecer la reutilización en el diseño de aplicaciones se han de crear componentes reutilizables de interfaz que permitan la mecanización del proceso de desarrollo con su consiguiente disminución de costes y tiempos.

10.2.2 Objetivos de las Normas de Estilo

Creación de una normativa que facilite la normalización, estandarización y mejora de las aplicaciones de negocio creando un entorno de trabajo uniforme, consistente y respetuoso con las normas de estilo y comunicación de las aplicaciones empresariales que queremos crear con nuestro FrameWork.
Promover la creación de aplicaciones con interfaces intuitivas, atractivas y fáciles de usar aplicando técnicas de Diseño Centrado en el Usuario.
Facilitar el desarrollo de un entorno flexible, escalable y modular a través de la identificación y desarrollo de componentes reutilizables de interfaz que optimicen los procesos de desarrollo y mantenimiento de aplicaciones.
El uso habitual, la interiorización y el desarrollo de estas pautas posibilitarán el desarrollo de aplicaciones cómodas, intuitivas, consistentes, de fácil aprendizaje, rápidas de desarrollar y que proporcionarán eficacia, eficiencia y satisfacción de los usuarios.

El interfaz de usuario es el punto de encuentro y comunicación entre un sistema y un usuario. **"**Para el usuario, el interfaz es el sistema**".** El usuario no sabe acerca de tecnología.
Un usuario identificará la calidad de una aplicación o sistema a través del funcionamiento satisfactorio de su interfaz.
El interfaz ideal es aquel que permite que una organización propietaria de un sistema logre (o supere) unos objetivos de negocio determinados a partir de la satisfacción de las necesidades de unos usuarios concretos (empleados, clientes, proveedores…) dando forma y sacando el mejor partido a la tecnología existente.

10.3 Diseño centrado en el usuario

10.3.1 Usabilidad ISO 9241-11

"Cómo un producto puede ser utilizado por usuarios específicos para conseguir metas específicas con:**Efectividad, Eficiencia y Satisfacción**."
• **Efectividad:** Es la medida del tiempo y cantidad de recursos necesarios para completar una tarea. *Una aplicación usable permite completar las tareas con agilidad, sin necesidad de aprender nuevas técnicas y con menor esfuerzo.*
• **Eficiencia:** Es la medida de lo bien que podemos completar nuestras tareas. *Una aplicación usable facilita la realización efectiva de tareas previniendo errores costosos.*
• **Satisfacción:** Es la medida del grado de suficiencia o placer con que hemos completado una tarea. *Una aplicación usable ofrece la información necesaria y da asistencia cuando es requerida ahorra esfuerzo al usuario.*

Usabilidad → Calidad

Problemas típicos derivados de una mala usabilidad en sistemas interactivos (entre ellos aplicaciones) son:
-Tiempos de respuesta excesivos.
- Pasos innecesarios para completar una tarea.
- Confusiones del usuario.
- El usuario no puede encontrar la información que necesita en cada momento.
- El sistema no funciona ni reacciona conforme a las expectativas del usuario.
- El usuario pierde el control de la aplicación.
- El usuario es incapaz de recuperarse de los errores.

El objetivo de desarrollar productos usables es permitir que los usuarios logren resultados en sus tareas teniendo en cuenta el contexto de uso de la aplicación. Para ello hay que fijarse en dos puntos claves:
• **Usuarios finales** o en su defecto representantes cualificados de los mismos que conozcan de primera mano sus necesidades y forma de trabajo.
• **Contexto de uso:** dónde se va a usar el sistema, en una oficina, ante el público, sometido a interrupciones...

Para el diseño de aplicaciones y sistemas de calidad es necesario aplicar técnicas de **Diseño Centrado en el Usuario** que tengan presente estos dos ejes.

10.3.2 Modelo de calidad en desarrollo de Software

Por su parte la ISO/IEC 9126-1 describe 6 categorías relevantes para la calidad de software durante su desarrollo:
• Funcionalidad
• Fiabilidad
• Usabilidad
• Eficiencia
• Mantenibilidad
• Portabilidad
La calidad de un producto (aplicación, sistema, software) está íntimamente relacionada con su uso satisfactorio.

A diferencia de la web, donde la competencia facilita al usuario acudir a alternativas más satisfactorias, en entornos de trabajo cerrados como intranets, aplicaciones internas, en los que el usuario no tiene capacidad de elección y debe pasar varias horas, un buen interfaz de usuario deriva en una mejor calidad de vida y de trabajo.

Figura 10-1. Categorías relevantes para la calidad del Software.

10.3.3 Diseño Centrado en el Usuario en sistemas interactivos

La ISO 13407 ofrece guías y pautas para obtener calidad en el uso mediante la incorporación de técnicas de diseño centrado en el usuario durante el ciclo de vida (creación, desarrollo y mantenimiento) de sistemas interactivos.
Describe el Diseño Centrado en el Usuario (DCU) o *User Centred Design* como una actividad multidisciplinar que incorpora conocimiento acerca de factores humanos y ergonomía así como técnicas con el objetivo de mejorar la efectividad y la productividad mejorando las condiciones de trabajo de las personas y teniendo en cuenta los posibles efectos adversos del uso en la salud humana, seguridad y rendimiento.
Existen 4 fases principales en un proceso DCU:
• Comprender y especificar el contexto de uso: cómo, dónde, en qué entorno va a ser usada la aplicación.
• Especificar los requerimientos organizacionales y del usuario: quién va a usar esa aplicación, qué características tienen sus usuarios, qué objetivos se persiguen con esa aplicación.
• Producir soluciones de diseño: creación de soluciones fácilmente modificables con el fin de introducir mejoras y corregir errores a bajo coste.
• Evaluar los diseños contra los requerimientos del usuario: mediante pruebas o tests se comprueba que el producto cumple con los requerimientos especificados corrigiendo errores e introduciendo mejoras.

10.3.4 Proceso de trabajo ISO 13407

Figura 10-2. Proceso de trabajso ISO 13407.

10.3.5 Beneficios del Diseño Centrado en el Usuario

Los beneficios de aplicar técnicas de Diseño Centrado en el Usuario siempre exceden a su coste.
80% de desviaciones en proyectos son motivadas por cuestiones de usabilidad *(Martin & McLure, 1983; Pressmen, 1992)*:

a) Continúas peticiones de cambio por parte de los usuarios.
b) Tareas no previstas inicialmente.
c) El usuario/cliente no tiene claros sus propios requerimientos.
d) Comunicación y entendimiento insuficiente entre analista y usuario.

Aplicar técnicas de Diseño Centrado en el Usuario desde fases tempranas del desarrollo proporciona beneficios como:
a) Mejor comunicación equipo de desarrollo con el usuario final que deriva en:
   i. Mejor toma de requerimientos
   ii. Menores desviaciones
   iii. Detección temprana de fallos
b) Acortamiento de plazos de desarrollo.
c) Menores costes de formación y soporte.
d) Mayor productividad y eficiencia de usuarios (la rentabilidad será mayor cuanto mayor sea el uso y número de usuarios de la aplicación).

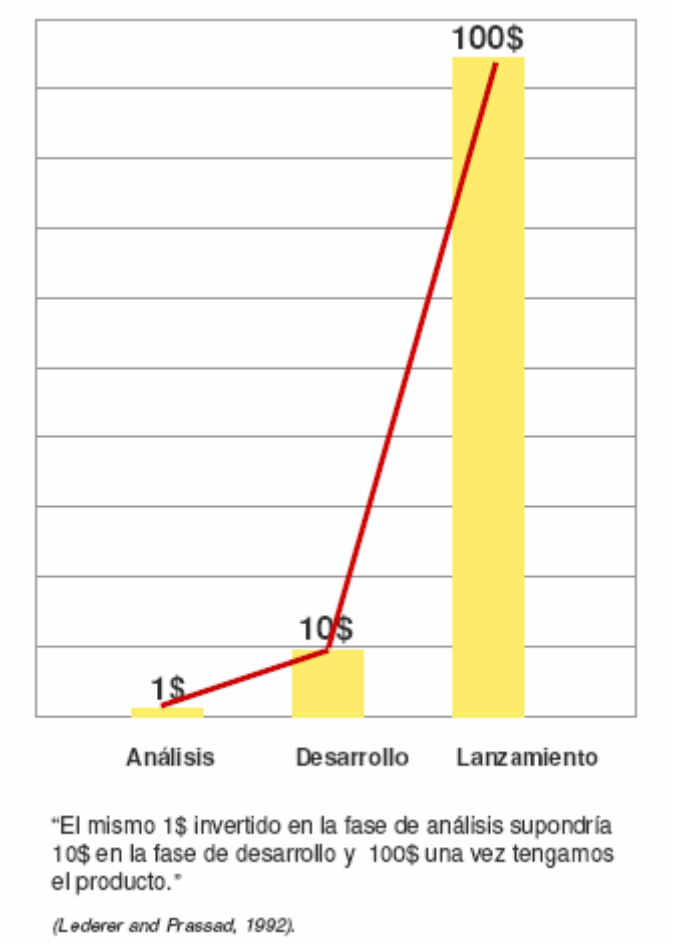


Figura 10-3. Proporción de los gastos en las distintas fases de un producto Software

10.3.6 El comportamiento del usuario

• **Cada usuario es diferente.** Entender, aceptar y respetar la diferencia es el punto de partida básico para crear aplicaciones usables. Es importante tener en cuenta su nivel de conocimiento y de aceptación de la tecnología, edad, sexo, capacidades (visión, oído), entorno de acceso.
• **El usuario tiene unos objetivos** concretos y quiere satisfacerlos rápidamente y sin complicaciones.
• Las decisiones del usuario durante el manejo de la aplicación son **muy rápidas y poco meditadas**. Utiliza el método prueba error.
• **Los usuarios esperan encontrar interfaces cómodas**; no quieren perder tiempo pensando, quieren poder actuar rápidamente.
• **Los usuarios son muy exigentes**, y exigen a la aplicación el máximo resultado con el mínimo esfuerzo.
• **Debemos evitar suposiciones y basarnos en datos empíricos.** Conocer la realidad de los grupos de usuarios, analizar estadísticas de uso y seguir un proceso de mejora continua contribuirá a mejorar la usabilidad de las aplicaciones.
• **Es más fácil replantear la estructura de nuestra aplicación que cambiar el comportamiento de los usuarios**.
• **Algunos usuarios pueden tener problemas de acceso**, debido a discapacidades físicas o sensoriales, al entorno de trabajo o a las limitaciones del sistema de acceso (software, hardware, comunicaciones).

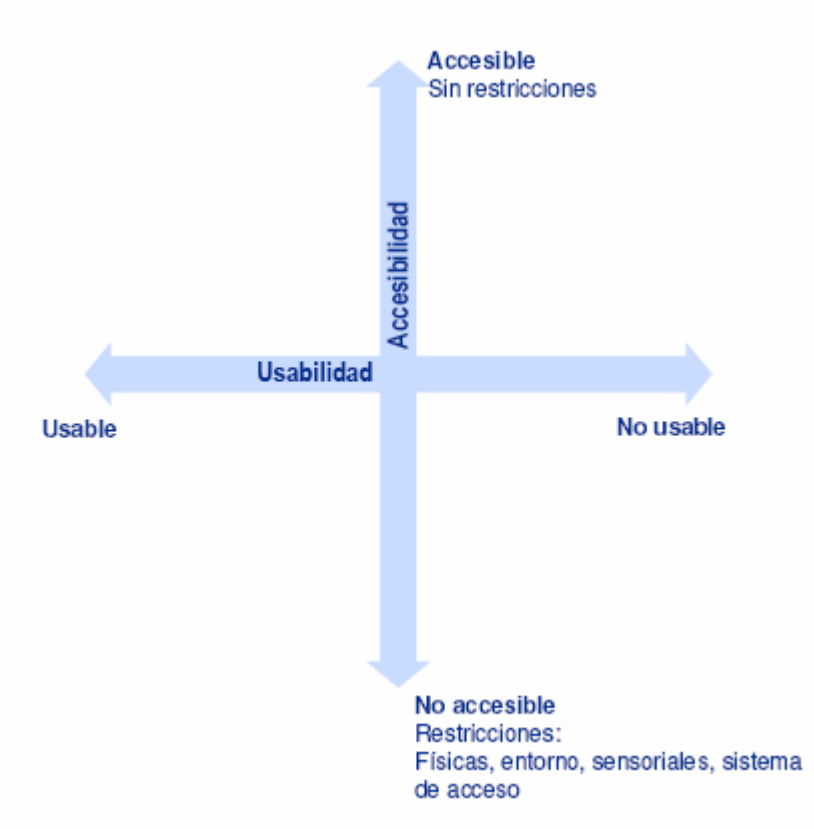


Figura 10-4. Ejes de definición de la usabilidad.

Capítulo 11.

11. Arquitectura Básica Struts

En el planteamiento inicial cuando se inicia un framework lo que tenemos que decidir es si partimos desde cero o partimos de un framework base y extenderlo.
En la comunidad de desarrollo existen varios frameworks base, algunos comerciales y otros opensource. Entre los opensource el de mayor aceptación y uso es Struts de Apache. Struts es un framework para la creación de aplicativos web basados en el patrón de diseño MVC. Proporciona una serie de facilidades que permiten poder concentrarnos en la lógica y el control del aplicativo y dejar de lado otros aspectos más mecánicos que ya tiene implementados.
El framework de Struts está basado en el Java Servlet Technology, realiza una extensión de JavaServer Pages, y es dependiente de un contenedor Web.

Struts proporciona las siguientes facilidades:

a) Un Controlador principal ya implementado (patrón Front Controller con Dispatcher View)
b) Gestión automática de los formularios con refresco entre pantallas y su posible validación
c) Gestión de los errores presentándolos en la vista
d) Internacionalización del aplicativo (multiidioma)
e) Librería de tags para poder ser usados en las vista
f) Permite añadir filtros de proceso (patrón Decorating Filter)
g) Un único fichero de configuración que liga los controladores con los formularios de la aplicación y deja clara su interacción y tiempo de vida
h) Permite añadir filtros de proceso (patrón Decorating Filter)

Otros de los puntos de fuertes de Struts son los siguientes:

a) Código Abierto de Jakarta
b) Integración correcto con J2EE
c) Buen soporte de librería de Tags
d) Facilidad de mantener el estado de los formularios
e) Tratamiento de errores unificados (Action Error)
f) Clara delimitación de las responsabilidades
g) Mantenimiento a largo plazo más fácil (más modular)

Como partes diferenciadas tenemos las siguientes:

a) Controlador (servlet)
    a. ActionServlet. Dispatcher de todas las peticiones. Mappings definidos en el fichero struts-config.xml
    b. BaseRequestProccesor
b) Clases “Action”
    a. Extienden de la clase Action y realizan la funcionalidad Real.
c) Modelo
    a. ActionForm (JavaBean con getters/setters)
d) Vista
    a. Páginas Jsp que usan librerías de tags de Struts.

En la siguiente figura podemos ver un diagrama general:

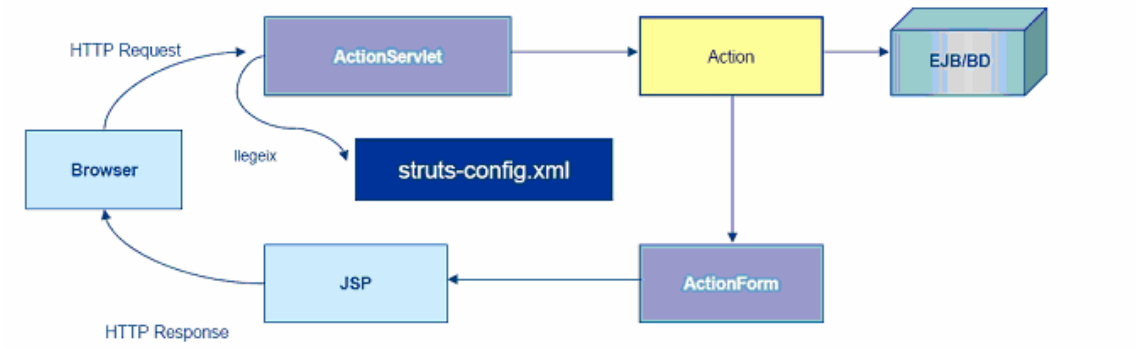


Figura 11-1. Esquema general de Struts

El diagrama de secuencias que presenta Struts es el siguiente:

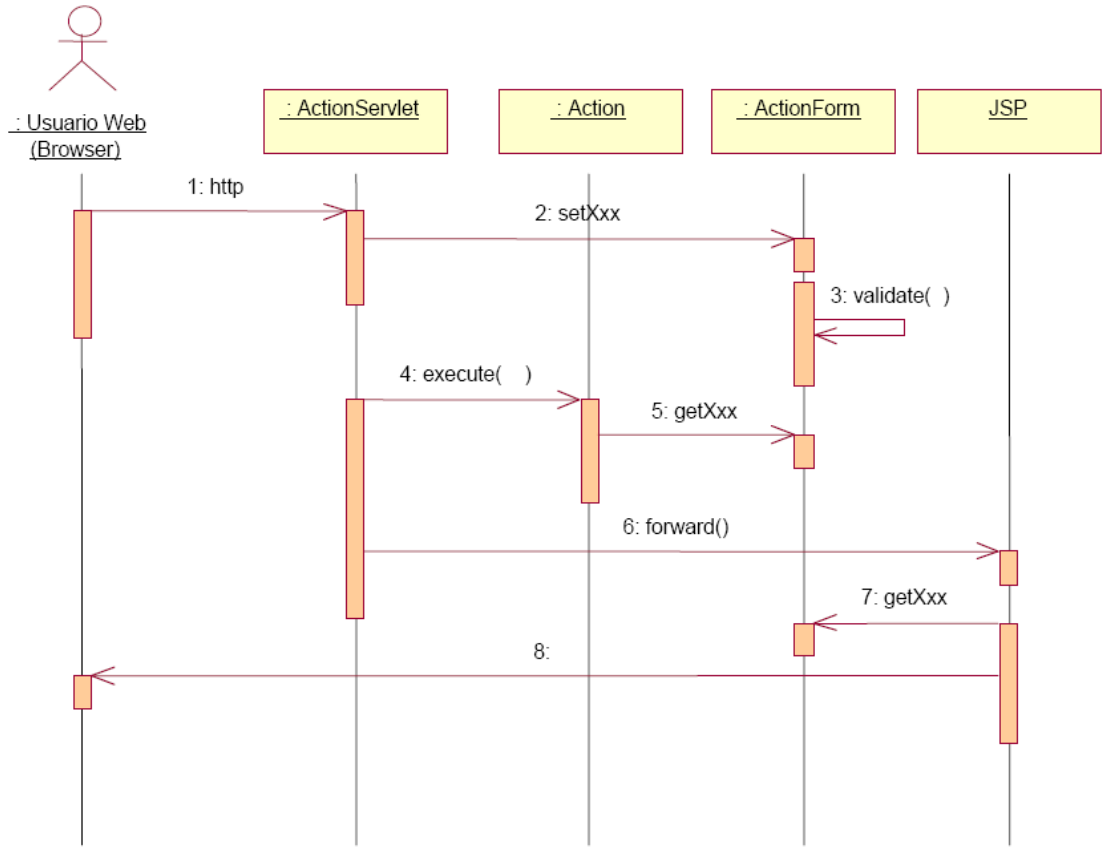


Figura 11-2. Diagrama de Secuencia Petición y Tratamiento Framework

En el gráfico de arriba se muestra el funcionamiento general de la arquitectura Struts:

1) La petición llega al Front Controller (el servlet ActionServlet)

2) Éste popula un objeto java formulario (inicializa sus datos) a partir de la petición http y llama a su método de validación

3) Realiza una llamada a la clase Action encargada de resolver la petición

4) Decide qué vista mostrar a continuación y redirige la petición a ésta

Capítulo 12.

12. Framework de Desarrollo JUoc

El framework JUoc (José Uoc) está inmerso en una arquitectura MVC (Modelo-Vista-Controlador)

El framework de desarrollo de aplicaciones J2EE JUoc utiliza como base Struts y lo extiende añadiendo un conjunto de servicios adicionales.

Partiendo de Struts, JUoc ha extendido el modelo. En el siguiente gráfico se recoge la arquitectura lógica de las aplicaciones construidas con JUoc. Solo se ha desarrollado la parte de presentación.

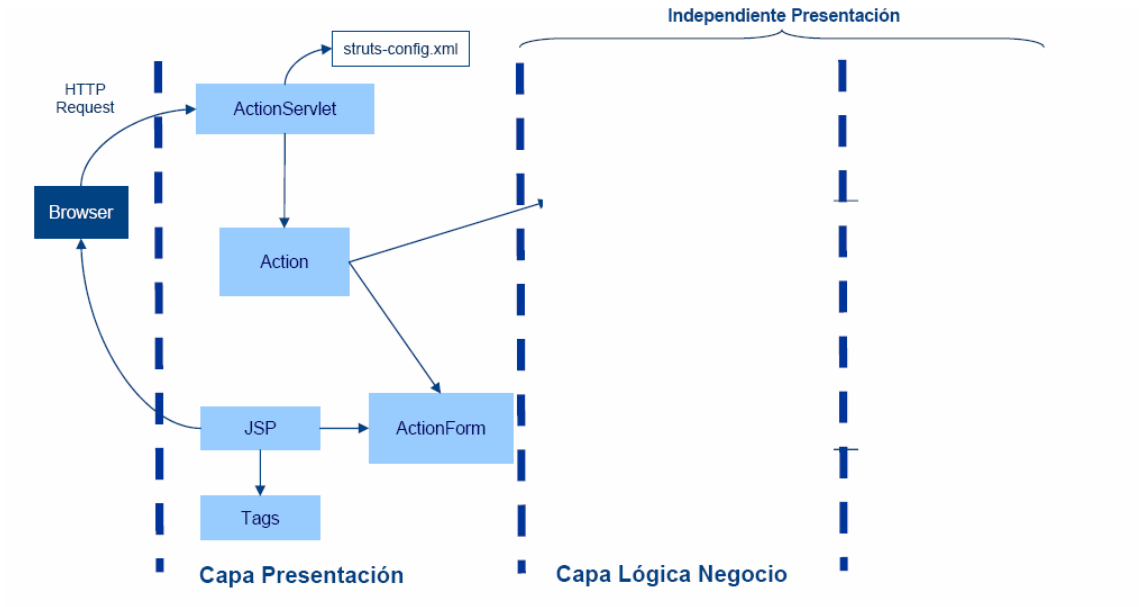


Figura 12-1. Arquitectura lógica de aplicaciones construidas con JUoc

Ventajas en la utilización del Framework de desarrollo José UOC (en adelante JUoc).

a) Infraestructura prefabricada

   El framework JUoc reduce la codificación y sobre todo la puesta en marcha, ya que proporciona subsistemas que sabemos que ya funcionan. En definitiva, proporciona código que no se tendrá que mantener ni rescribir.

b) Proporciona una arquitectura
   Cualquier programador que trabaje con JUoc no deberá invertir una gran parte de su tiempo en buscar las clases necesarias, interconectarlas o descubrir los métodos que contienen. JUoc oculta toda esta complejidad dando un nivel alto de abstracción.

c) Acelerar y simplificar el ciclo desarrollo de aplicaciones.

d) Facilidad en la factorización de aplicaciones, asegurando un comportamiento y un nivel de calidad homogéneos entre ellas.

e) Reducción del mantenimiento

f) Buena base para la industria de componentes

JUoc ofrece una serie de facilidades para la creación de aplicaciones de gestión en aplicaciones denominadas de Intranet. Este tipo de aplicaciones son de uso interno en las empresas para la manipulación de la información de sus procesos de negocio, información de Base de Datos de Personas etc.

JUoc define una serie de servicios tal y como muestra el siguiente esquema:

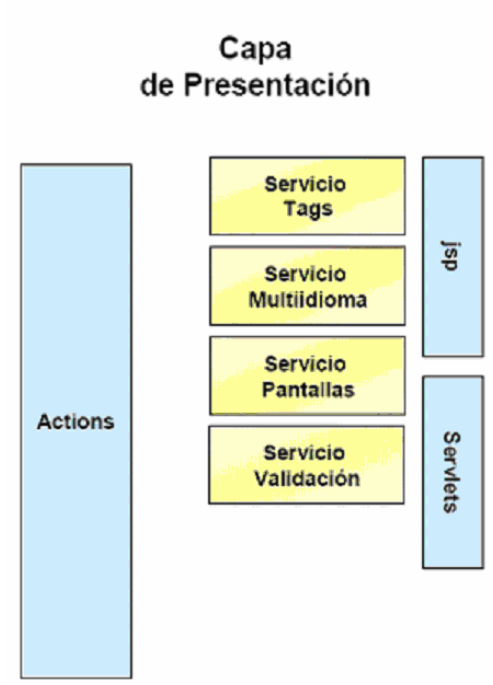


Figura 12-2. Servicios que ofrece JUoc

Servicio de Pantallas

Permite aislar todos los elementos comunes de diseño en plantillas independientes. Basado en Tiles.
Permite la definición de layouts de presentación: layout con 4 partes (título, menú, cabecera, cuerpo y pie).
Estos layouts permiten definir un look&feel genérico a toda la aplicación de forma que un cambio al look&feel de la aplicación no implique ningún cambio a las páginas jsp creadas, y solo requiera un cambio de configuración

Servicio de Multiidioma

Permite la definición de aplicaciones internacionalizadas, consiguiendo como mayor beneficio el desarrollo de la aplicación sin importar que posteriormente se añadan nuevos idiomas a la aplicación
Basada en el API estándar I18N y Jakarta.

Servicio de Tags

Proporciona tags para usar en las páginas JSP. Entre las utilidades que ofrece tenemos:

• Formularios y componentes con modo (una sola página para crear, editar, o consultar), de forma que se presenten editables o no dependiendo del perfil del usuario o de la acción a realizar
• Indicación de campos requeridos
• Conversión automática de datos
• Campos fecha con selección por calendario
• Independencia de codificación de las páginas si el portal es con o sin frames
• Indicación al usuario de que se está procesando la petición cuando ésta ha sido enviada al servidor
• Comprobación de posible pérdida de los cambios realizados en una pantalla si el usuario pulsa una opción que no realiza la grabación de los cambios e implica salir de esa pantalla
• Selección del valor de un textfield abriendo una nueva ventana de selección
• Ayuda asociada a los componentes

Servicio de Validación

Permite presentar al usuario errores de entrada de datos (campo obligatorio, campo entero, campo fecha, etc.) definiendo en un fichero qué tipo de validación realizar para cada formulario de la aplicación. Proporciona la capacidad de validación de los datos de los formularios tanto en el cliente como en el servidor. Basado en Validator.

12.1 Capa Cliente

Se mantendrá un nivel de intercambio de información razonable con el cliente. Salvo causa mayor que lo justifique no se utilizarán applets para esta capa, utilizando únicamente un browser convencional como cliente. Se deberá asegurar la compatibilidad con versiones de Internet Explorer igual o superior a la 5.5.

Para aplicaciones con salida a Internet se tendrá en cuenta la compatibilidad con los diversos exploradores (MS Internet Explorer, Mozilla / Netscape).

12.2 Capa de presentación

Se empleará la extensión de Struts proporcionada por JUoc. Para todas aquellas necesidades en las que JUoc no aporte una solución particular, se seguirán las directrices marcadas por la especificación de Struts.
Mediante esta arquitectura, se consigue independizar al máximo la capa de presentación de la capa de lógica de aplicación.

Se emplearán únicamente JSPs, Servlets, clases y ficheros de propiedades, y se intentará potenciar la legibilidad del código y la reutilización de componentes.
Se deberá estandarizar el diseño de la aplicación con el uso de estilos CSS separando de este modo la presentación del contenido, se deberán definir estilos comunes que se utilicen en toda la aplicación con el objetivo de que todas las aplicaciones tengan un “look & feel” standar.
Para la apertura de ventanas de dialogo se deberá utilizar window.showModalDialog para asegurar el funcionamiento dentro de un terminal server, para evitar tener un jsp contenedor de otro que permita navegación, se deberá incluir <base target="_self"/> en el <head> de la página que se muestra en modo modal..Se podrá utilizar también un window.open siempre que no se haga una carga inicial de la ventana en blanco y un posterior submit.
Se deberán seguir las siguientes normas para una correcta programación:

- Las tablas deben usarse para mostrar datos tabulares (es decir, ‘tablas’ de datos), no para alinear componentes, salvo en casos en los que no haya una forma clara de hacerlo con CSS.

- Las propiedades de presentación de los componentes se declararan mediante clases CSS (class=”nombreClase”), o bien mediante el atributo style (style=”width:20px”). Se destierran por tanto, etiquetas como <font> y atributos como width, height o margin.

- Se deberán definir clases CSS comunes a toda la aplicación, tratando de minimizar el número de ellas, de modo que las clases título, tabla, etiqueta, etc.. sean comunes por aplicación.

- Se deberá definir un nombre de la clase que sea autoidentificativo.

12.3 Resto de Capas

Para el resto de capas se utilizara Struts sin extender.

Capítulo 13.

13. Arquitectura Básica

En este punto veremos la arquitectura de presentación sobre la que se basa el framework. Mostramos el funcionamiento de las peticiones y su resolución.

13.1 Componentes Controlador

En una arquitectura MVC (arquitectura en la que se basa el framework), el controlador lleva a cabo varias acciones. Éstas incluyen la recepción de la petición de un cliente, invocar una operación de la lógica de negocio y coordinar la vista a devolver al cliente. En este apartado se exponen los componentes de presentación que realizan la funcionalidad de controlador en la arquitectura MVC. Veremos como Struts gestiona esta arquitectura y cómo se ha extendido en el framework.

13.1.1 ActionServlet

El ActionServlet es el Servlet que se encarga de recibir todas las peticiones recibidas y redirigirlas al gestor apropiado. En Struts, esta clase ya viene implementada y no es necesario modificarla.
En caso de querer añadir nuevo comportamiento al tratamiento de las peticiones (comprobar autentificación, realizar log, etc.) Struts recomienda extender la clase.
En el framework se ha extendido esta clase para definir, entre otros aspectos, la seguridad y el control de excepciones (clase *com.juoc.struts.base.BaseRequestProcessor*).
En el momento de crear una nueva aplicación con el framework es necesario definir en el fichero web.xml qué servlet gestionará todas las peticiones recibidas.

En el fichero de deploy de la aplicación (web.xml) debe definirse el servlet que actuará de dispatcher de todas las peticiones, en nuestro caso 'org.apache.struts.action.ActionServlet'.

```
<servlet>
<servlet-name>action</servlet-name>
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
<init-param>
<param-name>config</param-name>
<param-value>/WEB-INF/struts-config.xml</param-value>
</init-param>
<init-param>
<param-name>debug</param-name>
<param-value>2</param-value>
</init-param>
<init-param>
<param-name>detail</param-name>
<param-value>2</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
```

Código 13-1 Definición Servlet Central Peticiones

Además, es importante indicar la ruta en la que se encuentra el fichero de configuración 'struts-config.xml', que se encuentra en el directorio WEB-INF → struts-config.xml

Dentro de los parámetros de inicialización del servlet (especificados con init-param) podemos definir:

Parámetro	Descripción
config	Path de contexto relativo al fichero de configuración (por defecto /WEB-INF/struts-config.xml). Este es el fichero en el que se definirá cual es la correspondencia entre una url y una clase de la aplicación que gestione la petición
config/sub1	Para subaplicaciones adicionales, podemos especificar para cada subaplicación un fichero de configuración diferente. En sub1 se indicaría el prefijo de la subaplicación
debug	Nivel de debug para el servlet (0 por defecto)
detail	Nivel de detalle del debug (por defecto 0)
validating	Indica si hay que usar un parser XML para comprobar la validez del fichero de configuración (por defecto true). Se recomienda que se especifique true para detectar problemas
application	Especifica el nombre y ubicación del fichero .properties que actuará como repositorio de literales multiidioma

Una vez el controlador recibe la petición de un cliente, delega su gestión a una clase que conoce cómo ejecutar la lógica asociada a la acción de la petición. En Struts, esta clase, por lo general, es descendiente de *org.apache.struts.action.Action* (veremos más adelante que existen otros tipos de Action en Struts y en el framework).

Mapping de URLs a Action Servlet

Uno de los puntos más importantes en la arquitectura de struts es el mapeo de URLS. Para que el contenedor conozca que toda petición URL de la aplicación debe pasar por el ActionServlet es necesario especificar en el fichero */WEB-INF/web.xml este mapeo.*

```
<servlet-mapping>
       <servlet-name>action</servlet-name>
       <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Código 13-2 Configuración Mapping URLs a Action Servlet

De esta manera indicamos que toda url que finalice con el patrón *.do* deberá redirigirse al servlet indicado con nombre *action* (se ha definido previamente en el fichero web.xml)

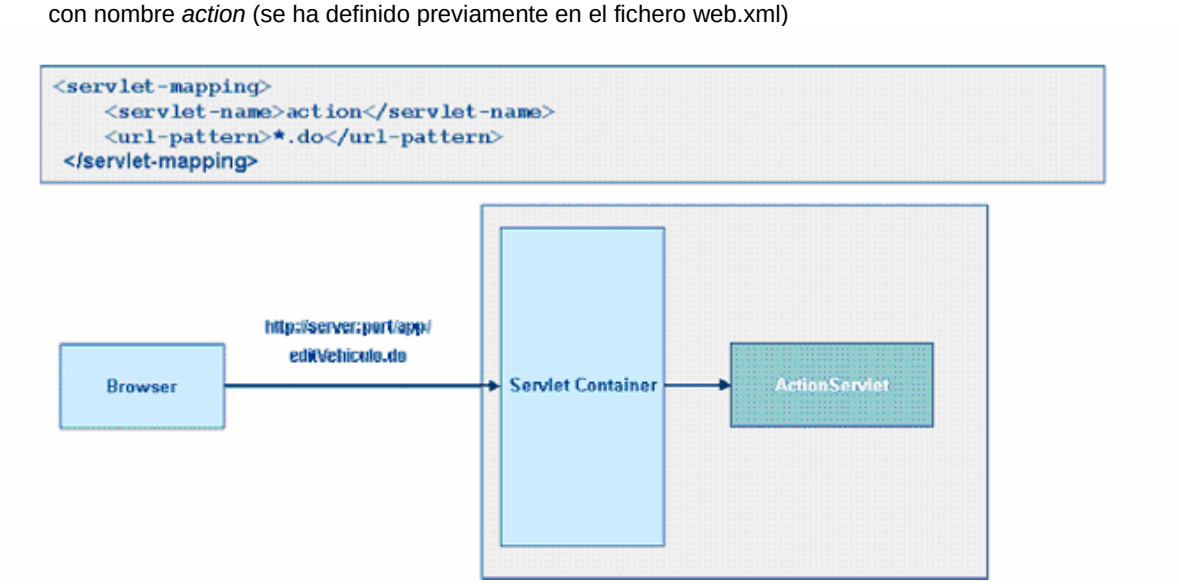


Figura 13-1. Mapeo del patron '*.do´ de las urls al ActionServet.

13.1.2 RequestProcessor

Toda petición que recibe el ActionServlet es redirigida para su tratamiento a la clase RequestProcessor que se haya configurado para la aplicación y se realizan las llamadas mostradas en el siguiente diagrama de secuencia:

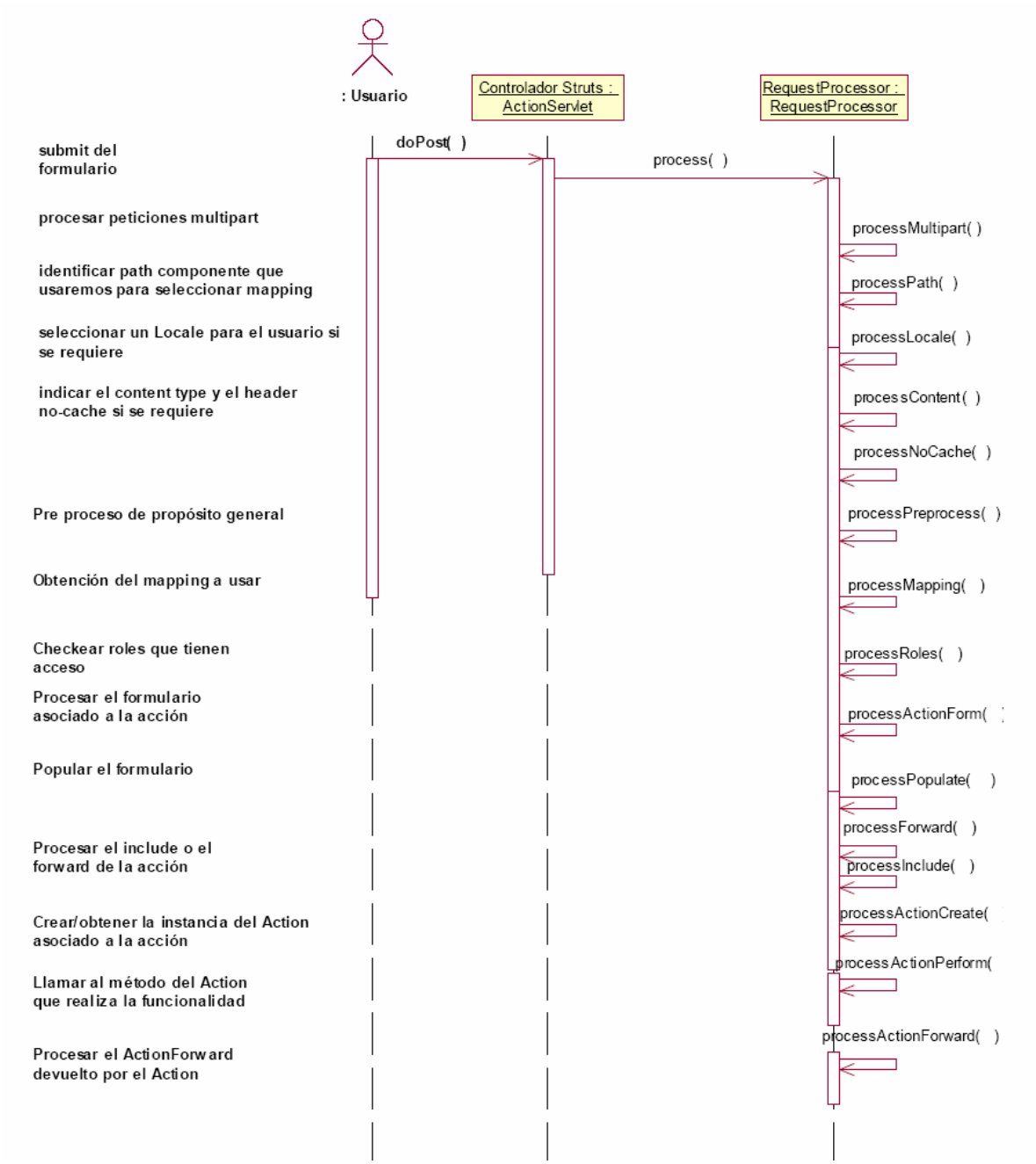


Figura 13-2.Diagrama de Secuencia RequestProcessor

Algunos de los métodos mostrados (*processRoles, processPreprocess, etc*) permiten definir aspectos como:

- Selección del Locale asociado al usuario (*processLocale*). El framework implementa este método para seleccionar el idioma del usuario.
- Desactivación de la entrada según el rol del usuario (*processRoles*). Este método permite hacer uso de los roles definidos por el estándar J2EE (con mapeo entre grupos LDAP y roles definidos de manera estática en la aplicación).En el framework, los roles se definen de manera dinámica (se permiten añadir nuevos roles) y no mediante la definición estática de roles J2EE, por lo que no se ha hecho uso de este método.
- Preproceso antes de ejecutar una acción (*processpreProcess*). Es en este método donde en el framework se realizan las comprobaciones de autentificación y autorización.

Para poder definir el comportamiento de estos métodos es necesario extender la clase de Struts RequestProcessor con el comportamiento que se desee. En el framework se ha creado la clase 'com.juoc.struts.base.BaseRequestProcessor'. Esta clase extiende de la clase 'org.apache.struts.tiles.TilesRequestProcessor', clase que hereda de RequestProcessor para hacer uso de pantallas.

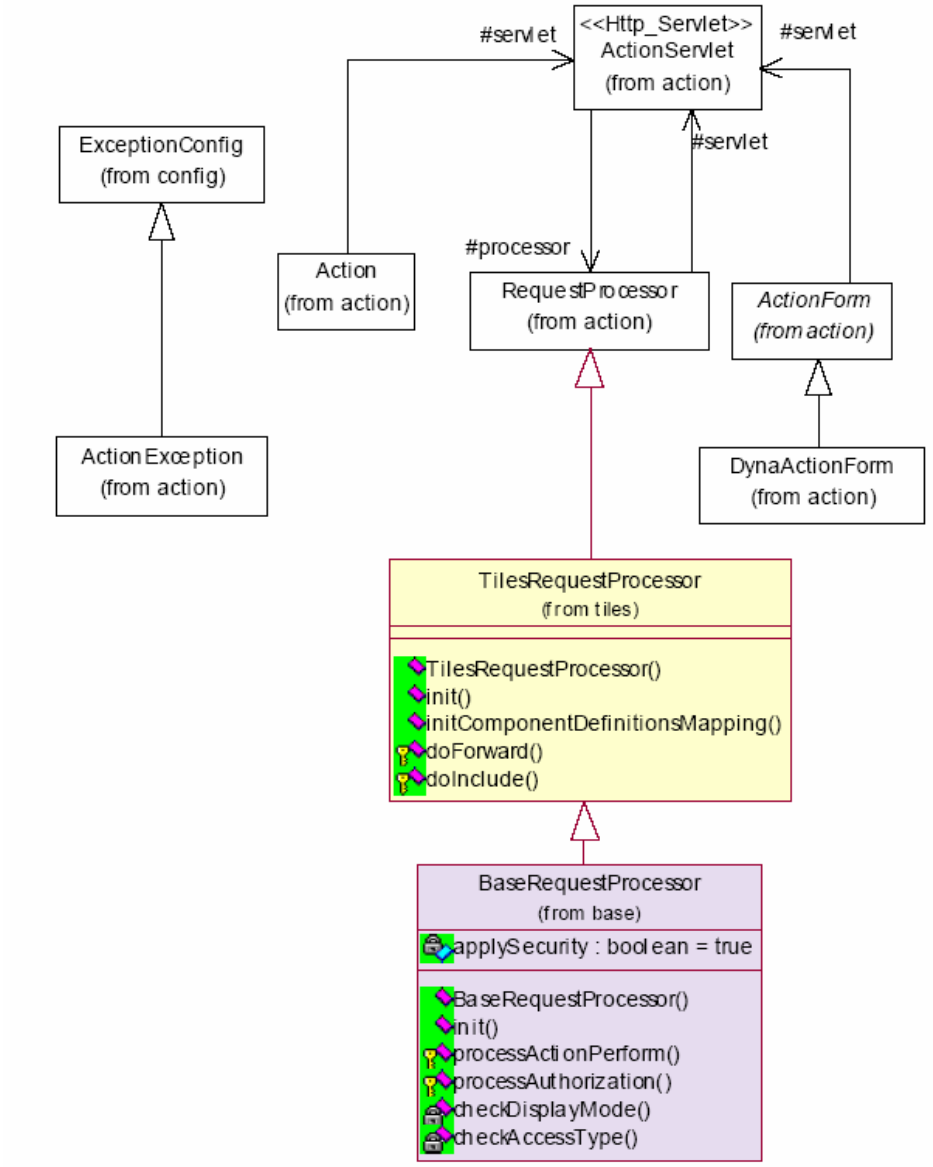


Figura 13-3. Jerarquía BaseRequestProcessor Arquitectura

En el fichero de configuración de struts debe indicarse como clase que funcionará como procesador de las peticiones a 'com.juoc.struts.base.BaseRequestProcessor':

```
<!-- ******************************************************************* -->
<!-- Processor encargado de tratar las peticiones recibidas -->
<!-- -->
<!-- ******************************************************************* -->
<controller processorClass="com.juoc.struts.base.BaseRequestProcessor">
</controller>
```

Código 13-3 Configuración Controlador Struts

En este tag controller se pueden definir entre otros atributos:

Parámetro	Descripción
bufferSize	Tamaño del buffer cuando se procesan uploads de ficheros
maxFileSize	Máximo tamaño en bytes que se acepta al recibir un upload. Se puede expresar el número seguido por K,M o G para indicar kilobytes, megabytes o gigabytes (por defecto son 250MB)
nocache	true para indicar nocache en el header
processorClass	Clase request processor
tempDir	Directorio temporal usado cuando se procesan uploads de ficheros

13.1.3 Plugins

Struts permite añadir puntos de extensión que afectan al framework. Hemos extendido el *RequestProcessor* para definir nuevo comportamiento (mediante la definición de los métodos preProcess,...). También existe la posibilidad de definir componentes que sean pluggados y cargados dinámicamente al inicializar la aplicación. Cualquier clase puede actuar como Plugin si implementa la interficie *Plugin*. Durante la inicialización de la aplicación, el *ActionServlet* llamará al método init de cada uno de los plugin que se hayan configurado en el fichero de configuración de struts. En el framework se usan actualmente los siguientes plugins:

• Plugin de Validator. Es el que se encarga de gestionar la validación dinámica de formularios
• Plugin de Tiles. Es el que define el uso de Tiles ('Servicio de Pantallas')
• Plugin de Configuración. Carga las variables de configuración para varios servicios: de seguridad.

La configuración de los plugins la realizamos en el fichero struts-config.xml:

```
<!-- ========== Plug Ins Configuration ================================= -->
<!-- Plugin para uso Tiles -->
<plug-in className="org.apache.struts.tiles.TilesPlugin">
<set-property property="definitions-config" value="/WEB-INF/tiles-defs.xm
<set-property property="definitions-debug" value="2" />
<set-property property="definitions-parser-details" value="2" />
<set-property property="definitions-parser-validate" value="true" />
</plug-in>
<!--
Add multiple validator resource files by setting the pathnames property
with a comma delimitted list of resource files to load.
-->
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
<set-property property="pathnames" value="/WEB-INF/validator-rules.xml,
/WEB-INF/validator.xml"/>
</plug-in>

<!-- ****************************************************************** -->
<!-- Plugin de carga de configuración general
<!-- -->
<!-- ****************************************************************** -->
<plug-in         className="com.juoc.struts.base.ConfPlugIn">         <set-property
property="filename"      value="Configuration.xml"      />      </plug-in>      <!--
****************************************************************** --> <!--
```

Código 13-4 Configuración de Plugins

13.1.4 Fichero de Recursos multiidioma

Para especificar cual será el fichero de recursos multiidioma (ver 'Servicio Multiidioma'), es necesario especificar en el fichero 'struts-config.xml', cual será el nombre de inicio del fichero para cada idioma y se indicará su localización a partir de WEB-INF/classes.

WEB-INF/classes
→ApplicationResources.properties
→ApplicationResources_ca.properties
→ApplicationResources_en.properties
…

Es recomendable que en desarrollo su ubicación se encuentre en el directorio raíz 'src' :

```
<message-resources parameter="ApplicationResources"/>
```

Código 13-5 Configuración Recurso Multiidioma

13.1.5 Action

La clase com.juoc.struts.base.BaseAction es la clase central del framework. Su propósito principal es actuar como puente entre una petición del cliente y una operación del negocio.
Cada clase *Action* puede ser diseñada para realizar una única tarea u operación del negocio, pero una sola tarea no significa que la clase acción pueda solo realizar una función del negocio. La clase *Action* debería agrupar toda la funcionalidad relativa a un requerimiento funcional. Así, funcionalidades relacionadas pueden ser gestionadas en el mismo action.

La clase *Action* contiene un método principal 'execute' con la siguiente signatura:

public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpSerlvetResponse response) throws IOException,ServletException

Este método es llamado directamente por el RequestProcessor cuando recibe la petición. Previamente a la llamada, el *RequestProcessor* ha preparado el objeto formulario ActionForm con los datos recibidos en la petición. Pero la gran pregunta es: ¿Cómo sabe el ActionServlet a qué Action corresponde la petición del usuario? La respuesta se encuentra en los *ActionMapping*

ActionMapping

Los *ActionMappings* se configuran en el fichero de configuración de struts (**struts-config.xml**). Cada <*action*> del fichero se representa en memoria por un objeto de tipo *org.apache.struts.action.ActionMapping*. El procesador (RequestProcessor), a partir de la url recibida, mira en el fichero 'struts-config.xml' cual es el Action (especificado en el atributo 'type') que corresponde a la url, y realiza una llamada a su método 'execute'. Un ActionMapping se define en el fichero 'struts-config.xml' mediante el tag <action> con los siguientes parámetros:

Parámetro	Descripción
path	Path relativo a la aplicación, comenzando con '/'
type	Nombre fully qualified de la clase Action que resuelve la petición, en caso de que no se indiquen los atributos *forward* o *include*
name	Nombre del bean formulario asociado a esta acción. Este valor debe haber sido declarado en el campo *name* de alguno de los formularios declarados en algún tag <*form-bean*> *(ver* 'Componentes Vista*')*

scope	Nivel scope donde se ubica el bean formulario. Valores posibles: request,session
input	Define cual es la página que se mostrará en caso de que haya excepción
validate	true indica que se llamara al método 'validate' del formulario antes de llamar al método 'execute' del *Action*
parameter	Parámetro de propósito general para usar desde el Action mediante *getParameter*. Su uso va intímamente ligado a los DispatchAction (ver siguiente apartado)

```
<action path="/enviarCorreoAceptacion"
type="com.juoc.test.vehiculos.EnviarCorreoAceptacionActio
name="correoForm"
scope="session"
input="pages.vehiculos.enviarCorreoAceptacion" validate="true"
>
<forward name="success" path="pages.vehiculos.enviarCorreoAceptacion" />
</action>
```

Código 13-6 Ejemplo de ActionMapping

En el ejemplo mostrado, cuando el controlador recibe un petición donde el path en la URI contiene la cadena '/enviarCorreoAceptacion.do', el controlador invoca al método 'execute' de la clase '*EnviarCorreoAceptacion*'

```
public ActionForward execute(ActionMapping mapping,
ActionForm form,
HttpServletRequest request,
HttpServletResponse response)
throws Exception {
...
return mapping.findForward("success");
}
```

Código 13-7 Definición del método execute

En el ejemplo, el método execute devuelve directamente la página que se haya definido en el forward con nombre 'success'. Si varias urls fueran gestionadas en el mismo Action, podríamos discernir cual es la petición recibida mediante: 'mapping.getPath()'. Este método nos retornaría la url que se ha utilizado para llegar al Action.

DispatchAction

Struts proporciona la capacidad de usar subclases de *Action* para realizar determinadas acciones. El más interesante es *org.apache.struts.action.DispatchAction,* que se ha extendido en el framework mediante la clase *com.juoc.struts.base.BaseDispatchAction*. Su propósito es permitir que múltiples operaciones que estarían en varias clases *Action* estén agrupadas en una única clase. En este caso, no se crea una función 'execute' única, se crean los métodos que se requieran, y mediante el campo '***parameter***' del mapping se define cual es el parámetro de la URL que definirá qué método llamar. Así, una url como esta:

mantVehiculo.do?reqCode=create

y como actionmapping el mostrado a continuación:

```
      <action  path="/mantVehiculo"  type="com.juoc.test.vehiculos.VehiculoAction"
      name="vehiculoForm" scope="session" input="pages.vehiculos.mantVehiculo"
   validate="true"
   parameter="reqCode"
   >
   <forward name="success" path="pages.vehiculos.mantVehiculo" />
   <forward name="print" path="pages.vehiculos.printVehiculo" />
</action>
```

Código 3-8 ActionMapping con DispatchAction

, provocará que se llame al método '**create**' de la clase *Action 'VehiculoAction'*.

```
/**
* Accion encargada de iniciar la creacion de un vehiculo
*/
public ActionForward create(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception {
HttpSession session = request.getSession();
if ("request".equals(mapping.getScope()))
request.removeAttribute(mapping.getAttribute());
else
session.removeAttribute(mapping.getAttribute());
return (mapping.findForward("success"));
}
```

En el Framework se han extendido respectivamente las clases de struts Action y DispatchAction para añadir comportamiento adicional. El desarrollo debe basarse sobre estas clases.

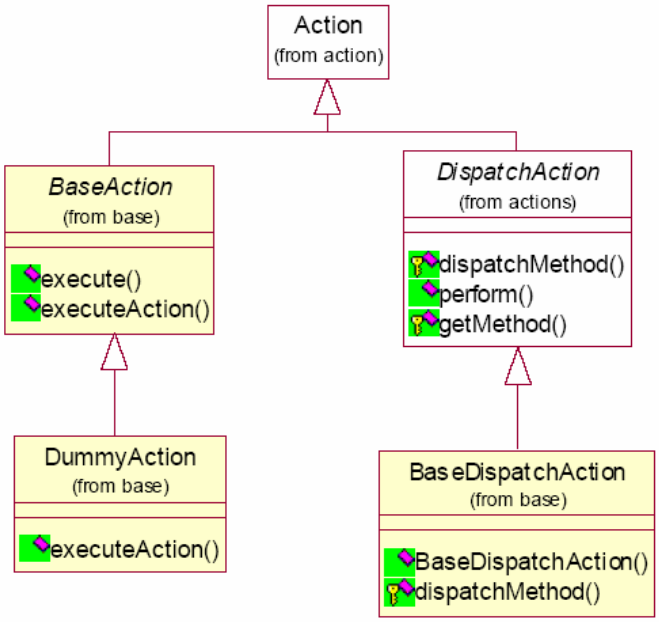


Figura 13-4. Clases Action del Framework

Así, las clases a utilizar son:
• com.juoc.base.struts.base.BaseAction, o bien
• com.juoc.base.struts.base.BaseDispatchAction

Adicionalmente, se proporcionan las clases:
• com.juoc.base.struts.base.DummyAction. Esta clase permite no tener que definir un Action para aquellas urls que únicamente quieren retornar una página y no realizar ninguna lógica.

```
<action path="/urlEjemplo"type="com.juoc.struts.base.DummyAction">
      <forward name="success" path="paginaResultado"> </forward>
</action>
```

Determinar la siguiente Vista

Una vez la clase *Action* ha resuelto la petición, decide cuál es la siguiente vista a mostrar. Para ello define una *org.apache.struts.action.ActionForward*.
Con el siguiente método obtiene del actionmapping el forward definido con nombre 'success':

```
return (mapping.findForward("success"));
```

El controlador, a partir del nombre devuelto, obtiene del fichero de configuración (struts-config.xml) cual es el path qué define la vista a mostrar.

```
    <action path="/mantVehiculo"
    …
    >
    <forward name="success" path="pages.vehiculos.mantVehiculo" />
</action>
```

Código 13- 9 Definición Retornos Vistas

En el ejemplo mostrado, el Action ha devuelto 'success' lo que implica que la siguiente vista a mostrar es 'pages.vehiculos.mantVehiculo'. En este caso se ha indicado un nombre de vista que corresponderá a una definición en el fichero '**tiles-defs.xml**' (ver
'Servicio de Pantallas'). Sin el uso de Tiles la referencia podría haber sido la de una página JSP directamente, por ejemplo:

```
    <action path="/mantVehiculo"
    …
    > <forward name="success" path="mantenimiento.jsp" />
    </action>
```

13.2 Componentes Modelo

Cuándo se habla de una arquitectura Struts, existe una divergencia entre definir donde está la frontera entre los componentes del modelo y los de controladores.
A efectos generales, en una arquitectura de 3 capas donde no se haga una separación clara entre la lógica de negocio y la arquitectura, existe la equivocada práctica de poner en la clase *Action* la lógica de negocio.
Si se sigue ésta filosofía la clase *Action* sería a la vez un componente Controlador y Modelo.
La buena práctica es que el Modelo se ubique en clases puras, es decir, clases Java que no dependan de la arquitectura de presentación utilizada.

13.3 Componentes Vista.

13.3.1 ActionForm

Los objetos *ActionForm* son usados en el framework para pasar los datos que vienen en la petición HTTP a objetos Java que puedan ser tratados desde los Action. Para separar la capa de negocio de la presentación (de manera que si cambiáramos de framework se mantuviera invariable la lógica de negocio) es conveniente que no se pasen objetos *ActionForm* a la capa de negocio, sino crear objetos ValueObject del negocio a partir de los formularios. Debido a que en el método *execute* (o el método definido en el caso de un DispatchAction) el argumento del formulario es de tipo *ActionForm* dentro del método deberemos hacer un casting al tipo de *ActionForm* que se haya definido.

De esta manera la forma de desarrollar es no usar ActionForms desde la lógica de negocio, pasar de ActionForm a Value Object mediante método BeanUtils.copyProperties y hacer cast del ActionForm al tipo necesario dentro del método execute del Action

El proceso general que se sigue en una petición en lo que se refiere a los formularios es:

1) El controlador (RequestProcessor) obtiene el mapping asociado a la petición

2) Accede al valor indicado en el parámetro *name* y obtiene su *scope*

3) El controlador intenta encontrar una instancia de formulario con el nombre *name* en el scope indicado

```
<action path="/mantVehiculo"
```

```
        type="com.juoc.test.vehiculos.VehiculoAction"
        name="vehiculoForm" scope="session"
        input="pages.vehiculos.mantVehiculo" validate="true" parameter="reqCode" >
        <forward name="success" path="pages.vehiculos.mantVehiculo" />
</action>
```

4) Si encuentra la instancia, la reutiliza, si no, crea una instancia nueva del objeto de tipo indicado en el parámetro *type* del formulario correspondiente al nombre *name* que se encuentra en la definición de formularios (<form-bean>)

```
<form-bean name="vehiculoForm"
type="com.juoc.test.vehiculos.VehiculoActionForm"
dynamic="false">
</form-bean>
```

5) El controlador llama al método *reset* del formulario

En este método definiremos la inicialización de las variables que entre requests deben poner su valor por defecto.

6) El controlador itera cada uno de los parámetros recibidos por el request. Para cada uno de ellos, si existe el método set de la clase formulario, lo llama pasándole como valor el correspondiente al parámetro

7) Por último llama al método *validate* del formulario si se ha definido en el mapping del fichero de configuración struts-config.xml que validate=true

El hecho de utilizar un action form implica definir una clase con varios atributos, donde el nombre de cada uno de ellos corresponde con el nombre de los parámetros del request (los nombres de los campos de los formularios si realizamos un submit del formulario).

Estos atributos del formulario solo pueden definirse como String (ya que en la url todo llega como cadenas) o bien como boolean si su componente asociado es un checkbox.

```
public class VehiculoActionForm extends BaseActionForm
implements java.io.Serializable {
private String matricula;
private String fechaFinFabricacionEsperada;
private String kmTotales;
private String precioBase;
private String precioReal;
private boolean tieneDanyos=false;
public VehiculoActionForm() {
}
public String getMatricula() {
return matricula;
}
public void setMatricula(String matricula) {
this.matricula = matricula;
}
public String getFechaFinFabricacionEsperada() {
return fechaFinFabricacionEsperada;
}
public void setFechaFinFabricacionEsperada(String fechaFinFabricacionEsper
{
this.fechaFinFabricacionEsperada = fechaFinFabricacionEsperada;
}
public String getKmTotales()
{
return kmTotales;
}
public void setKmTotales(String kmTotales)
{
this.kmTotales = kmTotales;
}
public String getPrecioBase()
```

```
    {
    return precioBase;
    }
    public void setPrecioBase(String precioBase)
    {
    this.precioBase = precioBase;
    }
    public void reset(ActionMapping mapping, HttpServletRequest request)
    {
    tieneDanyos = false;
}
```

Código 13-10 Definición de un Formulario

En el método reset se ha definido la inicialización del atributo 'tieneDanyos'. La razón es que si previamente el valor es 'true' y el usuario desmarca en la pantalla el checkbox, por protocolo http no se enviará el parámetro correspondiente al atributo, por lo que no se llamará al set, y el valor continuará siendo el anterior ('true'). Para evitar este comportamiento, el controlador siempre llama a reset previamente a popular los atributos a partir de los parámetros, de forma que tieneDanyos tendría el valor correcto.

En el Action, debemos hacer un cast del parámetro 'form' a la clase esperada:

```
    public ActionForward create(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) throws Exception {
    VehiculoForm vehiculoForm = (VehiculoForm)form;
```

Cómo puede verse, en determinados casos, puede parecer costoso el hecho de definir estos formularios para cada formulario de la aplicación. En determinados casos, cuando no necesitamos definir el método reset, o realizar validaciones se pueden usar formularios dinámicos.

Formularios Dinámicos

Las propiedades de un formulario dinámico (definido por la clase *DynaActionForm* o *DynaValidatorActionForm* si queremos realizar la validación automática mediante Validator) se configuran en el fichero de configuración de Struts. Para que Struts considere que un formulario es dinámico debemos indicar el parámetro *dynamic* como true. Para cada propiedad que queramos definimos un *form-property* donde indicaremos el nombre del atributo y como *type* String o Boolean.

```
<form-bean                                                    name="vehiculoForm"
type="org.apache.struts.action.DynaValidatorActionForm" dynamic="true">
      <form-property name=”matricula” type=”java.lang.String” />
      <form-property name=”tieneDanyos” type=”java.lang.Boolean” />
</form-bean>
```

Código 13-11 Definición de un Formulario Dinámico

Desde el *Action*, para obtener los valores del formulario, no se puede hacer directamente un cast como se hacía cuando usábamos formularios no dinámicos, ya que no existe una clase explícita. Obtenemos los valores con métodos de *org.apache.commons.beanutils.PropertyUtils*. Dependiendo del tipo de propiedad usaremos un método u otro.

A continuación se muestran las propiedades y el método a usar de PropertyUt

Simple (nombre): Propiedad individual del bean: getSimpleProperty(nombreBean, nombreAtributo)
Nested (nombre1.nombre2.nombre3): El primer atributo se obtiene con get, y a partir de éste se obtiene el valor de nombre2 y así consecutivamente: getNestedProperty(bean,atributo) Ejemplo: getNestedProperty(objeto,"nombre1.nombre2.nombre3")

Indexed (nombre[indice]) : getIndexedProperty(bean,atributo)

No es recomendable el uso de este tipo de formularios, pues su control por parte del Framework es menor que en el caso de formularios estáticos.

Validación Formularios

La validación de los datos del formulario se puede realizar desde el método '**validate**' del formulario (si este es estático y no dinámico).
El hecho de usar este método tiene sus peros, como puede ser el hecho de tener que realizar validaciones de formato que pueden resultar tediosas. El uso del 'Servicio de Validación' (basado en Validator) permite que no sea necesario especificar el método 'validate' para este tipo de validaciones y que la validación se defina en un fichero XML.

```
    <form name="vehiculoForm">
    <field property="matricula" depends="required">
    <arg0 key="field.vehiculo.matricula"/>
    </field>
    <field property="fechaFinFabricacionEsperada" depends="date">
    <arg0 key="field.vehiculo.fechaFinFabricacionEsperada"/>
    <var>
    <var-name>datePatternStrict</var-name>
    <var-value>dd/MM/yyyy</var-value>
    </var>
</field>
```

Código 13-12 Validación Formulario

Para poder usar la validación es necesario que las clases extiendan la clase com.juoc.struts.base.BaseActionForm.

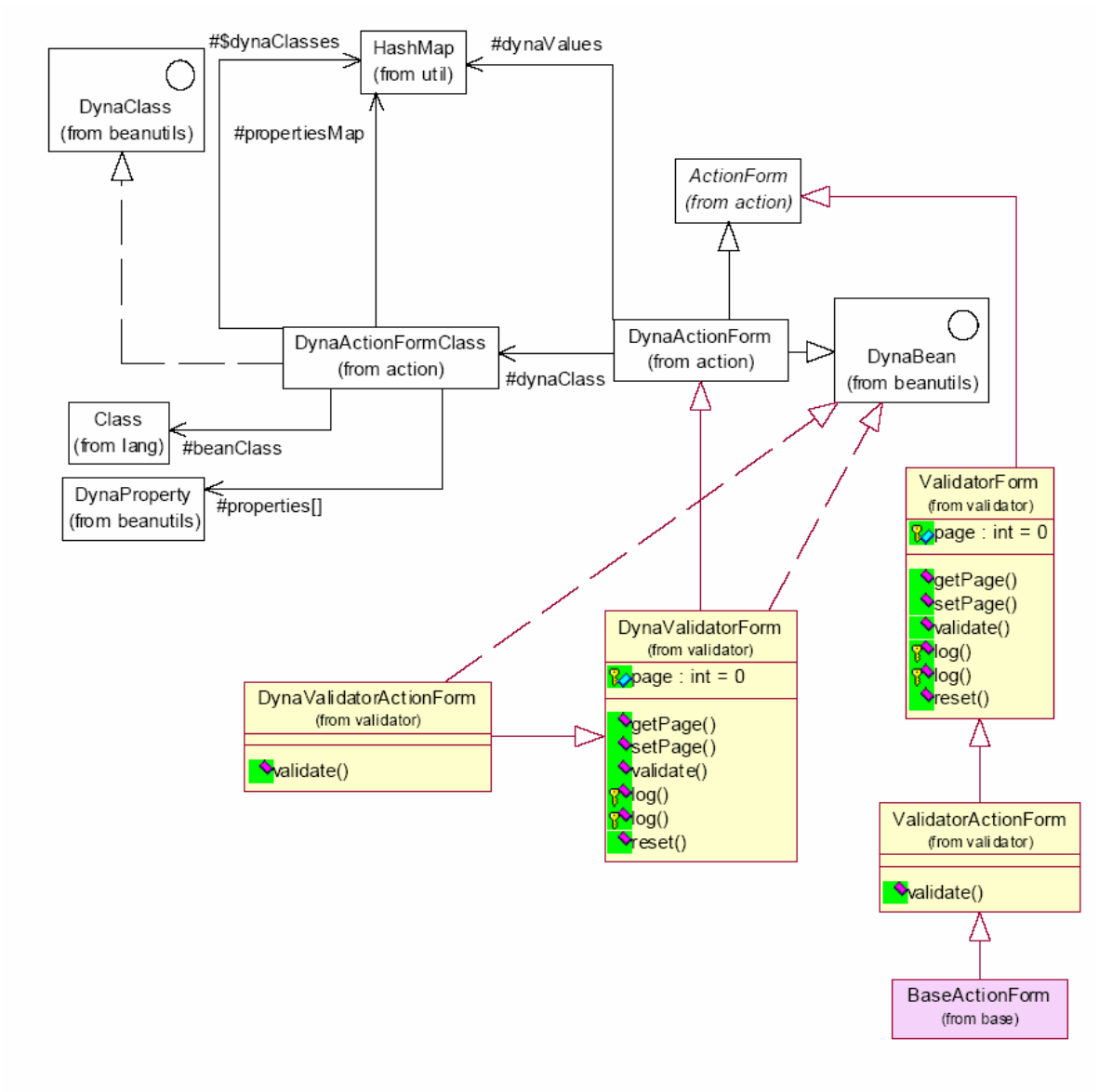


Figura 13-5 Diagrama de clases Formulario

13.3.2 Paso de Componentes Vista a Objetos del Negocio

En puntos anteriores se ha indicado que nunca deberían llegar a las clases de la lógica del negocio objetos de tipo ActionForm, porque si lo hiciéramos estaríamos sujetos al framework usado en la capa de presentación. Para ello, debemos usar mecanismos que permitan a partir de un objeto con unos atributos determinados copiar sus valores en otro objeto en el que quizás los tipos de los atributos no sean coincidentes con los del objeto origen. En el framework, se proporciona una clase 'com.juoc.util.beanutils.BeanUtils' que permite realizar esta copia mediante el método con la siguiente signatura:

```
public void copyProperties(Object dest, Object orig) throws
IllegalAccessException, InvocationTargetException
```

Para cada atributo del objeto origen realiza la llamada al método get y obtiene su valor. A continuación mira si en el objeto destino existe un método set para el mismo atributo. Si existe, realiza la conversión de tipos entre el atributo del objeto origen y el atributo del objeto destino.
Supongamos como punto de partida, el siguiente diagrama en el que muestra un formulario ('VehiculoForm') y su correspondiente objeto del negocio ('VehiculoVO'):

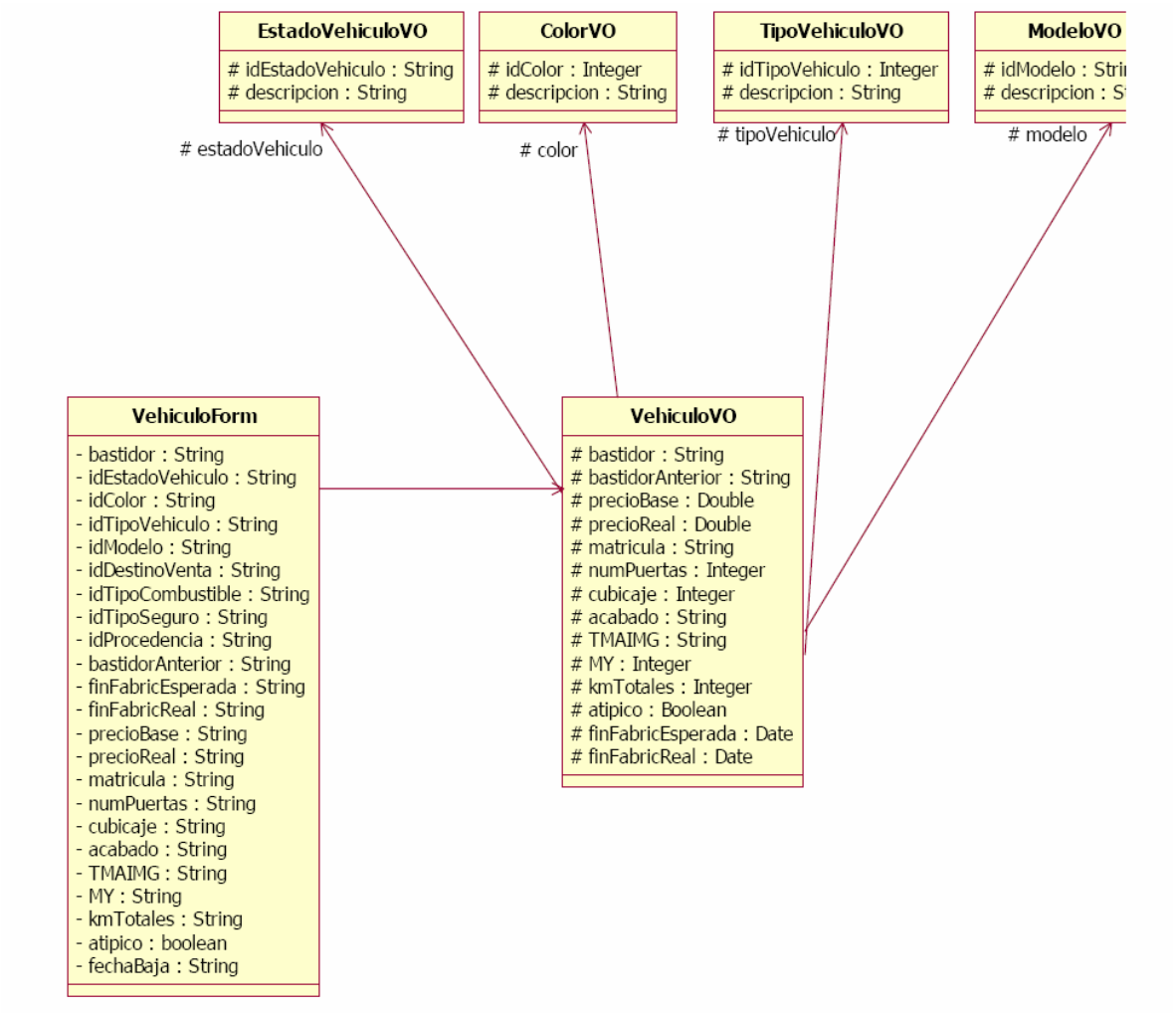


Figura 13-6. Clase Formulario y Clase ValueObjec

Imaginemos que se pretende realizar una copia de propiedades de un objeto origen instancia de ‘VehiculoForm’ a un objeto destino instancia de ‘VehiculoVO’. En el momento en el que encuentra el atributo ‘numPuertas’ en el objeto origen, obtiene su valor, de tipo ‘String’, y realiza la conversión necesaria a Integer para realizar la llamada a setNumPuertas en el objeto destino.
En esta clase hay una restricción importante dependiendo de la capa en la que queramos hacer uso de ella. Si queremos copiar un formulario a un objeto del negocio, debemos tener en cuenta el multiidioma. Si la fecha de entrada tiene un formato ‘dd/mm/yyyy’ para un usuario, y para otro usuario tiene un formato ‘mm/dd/yyyy’ el método no conoce cómo realizar la conversión a un Date.
Para hacer uso de multiidioma se define la clase ‘com.juoc.util.beanutils.LocaleBeanUtils’ que proporciona el mismo método ‘copyProperties’ con la salvedad de que tiene en consideración el formato de multiidioma que se está usando, definido por los parámetros del fichero de recursos referentes a tipos.
Esta clase hace uso de ‘com.juoc.util.beanutils.LocaleConverter’, clase utilidad que permite convertir de un string a objeto según los formatos del fichero de recursos (*convertString2Object)*, y convertir de objeto a string (*convert2String).*

```
#################################################################
# Formatos
#################################################################
org.apache.struts.taglib.bean.format.sql.timestamp = dd/MM/yyyy HH:
org.apache.struts.taglib.bean.format.sql.date = dd/MM/yyyy
org.apache.struts.taglib.bean.format.sql.time = dd/MM/yyyy
org.apache.struts.taglib.bean.format.date = dd/MM/yyyy
org.apache.struts.taglib.bean.format.int = #
org.apache.struts.taglib.bean.format.float = #.##
org.apache.struts.taglib.bean.format.double = #.##
org.apache.struts.taglib.bean.format.calendar = dd/MM/yyyy HH:mm
org.apache.struts.taglib.bean.format.time = dd/MM/yyyy HH:mm
```

Código 13-13 Definición de formatos de tipos según idioma

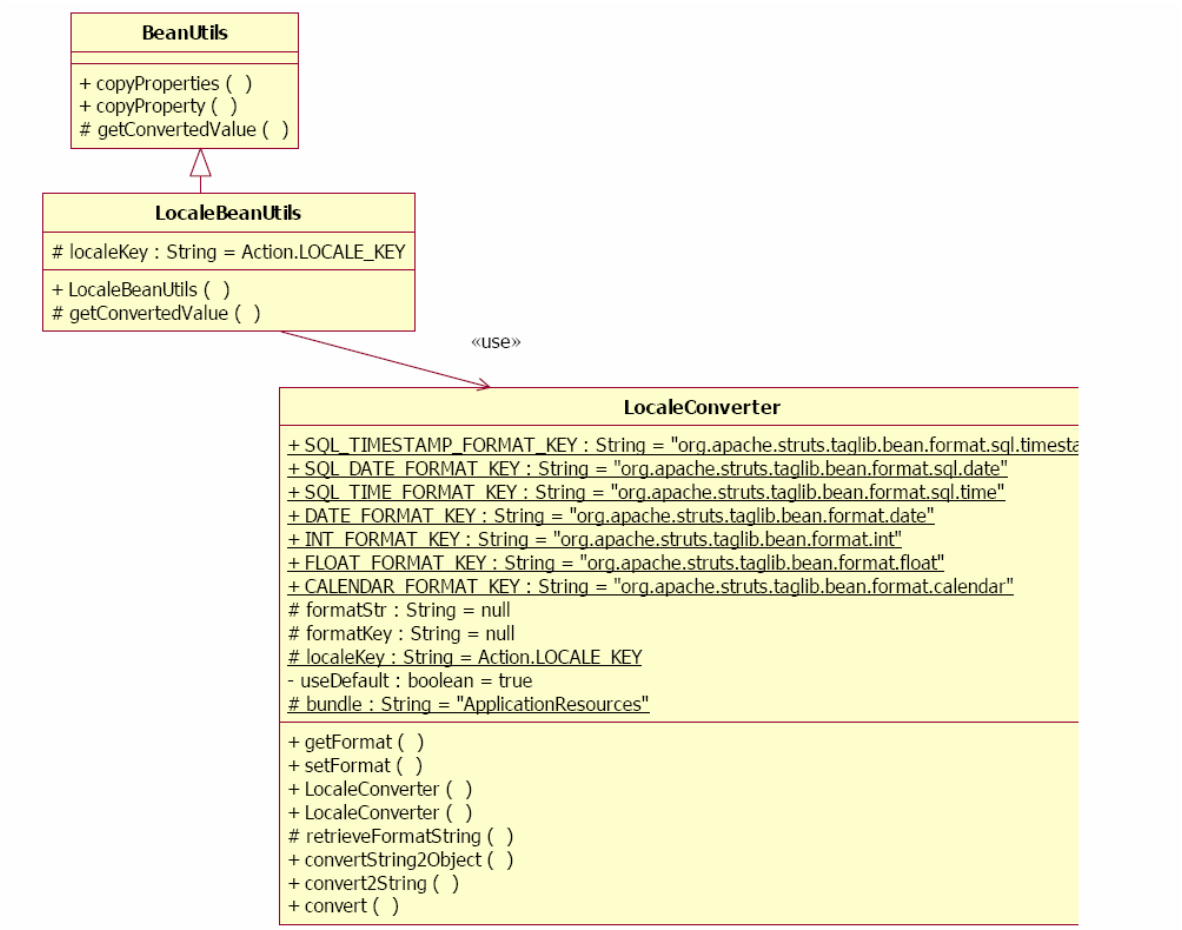


Figura 13-7. Diagrama de Clases BeanUtils

Según lo expuesto, el funcionamiento de las clases Action debería ser el mostrado a continuación:

```
/**
* Accion encargada de crear un vehiculo
*/
public ActionForward saveNew(ActionMapping mapping, ActionForm form,
HttpServletReqHttpServletResponse response) throws java.lang.Exception {
try {

   VehiculoVO tmpVehiculo = new VehiculoVO();
   // Copiar datos del formulario al vo
   LocaleBeanUtils beanUtils = new LocaleBeanUtils(request);
   beanUtils.copyProperties(tmpVehiculo,form);
   VehiculoBO vehiculoBO = new VehiculoBO();
   vehiculoBO.createVehiculo(tmpVehiculo);
   // Forward si todo ha ido bien a la edicion del vehiculo:
   return (new ActionForward("/mantVehiculo.do?reqCode=edit&bastidor=" +
tmpVehiculo.getBastidor()));

   initCombos(mapping,form,request);
}
}
/**
* Accion encargada de editar un vehiculo
*/
public ActionForward edit(ActionMapping mapping, ActionForm form,
```

```
    HttpServletRequest request, HttpServletResponse response) throws Exception {
    try {
    VehiculoBO vehiculoBO = new VehiculoBO();
    LocaleBeanUtils beanUtils = new LocaleBeanUtils(request);
    VehiculoVO voEdit = new VehiculoVO();
    beanUtils.copyProperties(voEdit,form);
    VehiculoVO vo = vehiculoBO.loadVehiculo(voEdit);
    // Copiar datos de retorno al formulario, del VO al formulario
    beanUtils.copyProperties(form,vo);
    return (mapping.findForward("success"));
    } finally {
    initCombos(mapping,form,request);
    }
}
```

Código 13-14 Uso de LocaleBeanUtils en las clases Action

En el ejemplo, en el que se muestra un ejemplo de inserción a la base de datos, inicializamos la clase LocaleBeanUtils con el request para que pueda acceder al fichero de recursos. Posteriormente llamamos a copyProperties para que el objeto formulario se copie en el objeto del negocio realizando la conversión de tipos.
En el caso de una consulta, como puede verse en el método ‘edit’, copiamos el objeto del negocio recuperado al formulario a presentar.

Capítulo 14.

14. Servicio de Pantallas

En toda aplicación Web se necesita un Look&Feel (L&F) común. Si el L&F se "*hardcodea"* en todas las páginas, un cambio de L&F se convierte en una auténtica pesadilla, pues deberíamos modificar todas las páginas de la aplicación.
En un buen diseño es importante que consigamos una separación clara entre el L&F y el contenido de las páginas.
El uso de plantillas nos permite definir un L&F común, donde podemos especificar que usaremos una cabecera, un cuerpo, un menú a la izquierda y un pie. La página final se construye pasándole como parámetro qué cabecera, qué cuerpo y qué menú corresponde a la pantalla a mostrar como respuesta a la petición.

14.1 Introducción

14.1.1 Mecanismos JSP Estándar

Con las JSP tenemos 2 maneras diferentes de incluir contenido de otras páginas: estática y dinámica.

- <%@ include file="..." %> (Estático)

  Si usamos <%@ include file="..." %>, se incluye el contenido de la página destino en tiempo de compilación y no de ejecución.
  De esta forma, el texto de la página incluída se inserta en la del padre. Si la página incluída es una página JSP (podría ser un HTML, ...), se evalúa y su contenido se inserta en la página padre. Después, la página padre continúa traduciendo la siguiente línea.
- <jsp:include page="..." /> (Dinámico)

  Si usamos <jsp:include page="..." /> el recurso es tratado dinámicamente, de manera que en tiempo de ejecución se evalúa la página y se vuelca el resultado.

14.1.2 Template

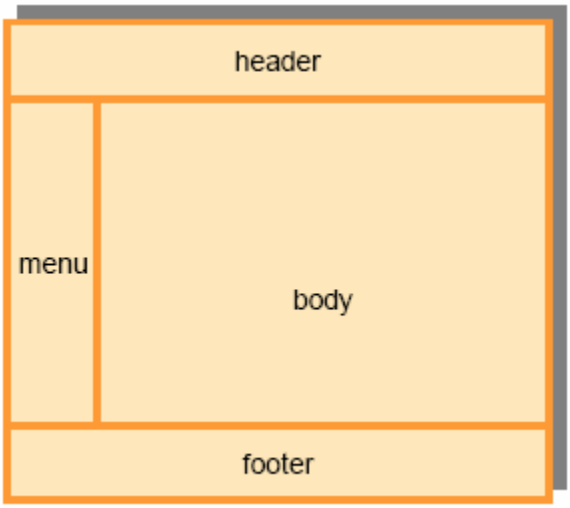
Un template no es más que una página JSP que usa una librería de tags JSP para definir el layout de una página. El template sirve como definición de cómo se ubicarán las páginas sin especificar el contenido. Éste se inserta en tiempo de ejecución en la página template. Todas las páginas de la aplicación pueden reutilizar el mismo template o bien usar otros.
Un ejemplo (*basicLayout.jsp*) de template sería:

```
<table border="0" width="100%" cellspacing="5">
    <tr>
    <td colspan="2"><tiles:insert attribute="header" /></td>
    </tr>
    <tr>
    <td valign="top">
    <tiles:insert attribute='menu'/>
    </td>
    <td valign="top" align="left">
    <table width="100%">
    <tr><td>
    <tiles:insert attribute='body' />
    </td></tr>
    </table>
    </td>
    </tr>
    <tr>
    <td colspan="2">
```

```
        <tiles:insert attribute="footer" />
        </td>
        </tr>
    </table>
```

Código 14-1 Ejemplo de Plantilla de Presentación

En este ejemplo, se entremezclan código HTML y tags propios JSP (como es el caso de tiles:insert). En el tag <tiles:insert> se está indicando de manera declaratoria que el contenido a mostrar en esa parte debe crearse de manera dinámica a partir del valor indicado en el parámetro 'attribute'. La gran ventaja de esta definición es que varias páginas de la aplicación pueden usar el mismo layout que está controlado por este fichero. Si en algún momento de la vida del proyecto se decide cambiar el layout, solo es necesario retocar esta página única y no todas las páginas de la aplicación.

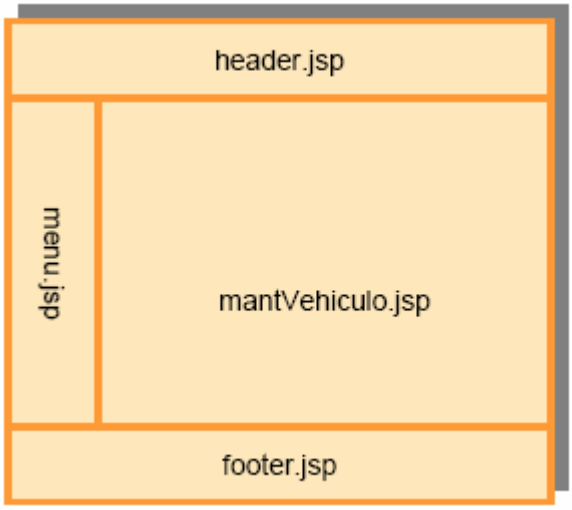


La pieza que encaja todo este mecanismo es cómo el contenido de las secciones 'header', 'menu', 'body', y 'footer' se insertan conjuntamente para formar la página final. Para ello, existe el fichero de definiciones de páginas (*tiles-defs.xml* por defecto).

```
<definition name="pages.vehiculos.mantVehiculo" path="/layouts/basicLayout.jsp">
       <put name="title" value="Mantenimiento Vehiculos" />
       <put name="header" value="/template/header.jsp" />
       <put name="menu" value="/template/menu.jsp" />
       <put name="footer" value="/template/footer.jsp" />
       <put name="body" value="/mantVehiculo.jsp" />
</definition>
```

Código 14-2 Ejemplo de Definición de una Pantalla

Con la definición arriba mostrada estamos creando una nueva página definida por el layout de nombre '*basicLayout.jsp*' , donde el contenido de cada una de las partes del layout está definida con cada uno de los tags put. De esta forma, se vuelca (evaluando por defecto dinámicamente) el contenido de cada uno de las páginas definidas en cada uno de los parámetros en el layout.



14.1.3 Construcción Pantalla a devolver

Así, el funcionamiento en el framework es el explicado a continuación:

1) La petición del usuario es recibida por Struts (/mantVehiculo.do...)
2) Struts obtiene del fichero 'struts-config.xml' la relación entre la url recibida y el Action asociado y redirecciona la petición a este Action .

```
<action path="/mantVehiculo"
type="com.juoc.test.vehiculos.VehiculoAction"
name="vehiculoForm"
scope="session"
input="pages.vehiculos.mantVehiculo"
validate="true"
parameter="reqCode"
>
<forward name="success" path="pages.vehiculos.mantVehiculo" />
</action>
```

3) Una vez el Action finalice el proceso, el Action devuelve cual es la siguiente pantalla a mostrar

```
public ActionForward edit(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception {
...
return (mapping.findForward("success"));
}
```

4) Struts recupera cual es el path de la página a devolver según el forward devuelto por el Action (en el ejemplo, el Action ha devuelto 'success' que en el mapping corresponde al path 'pages.vehiculos.mantVehiculo').
5) Debido a que el path se trata de una definición y no de un fichero .jsp directo, Tiles obtiene del fichero 'tiles-defs.xml' la definición de la pantalla. En esta definición aparece qué plantilla usa y define los valores parametrizados de los componentes definidos por la plantilla.

```
<definition name="pages.vehiculos.mantVehiculo" path="/layouts/basicLayout.jsp">
<put name="title" value="Mantenimiento Vehiculos" />
<put name="header" value="/template/header.jsp" />
<put name="menu" value="/template/menu.jsp" />
<put name="footer" value="/template/footer.jsp" />
<put name="body" value="/mantVehiculo.jsp" />
</definition>
```

6) Tiles construye la página dinámicamente y ésta es devuelta al usuario

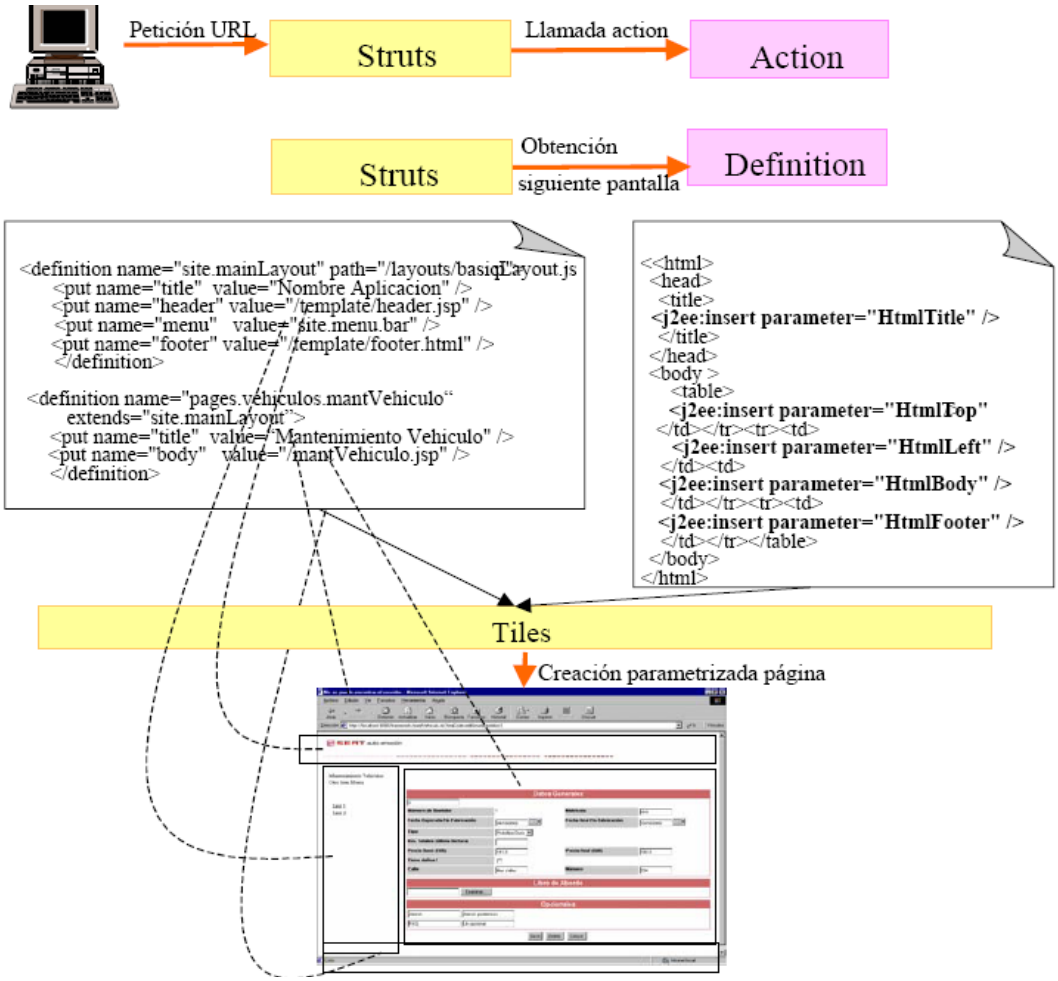


Figura 14-1. Proceso General de Creación de Pantallas

14.2 Instalación y Configuración

14.2.1 Uso de la librería

Para el uso de la librería de tags que proporciona Tiles deberemos tener en /WEB-INF el fichero 'struts-tiles.tld'.

En el fichero de deploy '/WEB-INF/web.xml' es necesario definir el uso de la librería:

```
<taglib>
   <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
</taglib>
```

Código 14-3 Uso de la Librería de tags para Pantallas

14.2.2 Uso de Tiles con Struts

Para poder hacer uso de Tiles es necesario configurar en el fichero de configuración de Struts el uso del plugin de Tiles:

```
<!-- Plugin para uso Tiles -->
<plug-in className="org.apache.struts.tiles.TilesPlugin">
   <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml" />
   <set-property property="definitions-debug" value="2" />
   <set-property property="definitions-parser-details" value="2" />
   <set-property property="definitions-parser-validate" value="true" />
</plug-in>
```

Código 14-4 Configuración del plugin Tiles

El parámetro '*definitions-config*' define cual será el fichero de definiciones (ver siguiente apartado). Si queremos usar más de un fichero de definiciones (para por ejemplo dividir el aplicativo en partes diferenciadas), indicaremos cada uno de los ficheros separados por una ','.

14.3 Definiciones

Las definiciones permiten especificar los atributos requeridos por una pantalla o Tile en una estructura reutilizable.
Estas definiciones se especifican en un fichero XML (aunque Tiles permite hacerlo en ficheros JSP esta parte está fuera del alcance de nuestro framework, por su poco uso y mal diseño).

```
<tiles-definitions>
<definition name="nombre" path="path">
<put name="nombreParam" value="valor" />
…
<putList name="nombreList">
<add value="valor1"/>
<add value="valor2"/>
</putList>
</definition>
```

Por defecto, una definición se especifica como se muestra arriba:

- name. Nombre de la pantalla. Este nombre debe ser único
- path. Ruta a la página JSP que construye la pantalla (la plantilla)
- put. Con cada parámetro a pasar a la definición definiremos una entrada put
- putList. Sirve para poder pasar una parámetro que esté compuesto por una lista de valores

14.3.1 Extender Definiciones

Una definición puede especificarse de manera que extienda otra definición. Al extender esta definición se heredan todas las propiedades de la definición padre.
La extensión de definiciones es básica para definir un layout común a toda la aplicación mediante una definición, para que todas las demás definiciones hereden esta definición redefiniendo solo aquellas partes que varíen entre una pantalla general y una específica.

```
<definition name="site.mainLayout" path="/layouts/basicLayout.jsp">
<put name="title" value="Nombre Aplicacion" />
<put name="header" value="/template/header.jsp" />
<put name="menu" value="site.menu.bar" />
<put name="footer" value="/template/footer.jsp" />
</definition>
...
<definition name="pages.vehiculos.mantVehiculo" extends="site.mainLayout" >
<put name="title" value="Mantenimiento Vehiculos" />
<put name="body" value="/mantVehiculo.jsp" />
</definition>
```

Código 14-5 Ejemplo de Extensión de una Definición

En el caso mostrado, se ha especificado una definición 'site.mainLayout' que usa el layout definido por la página 'basicLayout.jsp'. Esta definición ha sido extendida por 'pages.vehiculo.mantVehiculo' especificando únicamente cual será el título y el cuerpo del layout, mientras que mantiene invariable la cabecera, menú y pie del padre.
Si nos fijamos en la línea <put name=menu>, también podremos ver que una región se puede definir con otra definición en lugar de una página JSP.

14.3.2 Nombre de Definición en el Forward de los Action

Con Struts, la manera de especificar el retorno de la página a mostrar se indica como un forward. La vista a la que hacer forward se puede expresar con un nombre lógico y la URL de la página JSP asociada.

```
<action path="/mantVehiculo"
   type="com.juoc.test.vehiculos.VehiculoAction"
   name="vehiculoForm"
   scope="session"
   input="pages.vehiculos.mantDictamenVehiculo"
   validate="true"
   parameter="reqCode" >
   <forward name="success" path="/ejemplo.jsp" />
</action>
```

Si usamos Tiles, y ese es nuestro caso, podemos en lugar de indicar como path una página JSP, usar el nombre de una definición. En el framework, como norma general usaremos definiciones en lugar de páginas JSP.

```
   <action path="/mantVehiculo"
   type="com.juoc.test.vehiculos.VehiculoAction"
   name="vehiculoForm"
   scope="session"
   input="pages.vehiculos.mantDictamenVehiculo"
   validate="true"
   parameter="reqCode"
   >
   <forward name="success" path="pages.vehiculos.mantVehiculo" />
   </action>
```

14.4 Layouts del Framework

En el Framework se han definido varios layouts, pero también se pueden definir nuevos si se requiere.
Los layouts actualmente disponibles (disponibles en la carpeta '***layouts***') son:

- Layout Básico (*basicLayout.jsp*)
  Este layout es el básico de la aplicación, y permite definir pantallas en las que tenemos un título, una cabecera, una parte izquierda, el cuerpo y el pie

- Layout Frames (*framesLayout.jsp*)
  Layout que define un layout básico tal y como el anterior con la diferencia que cada una de las partes es un frame.

14.4.1 Layout Básico

Este layout proporciona la posibilidad de definir pantallas con una cabecera, menú, un cuerpo y un pie.
Los atributos requeridos para este layout son:

- title. Título a mostrar en el navegador
- header. URL o nombre de definición a usar para mostrar la cabecera
- menu. URL o nombre de definición a usar en la parte menú

- body. URL o nombre de definición usada para mostrar el cuerpo o contenido
- footer. URL o nombre de definición a usar para mostrar el pie

El uso típico de este layout es:

```
<definition name="site.mainLayout" path="/layouts/basicLayout.jsp">
<put name="title" value="Nombre Aplicacion" />
<put name="header" value="/template/header.jsp" />
<put name="menu" value="site.menu.bar" />
<put name="footer" value="/template/footer.jsp" />
</definition>
```

Y usaríamos el mecanismo de extensión para definir el cuerpo en cada una de las pantallas de la aplicación.
Una posible implementación de la plantilla, disponible en la distribución del framework, es:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-layout.tld" prefix="layout" %>
<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>
<%@ page errorPage="errorpage.jsp" %>
<% String openerForm = (String)pageContext.getRequest().getParameter("openerForm
boolean windowPopup = (openerForm!=null) && (!openerForm.trim().equals(""));
%>
<%-- Tenplate Basico
Template en el que sea crea una cabecera, un menu a la izquierda, el
cuerpo y un pie
@param title Titulo de la pagina
@param header Cabecera
@param menu Menu
@param body Body
@param footer Pie
--%>
<fwk:html>
<table border="0" width="770" cellspacing="0" cellpadding="0">
<% if (!windowPopup) { %>
<tr>
<td colspan="3"><tiles:insert attribute="header"/></td>
</tr>
<% } %>
<tr>
<td background="../template/images/fons_lateral.gif">  </td>
<% if (!windowPopup) { %>
<td valign="top" width="200">
<table width="100%" border="0" cellspacing="0" cellpadding="0">
<tr>
<td><html:img pageKey="images.template.pixel" width="4" height="4"/></t
</tr>
<tr>
<td><tiles:insert attribute='menu'/></td>
</tr>
</table>
</td>
<% } %>
<td valign="top" align="left">
<table width="100%">
<tr>
<td><html:errors/></td>
</tr>
<tr>
<td><tiles:insert attribute='body'/></td>
</tr>
</table>
</td>
</tr>
<% if (!windowPopup) { %>
<tr>
<td colspan="3"><tiles:insert attribute="footer" /></td>
</tr>
<% } %>
</table>
</fwk:html>
```

Código 14-6 Ejemplo de Plantilla Básica

El uso de windowPopup se ha incorporado para que cuando por el mecanismo de selección de valor con una nueva ventana (tag TextField) se abra una nueva ventana, no se presenten ni cabecera, ni menú, ni pie

Este layout, además de la estructura, hace uso de los tags <fwk:html> y <html:errors>. Con el primero está generando todas las referencias a ficheros de estilo, ficheros javascript, código específico para el uso de tags, etc. El segundo tag sirve para mostrar los errores resultado de las excepciones lanzadas en la aplicación.

14.4.2 Layout Frames

Los atributos requeridos son los mismos que los definidos para el Layout Básico, con la salvedad que ahora, debido al uso de frames, no es posible especificar directamente que las partes sean definiciones (pues cada frame es una petición que realiza el navegador), sino que debemos especificar URLs de páginas JSP. Ahora bien, esta página JSP es la que puede hacer uso de una definición para crearse a sí misma.
Así, en la definición siguiente, el menú se ha definido por una página jsp denominada 'menu.jsp' que indirectamente incorporar la definición que se encuentre en 'site.menu.bar' y la construye de manera dinámica.

```
<definition name="site.mainLayout" path="/layouts/framesLayout.jsp">
<put name="title" value="Titulo" />
<put name="header" value="/template/header.jsp" />
<put name="menu" value="/template/menu.jsp" />
<put name="footer" value="/template/footer.jsp" />
</definition>
```

Código 4-7 Definición de Ejemplo con Frames

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<fwk:html>
   <tiles:insert definition="site.menu.bar" flush="true" />
</fwk:html>
```

Código 14-8 Inserción de una Definición cuando se hace uso de frames

Con este código estamos consiguiendo el comportamiento equivalente tal y como se había mostrado en el ejemplo de layout básico.
La plantilla de las definiciones para frames se encuentra en la distribución del framework (bajo el directorio '/layouts') implementada de la siguiente forma:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-layout.tld" prefix="layout" %>
<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>
<%-- Frameset Layout component
parameters : title, header, menu, body, footer
--%>
<%@ page errorPage="errorpage.jsp" %>
<html>
<head>
<title>
<tiles:getAsString name="title"/>
</title>
<layout:skin/>
<fwk:javascriptlib/>
</head>
<frameset rows="60,*,60">
<frame src="<%=request.getContextPath()%><tiles:getAsString name="header" />"
SCROLLING="no" FRAMEBORDER=0 name="header" >
<frameset cols="*,*">
<frame src="<%=request.getContextPath()%><tiles:getAsString name="menu" />"
name="menu" SCROLLING="auto" FRAMEBORDER=0>
<frame src="<%=request.getContextPath()%><tiles:getAsString name="body" />"
name="body" SCROLLING="auto" FRAMEBORDER=0>
```

```
</frameset>
<frame src="<%=request.getContextPath()%><tiles:getAsString name="footer" />"
name="footer" SCROLLING="no" FRAMEBORDER=0>
</frameset>
<noframes>
<body></body>
</noframes>
</html>
```

Código 14-9 Implementación del Layout para Frames

Capítulo 15.

15. Servicio de Multidiioma

La Internacionalización tiene como objetivo conseguir una aplicación multiidioma. Para conseguir este objetivo es necesario que el desarrollo de la aplicación sea independiente, de forma que no haya que realizar ninguna reingeniería cada vez que se incorpore un nuevo lenguaje a la aplicación.
Una aplicación internacionalizada, puede realizarse de 2 formas:

- Mediante la replicación de páginas para cada idioma
- Mediante el uso de la Internacionalización de Java y extensiones

La primera solución, hasta ahora ampliamente usada en muchas Webs implica un gran coste de réplica y mantenimiento, ya que cualquier nuevo requerimiento o cambio comporta realizar más de 1 desarrollo, uno para cada idioma.
La segunda solución, usada por el framework permite que:

- Se puedan añadir nuevos lenguajes sin tener que hacer cambios al código
- Los textos, imágenes y mensajes se guardan de manera externa al código
- Datos como fechas, moneda, etc. se formateen correctamente según el idioma del usuario

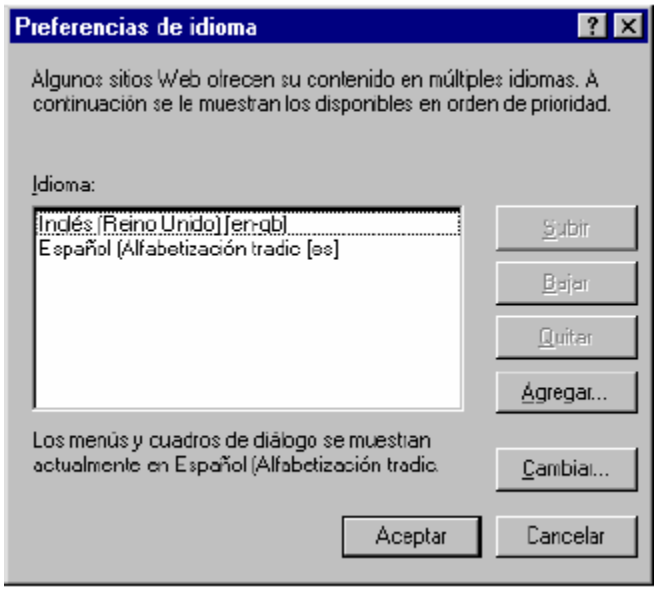
15.1 Selección Idioma Usuario

Cuando queremos una aplicación internacionalizada, nos surgen 2 necesidades:

1) Que el idioma de la aplicación sea por defecto el configurado en el navegador
2) Que el usuario pueda seleccionar otro idioma al de defecto

15.1.1 Inicio por Idioma por Defecto del Navegador

En el framework, cuando arranca el aplicativo, se instancia como idioma el que venga configurado por el navegador.
Por ejemplo, si el usuario quiere cambiar el idioma por defecto del navegador, en Internet Explorer seleccionará 'Herramientas|Opciones de Internet|Idiomas':



Una vez se ha procesado la petición, si el usuario todavía no se ha autentificado, aparecerá la página de logon. El Framework hará que esta página aparezca con el idioma por defecto del usuario:

NOTA: Esta página, debe denominarse 'logon.jsp' y encontrarse en la raíz del context-root.

En el código de la página de logon.jsp siguiente podemos ver cómo se ha obtenido el comportamiento arriba mostrado:

```
<% java.util.Locale locale = request.getLocale();
String language = locale.getLanguage();
%>
<fwk:select key="field.language" property="language" styleClass="LABEL"
match="<%=language%>">
<fwk:option key="label.language.ingles" value="en" />
<fwk:option key="label.language.catalan" value="ca" />
<fwk:option key="label.language.castellano" value="es" />
</fwk:select>
```

Mediante request.getLocale() se obtiene el locale que viene en la petición (en http en cada petición llega el idioma de usuario). Este valor es el que se toma para escoger del select el idioma.

15.2 Ficheros de Recursos

Los ficheros de recursos, denominados ResourceBundle en el mundo I18N (Internationalization) de Java, permiten definir mediante las convenciones de la clase PropertyResourceBundle (ver java.sun.com) tuplas clave=valor que definan la correspondencia entre la clave usada en las páginas o código y el literal a utilizar.
Los ficheros de recursos deben tener la extensión '.properties' y estar ubicados en '/WEB-INF/classes'.
En el framework este fichero se denomina 'ApplicationResources.properties'. Para cada idioma, debemos crear un fichero de recursos, que siga la nomenclatura ApplicationResources_lg_pa.properties', donde 'lg' es el código del lenguaje y 'pa' el código del país, o bien 'ApplicationResources_lg.properties', si no queremos hacer distinción por países.
En las direcciones http://www.unicode.org/unicode/onlinedat/languages.html y http://www.unicode.org/unicode/onlinedat/countries.html , pueden encontrarse las codificaciones de los lenguajes y países, respectivamente.
Así, por ejemplo, si queremos que nuestra aplicación esté para inglés, español y catalán tendríamos los ficheros: ApplicationResources_en.properties, ApplicationResources_es.properties y ApplicationResources_ca.properties.

En la página de logon, se definen 2 componentes donde la etiqueta a mostrar es la clave del literal:

```
<fwk:text key="field.username" isRequired="true" property="username"
styleClass="LABEL" size="10" maxlength="10"/>
<fwk:password key="field.password" isRequired="true" property="password"
styleClass="LABEL" size="10" maxlength="10"/>
```

Según el idioma seleccionado, el framework traducirá la clave indicada por el literal correspondiente en el fichero de recursos del idioma. Si por ejemplo se ha seleccionado como Locale 'en', se irá al fichero 'ApplicationResources_en.properties'.

ApplicationResources_ca.properties:

```
field.username = Usuari
field.password = Paraula de pas
```

ApplicationResources_en.properties:

```
field.username = User
field.password=Password
```

Es importante que se proporcione un fichero 'ApplicationResources.properties' por defecto. Este es el fichero que se utiliza en caso de que el lenguaje escogido por el usuario no corresponda a ninguno de los ficheros de recursos definidos.
En caso de que para un lenguaje determinado exista un fichero específico, la obtención del literal a partir de una clave se obtendrá de este fichero, y no se iría al fichero por defecto en ningún caso, se encuentre o no definida la clave.

15.3 Tags Multiidioma

En el framework se ofrecen multitud de tags que hacen uso del multiidioma. Por lo general, en ellos se define un parámetro 'key' que especifica la clave del fichero de recursos a usar.
Ver 'Servicio de Tags ' para más referencia.

15.4 Imágenes Multiidioma

Para mostrar imágenes, usaremos el tag <fwk:img>. En éste debe especificarse en el parámetro 'pageKey' la clave que especificará la ruta a la imagen.

```
<html:img pageKey="images.template.logos.seat"/>
```

La ruta a indicar en el fichero de recursos es siempre respecto al directorio '/WEB-INF'.

```
images.template.logos.seat = /template/images/logo_seat.gif
```

Es importante que todo el código sea internacionalizado. El caso más claro está en el uso de imágenes. Los desarrolladores olvidan en algunas ocasiones que la url de la imagen se especifique en el fichero de recursos y realizan el código con urls específicas.

15.5 Presentación de Mensajes Multiidioma

Para presentar mensajes en las páginas se hace uso del tag '<bean:message>':

```
<bean:message key="logon.help"/>
```

15.6 Obtención de Literales desde las Clases y JSPs

Para obtener un literal asociado a una clave podemos acceder desde diferentes puntos de la arquitectura. Desde una página JSP o tag podemos obtener un literal mediante el método:

```
getLabel(pageContext,key,null);
```

de la clase com.juoc.struts.taglib.util.LayoutUtils.

15.7 Definiciones Necesarias para el Framework

Para que el framework pueda mostrar los mensajes correctamente, es necesario que en el fichero de recursos de la aplicación se hayan definido varias claves con sus literales respectivos.
Por esta razón, es importante que en el fichero proporcionado en la distribución se añadan los literales específicos de la aplicación y se mantengan invariables los definidos para los servicios del framework.

Capítulo 16.

16. Servicio de Tags.

El framework proporciona un amplio abanico de tags, que proporcionan al desarrollador de las necesidades más comunes en el desarrollo de las páginas JSPs.

16.1 Configuración Librerías

Para poder hacer uso de las librerías de tags proporcionadas es necesario inicialmente definir en el fichero 'web.xml' las referencias a las librerías, y posteriormente, en cada página en la que hagamos uso de un tag introducir el tag de uso de la/s librería/s.

```
    <!-- Struts Tag Library Descriptors -->
    <taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
    </taglib>
    <taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
    </taglib>
    <taglib>
    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
    </taglib>
    <!-- Tiles Library Descriptor -->
    <taglib>
    <taglib-uri>/WEB-INF/tiles.tld</taglib-uri>
    <taglib-location>/WEB-INF/tiles.tld</taglib-location>
    </taglib>
    <!-- Tiles Library Descriptor -->
    <taglib>
    <taglib-uri>/WEB-INF/html-table.tld</taglib-uri>
    <taglib-location>/WEB-INF/html-table.tld</taglib-location>
    </taglib>
    <!-- Descriptor Framework-->
    <taglib>
    <taglib-uri>/WEB-INF/framework.tld</taglib-uri>
    <taglib-location>/WEB-INF/html-table.tld</taglib-location>
</taglib>
```

Código 16-1 Definición Librerías en el fichero web.xml

En cada una de las páginas realizaremos la referencia a las librerías de tags de las que hagamos uso:

```
    <%@ taglib uri="/WEB-INF/struts-layout.tld" prefix="layout" %>
    <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
    <%@ taglib uri="/WEB-INF/html-table.tld" prefix="tabla" %>
    <%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
    <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
    <%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>
```

Código 16-2 Declaración uso librerías tags en las páginas JSP

16.2 Uso de Formularios

En las aplicaciones Web que desarrollamos tenemos multitud de formularios. Estos formularios, están estructurados, por lo general, mediante una rejilla en la que aparecen literales a la izquierda de cada uno de los componentes. En Struts, la realización de un componente como el explicado se haría de la siguiente forma:

```
<th valign="top" class="LABEL">
      <span class="LABEL">
            <bean:message key="field.vehiculo.matricula "/>
      </span>
</th>
<td valign="top" class=LABEL>
      <html:text       property=”matricula”       size=”20”       maxlength=”20”/>
</td>
```

Código 16-3 Código de un Componente mediante Struts

Este código resulta complejo de gestionar y se requiere un esfuerzo adicional de desarrollo. Por qué no un tag que creara todo este código y fuera sencillo de usar? Con el nuevo tag proporcionado por el framework el código equivalente sería:

```
<fwk:text     key="field.vehiculo.matricula"     property="matricula"     size="20"
maxlength="20"/>
```

Como podemos ver, el código es más escueto y obtenemos un código mucho más legible y estructurado.

Modos de Display de Formulario

En el framework, la mayor ventaja en el uso de los tags de formularios es el uso de los modos. Para explicarlo, veamos un ejemplo. Imaginemos que tenemos un mantenimiento de vehículos y existe una página para mostrar la información de un vehículo. Habrá usuarios que podrán ver la información pero no editarla, mientras que otros usuarios podrán modificar la información. Normalmente, sin el uso del framework, deberíamos desarrollar 2 páginas jsp: una para mostrar la información y otra para modificarla (también podríamos hacer una única JSP con una lógica de gestión en ella, pero sería complicada y el código resultaría arcaico y poco legible). El objetivo es que se cree una sola página para crear,editar y visualizar.
Existen 3 modos de visualización definidos en el framework:
• FormUtils.INSPECT_MODE. Modo de consulta
• FormUtils.EDIT_MODE. Modo de edición
• FormUtils.CREATE_MODE. Modo de alta

Para definir en qué modo se encontrará el formulario una vez el Action devuelva la página, se puede establecer mediante el método:
com.juoc.struts.taglib.util.FormUtils.setFormDisplayMode(request,form,FormUtils.XXX_MODE)

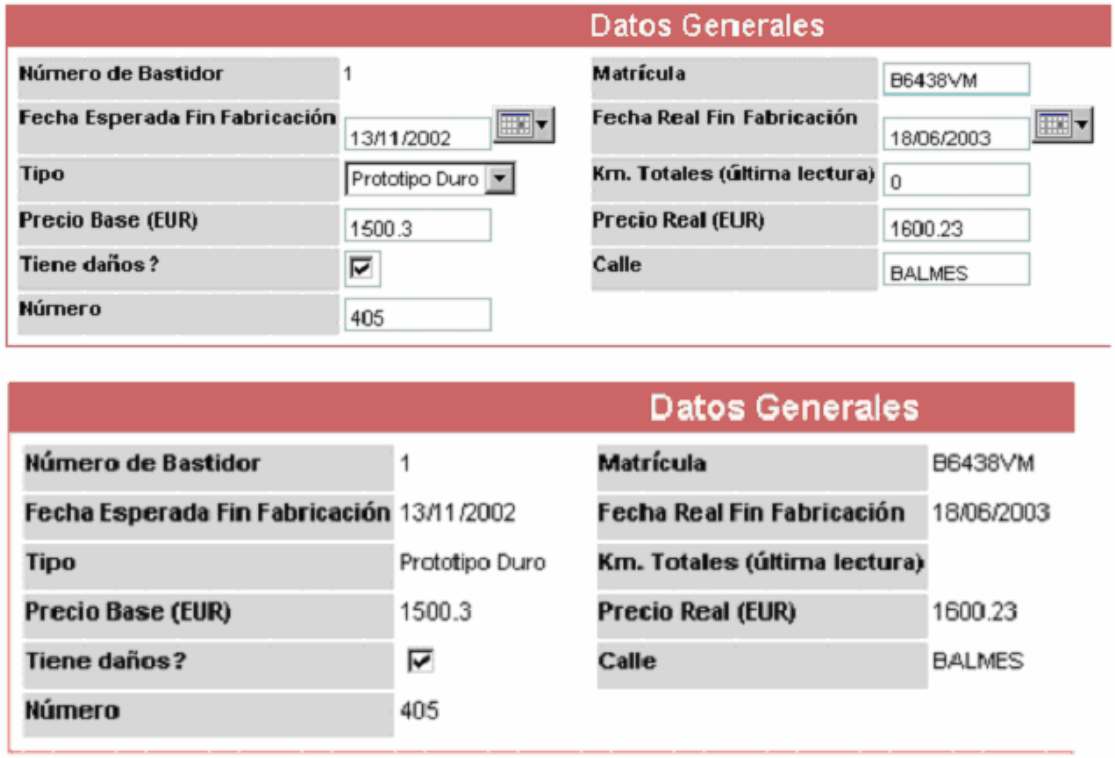


Figura 16-1. Ejemplo de Modo de Formulario

Modos de Display de Campos

El modo de los campos permite especificar si un campo input debe ser escribible, de lectura, oculto o que no aparezca en el formulario dado un modo de display de formulario. Esto permite que un campo que forma parte de la clave primaria sea solo editable durante la creación, o mostrar campos calculados solo cuando se visualiza la información y no cuando se edita.
El comportamiento por defecto es que los campos se muestren de escritura (como los tags struts o HTML) si el modo del formulario es CREATE o EDIT, y se muestren de lectura (valor texto plano sin componente HTML y un hidden también para el campo) si el modo del formulario es INSPECT.
Este comportamiento por defecto puede alterarse en los campos mediante el atributo '*mode*' (ver cada uno de los tags), cuyo valor debe seguir el siguiente patrón: 'X,Y,Z' donde X es el comportamiento que tendrá el campo en modo de creación del formulario, Y el comportamiento en modo edición y Z en modo de consulta. Los valores posibles para X,Y y Z son:

- E. Editable
- I: Inspeccionable (de consulta)
- N: No se muestra y no se crea un campo hidden
- H: Oculto (no se muestra pero se crea un campo hidden)

Ejemplo:

```
<fwk:text key="field.vehiculo.numBastidor" styleClass="LABEL" isRequired="true" property="numBastidor"
mode="E,I,I" size="20" maxlength="20"/>
```

Muestra de botones según modo:

```
<fwk:image srcKey="images.template.button"
       keyText="buttons.text.create"          mode="D,D,D"          reqCode="create"
       styleText="textImage">
</fwk:image>
<fwk:image srcKey="images.template.button"
```

```
      keyText="buttons.text.save"mode="D,N,N"                    reqCode="saveNew"
      styleText="textImage">
</fwk:image>
<fwk:image srcKey="images.template.button"
      keyText="buttons.text.save"          mode="N,D,N"          reqCode="save"
      styleText="textImage">
</fwk:image>
```

Una vez especificado en qué modo funcionará el formulario, los componentes aparecerán según el modo especificado, sin que por ello hayamos tenido que crear código JSP adicional.

16.3 Configuración Presentación Tags

El Servicio de Tags permite definir varias propiedades de uso general para configurar su comportamiento. Parte de la configuración se define mediante el fichero '/WEB-INF/classes/**Struts-Layout.properties**'. En este fichero, se definen principalmente:

Parámetro	Descripción
skin	Fichero de estilos que usaremos en la aplicación. En caso de que en un momento determinado nuestra aplicación deba usar otro fichero de estilos podremos cambiar este valor y las páginas no deberán cambiar su código Ejemplo: skin = seat.css El lugar de ubicación del fichero de estilos lo define el parámetro '**directory.config**' NOTA: Se permite en el framework, para temas de personalización que se pueda cambiar cual es el estilo a usar dependiendo del usuario, mediante el siguiente método: com.juoc.struts.taglib.util.LayoutUtils.setSkin(HttpSession aSession, String anSkin)
directory.images	Directorio en el que se encuentran las imágenes de uso general (imágenes listados, calendario,etc.). Este directorio se considera a partir de 'WEB-INF' Ejemplo: directory.images = layouts/images
directory.config	Directorio en el que se encuentra el fichero de estilos definido mediante 'skin' y el fichero js 'javascript.js' que realiza funciones de los tags (presentar el asterisco para los campos obligatorios y que desaparezca una vez entrado un valor, etc.) Ejemplo: directory.config = layouts/config Este directorio se considera a partir de 'WEB-INF'
display.null.fields	Indicar 'true' si queremos mostrar los campos que tengan un valor vacío en modo INSPECT
panel.class	Especifica el tipo de panel a usar cuando se especifique el tag <fwk:panel>. Los valores posibles son: • fr.improve.struts.taglib.layout.util.BasicPanel: Representa un panel con bordes y una fila inicial con formato que presenta el título • fr.improve.struts.taglib.layout.util.ImagePanel: Representa un panel con título pero con los bordes definidos por imágenes. Las imágenes que se usarán como bordes se definen mediante los parámetros: 'top_LEFT','top','top_RIGHT','left','right','bottom_LEFT'bottom', y 'bottom_RIGHT'. En estos parámetros se especifican las rutas a las imágenes Mediante 'height' y 'width' se determina el tamaño de las imágenes
layout.calendar	Nombre de la imagen dentro del directorio de imágenes que mostrará el icono para seleccionar

una fecha en el tag ‘date’

16.4 Tags HTML

Algunos de los tags proporcionados por el framework extienden el comportamiento definido por los tags de struts.

16.4.1 Aspectos Generales

La mayoría de los tags incluidos en este apartado permiten definir serie de atributos comunes. Algunos de estos atributos más genéricos son:

- key. Clave del literal que aparecerá a la izquierda del componente
- property. Nombre del atributo del ActionForm
- styleClass. Estilo a aplicar, tanto a la etiqueta como al componente. Por defecto, si no se especifica es ‘LABEL’.

Para representar la etiqueta se hace uso del estilo ‘TH.*nombreEstilo*’ y ‘SPAN.*nombreEstilo*’. Mediante TH se representa el contorno de la etiqueta, mientras que mediante SPAN se representa el estilo del texto. Para representar el componente se hace uso del estilo ‘TD.*nombreEstilo*’.

```
SPAN.LABEL
       { text-align:left;
       color: black;
       font-size: 8pt;
       font-family: Arial;
       }
TD.LABEL
       { background-color : #ffffff;FONT-WEIGHT: bold;FONT-SIZE: 12px;
       COLOR: #00000 FONT-FAMILY: Arial, Helvetica, sans-serif;
       TEXT-DECORATION: none }
TH.LABEL { background-color : #F3F3F3; FONT-WEIGHT: bold; FONT-SIZE: 12px;
       COLOR: #00000 FONT-FAMILY: Arial, Helvetica, sans-serif;
       TEXT-DECORATION: none }
```

Código 16-4 Estilos asociados a los componentes

16.4.2 Tag HTML

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:html>

El tag HTML genera la estructura de contenido de una página HTML (head, body, etc.) e incorpora todas las referencias a ficheros de estilos y ficheros o código Javascript necesarios en el framework.

El contenido generado por el tag html sería del siguiente estilo:

```
<html>
<head>
<title>Titulo de la página</title>
<link rel="stylesheet" href="ficheroEstilos.css" type="text/css">
<SCRIPT SRC="/js/utils.js">
</SCRIPT>
```

```
    <SCRIPT SRC="/js/layout.js">
    </SCRIPT>
    …
    Código Javascript dependiente Locale (calendario, etc.)
    </head>
    <body>
    Navegación
    Cuerpo página
</body>
```

Código 16-5 Esqueleto del código generado por el tag html

16.4.3 Uso de frames

El uso de frames incide en la manera de generar el código de las páginas contenidas.
Si usamos un layout sin frames, las páginas contenidas por el layout no deben generar este código, pues obtendríamos varios tags HTML, HEAD,... anidados. Por este motivo, el tag crea o no el código arriba mostrado en función de un parámetro de configuración que se especifica en *Struts-Layout.properties*.

```
#uso de frames para el layout de la aplicacion
usingFrames = true
```

Código 6-6 Configuración para uso de frames

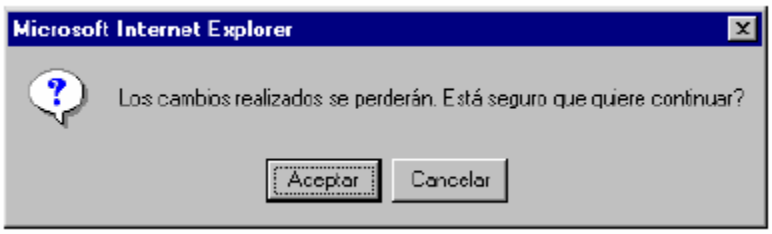
16.4.4 Tag Formulario

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:form>

Para crear un formulario usaremos el tag <fwk:form>. Este tag define cómo se obtendrán y presentarán los datos contenidos en un *ActionForm* del *Action* que resuelve la petición. El tag, además del comportamiento propio de Struts proporciona:

• Validación dinámica de los datos del formulario (el desarrollador no debe realizar ningún código de validación Javascript, se crea dinámicamente)

• Comprobación de pérdida de datos cuando el usuario pulse una opción que le lleve fuera del formulario



Algunos de los atributos que se pueden definir son:

Atributo	Descripción
action	URL del submit del formulario. Éste valor debe coincidir con algún valor que se haya definido en el fichero de configuración de struts (struts-config.xml) como un mapping. **Importante**: Indicar la barra inicial y el .do en el nombre de la url Ejemplo: '/mantVehiculo.do'
validator (opcional)	Indica si se va a realizar la validación del formulario mediante Validator (ver 'Servicio de Validacion'). Si se indica 'true' es obligatorio que se haya definido la validación en el fichero 'validator.xml'. Por defecto es 'true'. En cualquier caso, si el reqCode asociado al submit es '**delete**' o '**create**' el framework no realiza validación
enctype (opcional)	Permite especificar el content encoding a usar cuando se haga el submit si el method es POST (método por defecto del formulario). Indicar 'multipart/form-data' si usamos internamente al formulario el tag file. Si no se especifica, por defecto es 'application/x-www-form-urlencoded'
checkFormChange (opcional)	Indicar 'true' si queremos que se avise al usuario de que va a perder los cambios realizados en el formulario. Este aviso se realizará si se han hecho cambios en el formulario y se ha pulsado un link creado con el tag image o el tag link que tenga como valor de parámetro checkFormChange=true (ver tag Image y tag Link) El mensaje a mostrar se configura en el fichero 'ApplicationResources.properties' en el literal '**layout.dataLost**'
target (opcional)	Indica cual es el target de ventana donde se mostrará el resultado del submit del formulario: '_self', '_top', '_parent',.etc. En caso de que el formulario se encuentre en un frame y no se haya especificado el target, es el tag com.juoc.struts.taglib.html.ImageTag quien generará el código necesario para que la carga se realice en el '_top' en lugar de '_self' como sería el comportamiento propio de Struts

Ejemplo:

```
<fwk:form action="/mantVehiculo.do" width="100%" enctype="multipart/form-data"> …
</fwk:form>
```

Código 16-7 Ejemplo de uso del tag form

El tag genera siempre un <table> de inicio y fin que contiene todo el formulario. Es importante tener en cuenta este aspecto, pues determinará como deben generarse internamente los tags. Cada uno de los tags HTML (textfield, etc.) generan un <tr><td> de inicio y fin. De esta forma, los componentes son alineados verticalmente, uno por fila. Si queremos crear 2 o más componentes en una misma fila, debemos hacer uso del tag <fwk:line> (ver 'Tag Línea').

16.4.5 Tag Imagen Submit

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:image>
Ejemplo de uso:	<fwk:image srcKey="images.template.button" keyText="buttons.text.create" checkFormChange="true" validate="false" mode="D,D,D" reqCode="create" styleText="textImage"> </fwk:image>

Para introducir los botones que actuarán como submit del formulario, usaremos principalmente el tag <image> (actúa como un 'input type=image'). Este tag permite indicar los siguientes parámetros:

Atributo	Descripción
srcKey	Clave del fichero de recursos de mensajes que especifica la URI de la imagen a utilizar para este tag
keyText (opcional)	Clave del fichero de recursos de mensajes para indicar un texto adicional a la imagen
mode	Modo en el que aparece el botón imagen ('D' si se quiere mostrar, 'N' para no mostrar). Especifica si debe mostrarse o no el botón en cada uno de los modos del formulario (creación, edición e inspección) Ejemplo: mode="D,N,N" significa que el botón imagen aparecerá en modo creación, pero no en modo edición ni en inspección. Este sería el caso del botón al que asociaríamos la acción de inserción de un elemento en la base de datos
reqCode	Valor del método a llamar de la acción que se haya configurado como controlador del formulario (subclase de com.juoc.struts.base.BaseDispatchAction)
alignText (opcional)	Alineación del texto con respecto a la imagen. Valores posibles: left, right. Por defecto: right
validate (opcional)	Indicar 'false' si no se quiere validar la entrada de datos del formulario, por defecto 'true'.
checkFormChange (opcional)	Indicar 'true' si debe comprobarse al pulsar el botón que el usuario ha realizado cambios.

En el siguiente ejemplo (un mantenimiento), se han especificado 4 botones: crear, insertar, actualizar y borrar. El primero tendría como objetivo mostrar una pantalla inicializada para la entrada de datos. El segundo realizaría la propia inserción. El tercero serviría para actualizar el elemento que estemos consultando y el último para borrarlo de la base de datos.

```
<fwk:image srcKey="images.template.button" keyText="buttons.text.create"
checkFormChange="true" validate="false" mode="D,D,D"
reqCode="create" styleText="textImage">
</fwk:image>
<fwk:image srcKey="images.template.button" keyText="buttons.text.save"
mode="D,N,N" reqCode="saveNew" styleText="textImage"></fwk:image>
<fwk:image srcKey="images.template.button" keyText="buttons.text.save"
mode="N,D,N" reqCode="save" styleText="textImage"></fwk:image>
<fwk:image srcKey="images.template.button" keyText="buttons.text.delete"
mode="N,D,N" reqCode="delete" styleText="textImage"></fwk:image>
```

Código 16-8 Ejemplo de JSP de uso del tag image

El primer tag imagen especifica que debe mostrarse como imagen la especificada en el literal 'images.template.button', y como texto el especificado en la clave 'buttons.text.create'. Además, especifica que en caso de ser clickado, debe comprobar que dentro del formulario en el que se encuentra el botón imagen, el usuario no ha realizado cambios.
Mediante el modo 'D,D,D' estamos especificando que el botón aparecerá en modo creación, edición e inspección.
Con 'create' como reqCode, indicamos que en caso de ser clickado, se hará una llamada al método llamado 'create' del Action asociado a la url del formulario (se configura en 'struts-config.xml'). En el Action tendríamos el siguiente método asociado a la llamada del botón:

```
public ActionForward create(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception
```

El estilo 'textImage' define cual es el estilo a usar para el texto que aparece al lado de la imagen.

Ejemplo estilo textImage

```
textImage {
text-align:left;
color: #cc0000;
font-size: 8pt;
font-weight : bold;
font-family: Arial;
}
```

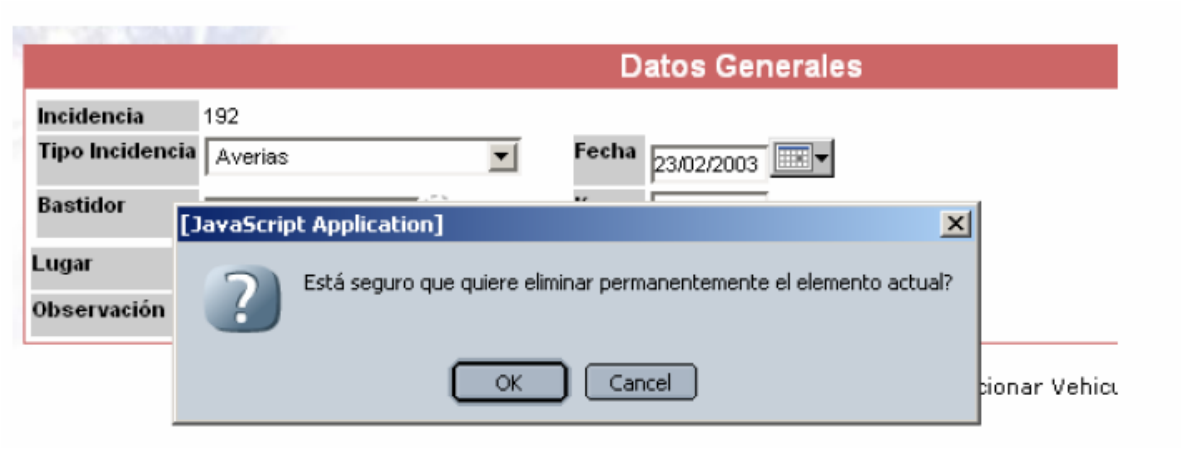
ApplicationResources_es.properties

```
images.template.button = /template/images/
buttons.text.save = Guardar
buttons.text.create = Nuevo
buttons.text.delete = Eliminar
```

Código 16-9 Configuración del estilo del texto para el tag image

16.4.6 Aviso de Confirmación de Borrado

En caso de que se haya especificado como reqCode '**delete'**, el tag generará un mensaje de confirmación de borrado.
El mensaje internacionalizado a mostrar se define en la clave '**messages.deleteConfirmation**'.




```
ApplicationResources_es.properties
messages.deleteConfirmation = Está seguro que quiere eliminar permanentemente el
elemento actual?
```

Figura 16-2. Comprobación de Borrado

16.4.7 Tag TextField

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:text>
Ejemplo de uso:	<fwk:text key="field.vehiculo.numBastidor" keyToolTip="help.fields.vehiculos.numBastidor" styleClass="LABEL" isRequired="true" property="numBastidor" mode="E,I,I" size="20" maxlength="20"/>

Este tag genera código de entrada de un campo de texto.
Además del comportamiento implícito de Struts proporciona:

- Conversión de datos cuando el usuario cambia de componente: cambio a mayúsculas, eliminación blancos, etc.
- Creación de estructuración de los componentes (<td>...), de manera que el layout del componente no se especifica, se crea automáticamente
- Indicador de obligatoriedad del campo mediante una iconificación. En el momento en el que el usuario introduce el valor desaparece la indicación.
- Autotabulación cuando se llega a un número de caracteres introducidos
- Modo. Permite especificar en qué modo debe aparecer el campo dependiendo del modo del formulario.
- Ayuda sensitiva

Atributo	Descripción
key	Clave del fichero de recursos que indica la etiqueta que aparecerá a la izquierda del componente de texto
property	Nombre del campo input y el nombre de la propiedad correspondiente al ActionForm

convertTo (opcional)	Permite especificar qué tipo de conversión realizar. Esta conversión se realiza en tiempo de entrada de datos y también en tiempo de obtención de datos. Los valores permitidos son: • uppercase. Pasa a mayúsculas el valor del componente • lowercase. Pasa a minúsculas el valor del componente • trim. Elimina espacios en blanco a izquierda y derecha del valor del componente, y sustituye 2 blancos seguidos por un único blanco • double. Realiza una conversión al formato definido en el fichero de recursos en la clave '**org.apache.struts.taglib.bean.format.double'**, de forma que traduce las ',' al punto decimal y realiza el redondeo al número de decimales que se haya indicado en este literal.
autoTab (opcional)	Permite indicar el número de caracteres a introducir por el usuario, para que el sistema pase al siguiente componente automáticamente
isRequired (opcional)	Indicar 'true' para indicar al usuario que la entrada de datos es obligatoria. Se mostrará una imagen a la derecha del componente de texto (actualmente un asterisco en rojo) cuando la entrada esté vacía. En el momento que el usuario introduzca datos y cambie de componente, desaparece el símbolo que indica obligatoriedad Por defecto: false Si se requiriera cambiar la imagen que muestra cuando un campo es obligatorio, podemos reemplazar el fichero **layouts\images\last.gi**
mode (opcional)	Indica el modo en el que mostrará el componente dependiendo del modo del formulario.
formatStr	Formato a aplicar al componente actual. **Actualmente** solo se puede especificar un formato decimal. De esta forma, se puede especificar, por ejemplo, que el formato para el componente actual sea de '#.##'. En este caso, la entrada realizada obtenida del objeto, se pasaría a 2 decimales. Este mecanismo permite sobrescribir el formato por defecto que se haya definido en la clave 'org.apache.struts.taglib.bean.format.double' del fichero de recursos (en este podría ser de 4 decimales, por ejemplo)
formatKey	Clave que especifica el formato a aplicar al componente actual. De la misma aplicación que el atributo anterior con la diferencia de poder aplicar un formato diferente según el Locale del usuario conectado.
keyToolTip (opcional)	Clave del mensaje a mostrar como ayuda sensitiva. En el literal del fichero de recursos se puede usar **código HTML o texto plano**.
selectOnFocus	Indicar true si debe seleccionarse todo el texto contenido cuando reciba el foco. Por defecto: true
styleClass (opcional)	Clase estilo asociada al componente. Por defecto, el estilo que el framework usa es '**LABEL**'

popupReqCode **popupPageKey** **popupKeyText** **popupStyleText**	Campos que permiten definir la selección de un valor en un nueva ventana que será copiado a este textfield popupReqCode : reqCode de la acción popupPageKey, popupKeyText,popupStyleText (ver definición en ImageTag)
popupWidth **popupHeight**	popupWidth, popupHeight: Tamaño de la ventana que se abrirá
indexed (opcional)	Indicar 'true' si el atributo asociado forma parte de un array
size	Tamaño visible del campo
maxlength	Número máximo de caracteres intraducibles en el campo

Tag en la página JSP

```
<fwk:text key="field.vehiculo.numBastidor"
keyToolTip="help.fields.vehiculos.numBastidor"
styleClass="LABEL" isRequired="true" property="numBastidor" mode="E,I,I"
size="20" maxlength="20"/>
```

ApplicationResources.properties

```
field.vehiculo.numBastidor = Número de Bastidor
...
help.fields.vehiculos.numBastidor = <HTML><BODY>Número de bastidor, ayuda<BR>..
</BODY></HTML>
```

Código 16-11 Ejemplo de uso del tag textfield

16.4.8 Indicación de Campos Requeridos

En el tag, la propiedad '**isRequired**' permite indicar que un campo es obligatorio. Este parámetro se puede definir en cualquiera de los tags HTML en los que por lógica tendría validez (textfield, textarea, select, etc.).
El framework, cuando un componente indica 'isRequired=true', si el valor asociado no está informado muestra un icono (un asterisco si no se modifica la imagen). En el momento en que el usuario introduce el valor y cambia de componente, el icono desaparece.

Ahora bien, este atributo no implica la validación de su introducción en el momento de realizar el submit, se trata únicamente de una representación que indica al usuario qué debe introducir un valor. Esta validación de requerido debe realizarse mediante el 'Servicio de Validación'.

16.4.9 Ayuda Sensitiva

La ayuda sensitiva se configura mediante la propiedad 'keyToolTip', donde se indica la clave del fichero de recursos que define el mensaje a mostrar.
La ayuda sensitiva puede mostrarse con cabecera o sin cabecera. El hecho de usar o no cabecera se define por el literal 'help.caption'. En caso que se defina el literal aparecerá la cabecera, en caso contrario no.

Figura 16-3. Ayuda sensitiva

El estilo a usar (fichero css) para la ayuda sensitiva se define mediante los estilos:

- tooltipfg. Clase estilo para la tabla interna del tooltip (es la parte propia donde aparece la documentación)
- tooltipbg. Clase estilo de la tabla externa
- tooltiptextfont. Clase estilo para el texto del área principal (donde aparece la documentación)
- tooltipcaptionfont. Clase estilo del texto que se muestra en la cabecera

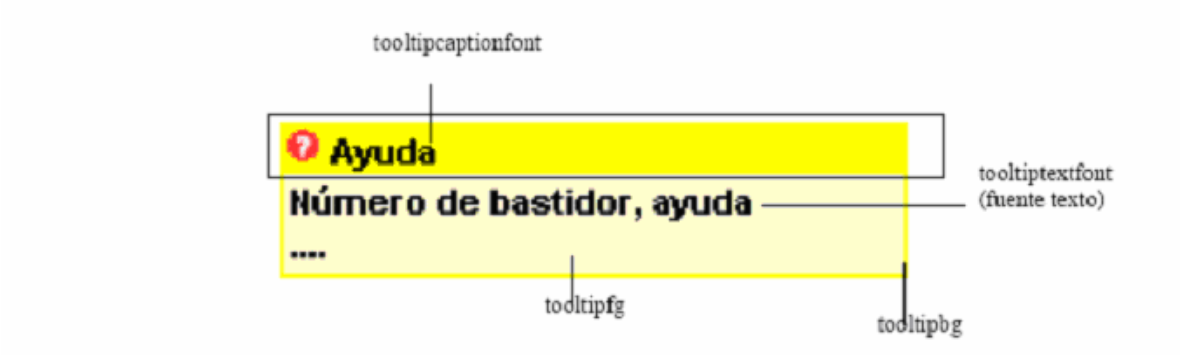


Figura 16-4. Estilos Ayuda sensitiva

En el ejemplo mostrado, los estilos se han definido de la siguiente forma:

```
/* Clase para la tabla interna del tooltip */
 .tooltipfg {
text-align:left;
color: #FFFFCC;
font-size: 8pt;
font-weight : bold;
font-family: Arial;
background-color: #FFFFCC;
}
/* Clase para la tabla externa del tooltip */
.tooltipbg {
text-align:left;
color: #FFFF00;
font-size: 8pt;
font-weight : bold;
font-family: Arial;
background-color: #FFFF00;
}
/* Clase texto área principal */
.tooltiptextfont {
text-align:left;
```

```
color: #000000;
font-size: 8pt;
font-family: Arial;
}
/* Clase para el texto del titulo de la ventana tooltip */
.tooltipcaptionfont {
text-align:left;
color: #000000;
font-size: 8pt;
font-weight : bold;
font-family: Arial;
}
.tooltipclosefont {
text-align:left;
color: #cc0000;
font-size: 8pt;
font-weight : bold;
font-family: Arial;
background-color: #CCCCCC;
}
```

La imagen de ayuda se encuentra en el directorio de imágenes definidas en el fichero 'Struts-Layout.properties' en la variable *directory.images* (actualmente 'directory.images = layouts/images') y con nombre de fichero '**help.gif**'.

16.4.10 Selección de Valor en una Nueva Ventana

En determinados casos, el valor a introducir en un textfield es desconocido por el usuario, y el número de valores posibles es inmensamente grande como para usar una combo. Una opción a usar en estos casos, es abrir una ventana en la que el usuario puede realizar una búsqueda, y el usuario escoja de la lista de resultados el valor que será copiado al textfield original.
Para conseguir este comportamiento se hace uso de las propiedades que comienzan por popup.

Los pasos necesarios son:

1) Definir el textfield origen

```
<fwk:text key="field.incidencia.bastidor" property="bastidor" isRequired="true"
mode="E,E,I" styleClass="LABEL" size="20" maxlength="15"
popupReqCode="searchVehiculo" popupPageKey="images.template.button"
popupKeyText="buttons.text.create" popupStyleText="textImage" />
```

En este ejemplo se especifica que el reqCode que debe enviar cuando sea pulsado el botón (representado con los atributos popupPageKey, popupKeyText y popupStyleText) sea 'searchVehiculo'. Cuando el botón sea pulsado se abrirá una nueva ventana que mostrará como contenido el devuelto por la operación asociada al 'reqCode'.

2) Especificar la operación asociada al reqCode en la clase Action. La operación definida únicamente debe hacer un forward a la página deseada.

```
/**
* Accion encargada de crear una incidencia
*/
public ActionForward searchVehiculo(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws java.lang.Exception {
try {
return new ActionForward("/listSelecVehiculosCesion.do?reqCode=newSearch");
} finally {
}
```

3) Definir en el link de la página destino cual es el campo de la ventana origen sobre el que debe copiar el valor en caso de que ese enlace sea clickado.

```
<tabla:define id="bastidor">
<% java.util.Hashtable mapa = new java.util.Hashtable();
mapa.put("bastidor",
```

```
((com.test.sigeva.general.model.CesionVO)elemento).getBastidor());
pageContext.setAttribute("mapa", mapa);
%>
<fwk:link page="/mantPrestamoMedio.do?reqCode=edit" name="mapa" scope="page"
popupField="bastidor">
<bean:write name="elemento" property="vehiculo.bastidor"/>
</fwk:link>
</tabla:define>
```

En 'page' se especifica cual será la petición a realizar si no se ha llegado a la página por el mecanismo de selección por ventana.
El usar page y popupField conjuntamente permite que podamos reutilizar la misma página para que al seleccionar un enlace de la lista de resultados nos muestre los datos asociados o bien copie el valor a una ventana origen y cierre la ventana.

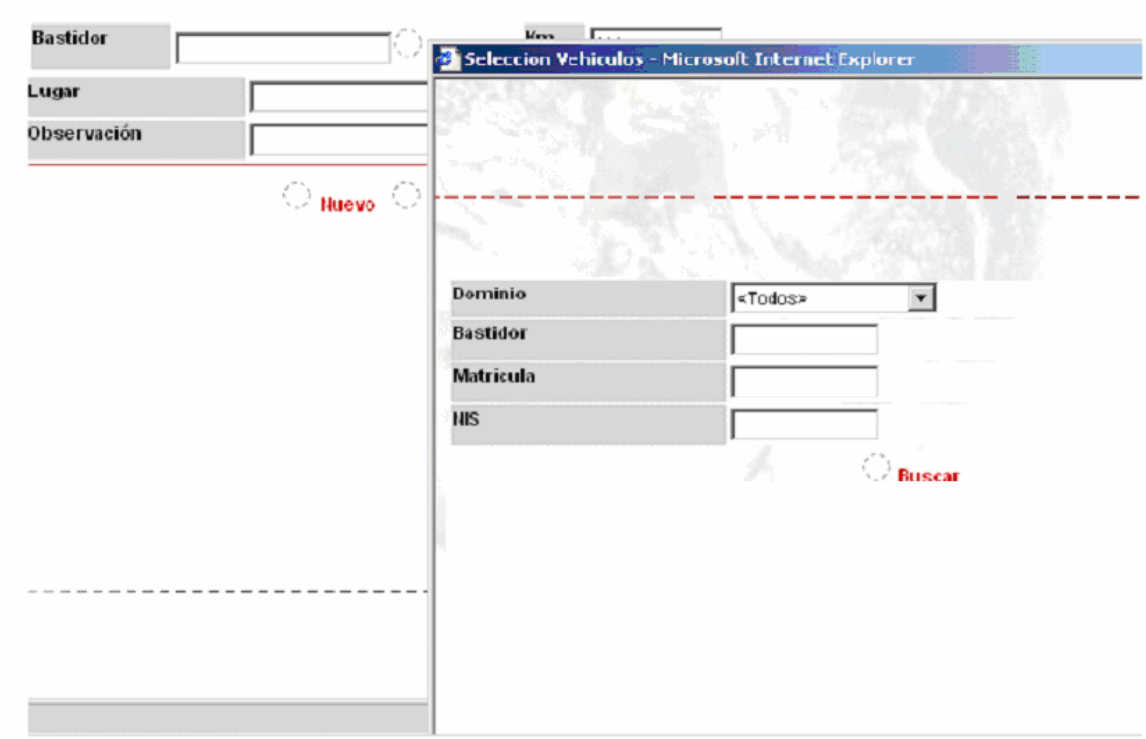


Figura 16-5. Servicio Presentación: Tags - Selección valor ventana

En el ejemplo mostrado, al clickar sobre el enlace 'AABBCC232', éste valor se copiaría sobre el campo 'Bastidor' de la ventana origen. En caso de que esta misma página se usará en el listado de vehículos, al clickar sobre el enlace 'AABBCC232' se abriría sobre la misma ventana el detalle del vehículo.

16.4.11 Tag Double

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:double>
Ejemplo de uso:	<fwk:double key="field.precioventa.precio" property="precio" isRequired="true" mode="E,I,I" styleClass="LABEL" size="10" maxlength="10"

Este tag es simplemente un tag TextField con la particularidad de que pone por defecto el valor de conversión a double.

16.4.12 Tag TextArea

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:textarea>
Ejemplo de uso:	<fwk:textarea key="field.dictamen.observaciones" rows="5" property="observaciones" isRequired="false" mode="E,E,I" styleClass="LABEL" size="40" maxlength="250"/>

El tag textarea, a diferencia del tag básico de Layout o del propio Struts, realiza 2 aspectos diferenciadores:

- A partir del atributo maxlength comprueba que no se exceda el máximo de caracteres definido (el componente TEXTAREA de Html no realiza este comportamiento)
- Para las rows definidas, en el momento en que el contenido que escribe el usuario excede este número de filas, se muestra una SCROLLBAR vertical automáticamente
- En caso de que se especifique (se puede desactivar, si se cree conveniente), aparece un componente adicional al lado del textarea que muestra cuantos caracteres quedan por introducir para llegar al máximo

Atributo	Descripción
maxlength	Número máximo de caracteres a introducir
size	Longitud del componente
rows	Filas a mostrar
key	Clave del fichero de recursos que indica la etiqueta que aparecerá a la izquierda del componente de texto
property	Nombre del campo input y el nombre de la propiedad correspondiente al ActionForm
isRequired (opcional)	Indicar 'true' para indicar al usuario que la entrada de datos es obligatoria. Se mostrará una imagen a la derecha del componente de texto (actualmente un asterisco en rojo) cuando la entrada esté vacía. En el momento que el usuario introduzca datos y cambie de componente, desaparece el símbolo que indica obligatoriedad Por defecto: false Si se requiriera cambiar la imagen que muestra cuando un campo es obligatorio, podemos reemplazar el fichero *layouts\images\last.gif*

```
<!-- observaciones -->
<fwk:textarea key="field.dictamen.observaciones" rows="5"
property="observaciones" isRequired="false"
mode="E,E,I" styleClass="LABEL" size="40" maxlength="250"/>
```

Código 11 Ejemplo de uso del tag textarea

En la variable '**textarea.useCounterField**' del fichero '**Struts-Layout.properties**' podemos especificar 'true' o 'false' para indicar si queremos que aparezca o no el componente adicional que muestra cuantos caracteres le quedan al usuario para llegar al máximo permitido.

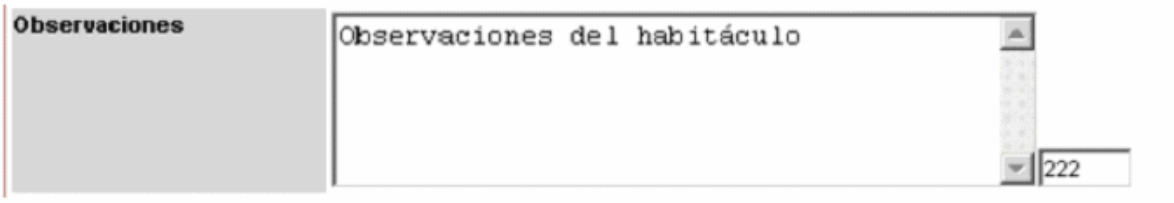


Figura 16-6. Uso del contendor textArea

16.4.13 Tag Fecha

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:date>
Ejemplo de uso:	<fwk:date key="field.vehiculo.fechaFinFabricacionEsperada" property="fechaFinFabricacionEsperada" styleClass="LABEL" size="10" maxlength="10"/>

El tag fecha muestra un campo de entrada de texto y adicionalmente un botón que permite seleccionar de un calendario la fecha a escoger.

- Entrada Manual. El usuario introduce la fecha en el campo texto

El Sistema realiza una conversión automática según el formato definido en el fichero de recursos de mensajes en la clave 'org.apache.struts.taglib.bean.format.date'.
Ejemplo: org.apache.struts.taglib.bean.format.date = dd/MM/yyyy

Figura 16-7. Tag Fecha: Conversión Automática

Así, si el usuario entra '07102002' el sistema añadirá automáticamente el separador y lo convertirá a '07/10/2002'.

- Automática. El usuario selecciona la fecha del calendario

En este caso, el sistema, a partir del formato de fecha definida por el fichero de recursos y la fecha seleccionada introduce en el componente texto la fecha correctamente formateada

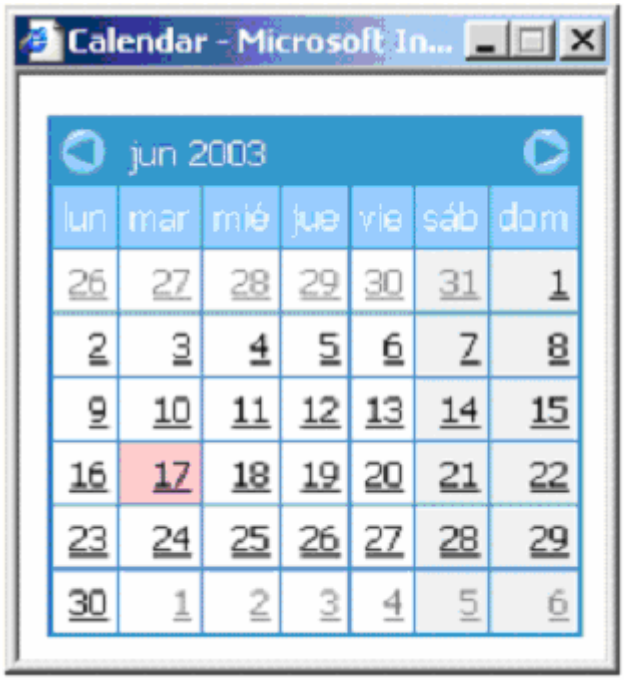


Figura 16-8 Tag Fecha: Selección de la fecha en el calendario

Atributo	Descripción
key	Clave del fichero de recursos que indica la etiqueta que aparecerá a la izquierda del componente de texto
property	Nombre del campo input y el nombre de la propiedad correspondiente al ActionForm
styleClass (opcional)	Clase estilo asociada al componente. Por defecto, el estilo que el framework usa es 'LABEL'
size	Tamaño visible del campo
maxlength	Número máximo de caracteres introducibles en el campo
isRequired	Obligatoriedad del campo

16.4.15 Tag Radios

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:radios> (usar en conjunción con <fwk:options> y <fwk:option>)
Ejemplo de uso:	<fwk:radios key="field.incidencia.idTipoIncidencia" property="idTipoIncidencia" mode="E,E,I" styleClass="LABEL" cols="3"> <fwk:options property="codigo" labelProperty="descripcion" collection="<%= Constants.LIST_TIPOSINCIDENCIAOPTIONS_KEY %>"/> </fwk:radios> **Tipo Incidencia** ◉ Averias ○ Siniestros ○ Multas

El tag radios funciona en conjunción con el tag Options y el tag Option. Estos 2 tags permiten generar la colección de valores que aparecerán. El valor que nos venga en el property del bean asociado al formulario determinará cual es el valor seleccionado de la lista de opciones.
Los atributos que permite definir son los comunes a todos los componentes y adicionalmente:

Atributo	Descripción
cols (opcional)	Número de columnas en el que representaremos los valores. Por defecto cols=1

Normalmente, el uso de radios vendrá condicionado por el uso interno de options u option. Por lo general, usaremos radios cuando conozcamos la lista de posibles valores, mientras que usaremos select cuando no conozcamos la lista de posibles valores o estos provengan de una tabla de la base de datos.

16.4.16 Tag Options

Uso Librería:	<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>
Definición tag :	<fwk:options>

Este tag genera una colección de elementos <option> que representan las opciones del select o del radios que contiene al tag (su funcionamiento es el mismo que el proporcionado por Struts).

Este tag funciona de 2 modos, dependiendo de si especificamos el atributo ‘collection’ o no lo especificamos. El caso más general es hacer uso de ‘collection’, para lo cual indicamos los atributos:

Atributo	Descripción
collection	Nombre del bean que se encuentra en algún scope y que contiene la colección de beans, uno por opción a mostrar
property	Nombre del atributo de cada bean de la colección que será usado en el option como código (el que se enviará en el submit)
labelProperty	Nombre del atributo de cada bean de la colección que será usado como etiqueta del option

Ejemplo

Página JSP – Uso Tag

```
<fwk:select key="field.prestamomedio.idMedio" property="idMedio" mode="E,I,I"
styleClass="LABEL" optionSelectOne="true" >
<fwk:options       property="codigo"       labelProperty="descripcion"       coll
Constants.LIST_MEDIOSCOMUNICACIONOPTIONS_KEY %>"/
</fwk:select>
```

Creación de la Colección desde la clase Action

```
ArrayList tmpList = null;
tmpList = (ArrayList) anCatalogBO.getComboVO(MedioComunicacionVO.class,"idMedio",
"nombre");
request.setAttribute(Constants.LIST_MEDIOSCOMUNICACIONOPTIONS_KEY, tmpList);
```

Clase Bean que representa cada uno de los elementos de la colección

```
package com.juoc.util;
/**
* Objeto generico que contiene dos Strings (clave y valor)
* Uso: Selects de las páginas JSP
*/
public class KeyValueObject implements java.io.Serializable {
private String codigo, descripcion;
/** Creates new KeyValueObject */
public KeyValueObject() {
}
/** Creates new KeyValueObject */
public KeyValueObject(String aCodigo, String aDescripcion) {
setCodigo(aCodigo);
setDescripcion(aDescripcion);
}
/**
* Gets the codigo
* @return Returns a String
*/
public String getCodigo() {
return codigo;
}
/**
* Sets the codigo
* @param codigo The codigo to set
*/
public void setCodigo(String codigo) {
this.codigo = codigo;
}
/**
```

```
* Gets the descripcion
* @return Returns a String
*/
public String getDescripcion() {
return descripcion;
}
/**
* Sets the descripcion
* @param descripcion The descripcion to set
*/
public void setDescripcion(String descripcion) {
this.descripcion = descripcion;
}
}
```

Código 16-12 Ejemplo de uso del tag optio

En el tag ejemplo mostrado, se indica que se usará la colección definida en el nombre de bean definido como 'Constants.LIST_MEDIOSCOMUNICACIONOPTIONS_KEY'. Para los elementos de la colección se usará el atributo 'codigo' como valor a enviar, y el atributo 'descripcion' como texto a mostrar en la opción.
Los elementos de la colección introducidos son objetos de la clase KeyValueObject. Esta clase es proporcionada por el framework para diferentes usos, pero su uso principal es el de opciones en los Select.
En el ejemplo, la clase Action se encarga de preparar la colección y ponerla en el contexto 'request', aunque podría dejarla tanto en 'session' como en 'page'.

16.4.17 Tag Option

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:option>

Este tag permite crear una opción en un Select. Su uso, en comparación con el tag options es que permite crear las opciones una a una desde el código.

Atributo	Descripción
key	Descripción a mostrar para la opción
value	Valor clave a usar (lo que se enviará en el submit del formulario)

El uso de uno u otro tag podemos definirlo según el siguiente criterio:
Si la lista de valores a mostrar es una lista dinámica que depende de la base de datos (pueden añadirse nuevos valores, eliminar, etc.) usaremos el tag 'options', si por el contrario la lista de valores a mostrar está determinada usaremos el tag 'option' (uno para cada valor).

16.4.17 Tag Checkbox

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:checkbox>
Ejemplo de uso:	<fwk:checkbox key="field.vehiculo.atipico" property="atipico" mode="E,E,I" styleClass="LABEL" />

Muestra un elemento HTML de tipo checkbox.
La propiedad del ActionForm asociada ha de ser de tipo **boolean**. Para los demás tags la propiedad del ActionForm será de tipo 'String'.

Atributo	Descripción

key	Valor clave a usar
value (opcional)	Valor que se enviará si se marca el checkbox.. Por defecto se envía el valor 'on'. Posteriormente este valor, por arquitectura se transforma en **true** en la propiedad del formulario.

Cuando un checkbox es desmarcado su valor no es enviado. Esta restricción es simplemente por definición del protocolo HTML y http. Por este motivo, es muy importante, seguir la siguiente estrategia:

1) Si el formulario se guarda en el contexto request, es suficiente con que el valor del atributo sea false en el constructor de la clase
2) Si el formulario se guarda en el contexto session, definir en el método reset la inicialización del valor a false

16.4.18 Tag Img

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:img>
Ejemplo de uso:	<fwk:img pageKey="images.template.logos.seat"/> En el fichero de recursos: images.template.logos.seat = /template/images/logo_seat.gif

El tag img tiene como función generar un <img> de html.
El atributo de mayor uso es '**pageKey**', el cual define mediante una clave del fichero de recursos qué ruta de imagen usar, ruta considerada desde 'WEB-INF'.

16.4.19 Tag Link

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:link>
Ejemplo de uso:	<fwk:link page="/mantPrestamoMedio.do?reqCode=edit" name="mapa" scope="page"> <bean:write name="elemento" property="vehiculo.bastidor"/> </fwk:link>

El tag link es equivalente a Struts con 2 diferencias importantes:

- Si clickamos al link y tenemos algún formulario en el que el usuario haya hecho cambios, se nos avisa de la pérdida de datos (se puede configurar para que no haga comprobación alguna)
- Si el link se encuentra en una ventana abierta desde otra, el comportamiento que se realiza es copiar uno de los valores de los parámetros de la URL del link sobre un campo de la ventana que ha abierto la ventana contenedora del link

Atributo	Descripción
popupField (opcional)	Nombre de uno de los parámetros de los pasados al link (en el atributo 'name'), cuyo valor se copiará a un textfield si este ha especificado la propiedad popupReqCode
checkFormChange (opcional)	Indicar true, si se quiere avisar al usuario de que ha realizado cambios a algún formulario y los perderá. Por defecto 'false'.

El paso de parámetros en el link puede realizarse de 2 formas dependiendo de si queremos pasar un único parámetro o varios parámetros:

• Varios parámetros: Usando el atributo ‘name’ con ‘scope’
En este caso, debemos definir un Hashtable en el que pondremos los parámetros y sus valores y dejarlo en pageContext.
En el tag debemos indicar como **name** el nombre de atributo que hemos indicado para la Hashtable en pageContext y en **scope** indicar ‘page’

```
<%
java.util.Hashtable mapa = new java.util.Hashtable();
mapa.put("bastidor",
((com.test.sigeva.general.model.PrestamoMedioVO)elemento).getBastidor());
mapa.put("idMedio",
((com.test.sigeva.general.model.PrestamoMedioVO)elemento).getIdMedio());
pageContext.setAttribute("mapa", mapa);
%>
<%-- bastidor --%>
<tabla:define id="bastidor">
<fwk:link page="/mantPrestamoMedio.do?reqCode=edit" name="mapa" scope="page">
<bean:write name="elemento" property="vehiculo.bastidor"/>
</fwk:link>
</tabla:define>
```

Código 6-13 Link con paso de parámetros

• Un solo parámetro: Usando los atributos ‘paramId’, ‘paramName’ y ‘paramProperty’

```
<%-- idIncidencia --%>
<tabla:define id="idIncidencia">
<fwk:link page="/mantIncidencia.do?reqCode=edit" paramId="idIncidencia"
paramName="elemento" paramProperty="idIncidencia">
<bean:write name="elemento" property="idIncidencia"/>
</fwk:link>
</tabla:define>
```

Código 16-14 Link con paso de un solo parámetro

16.4.20 Tag Imagen Texto Link

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:imglink>
Ejemplo de uso:	<fwk:imgLink page="/logoff.do" pageKey="images.template.button" keyText="buttons.text.logoff" alignText="right" styleText="textImage"/> **Logoff**

Este tag es una combinación del tag ‘link’ con el tag ‘img’, de forma que permite indicar una imagen y un texto y los 2 formarán parte del link.

16.4.21 Tag Password

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:password>

Ejemplo de uso:	<fwk:password key="field.password"isRequired="true" property="password"styleClass="LABEL" size="10" maxlength="10"/>

Tag que genera el código equivalente a un input type=password.

16.4.22 Tag File

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:file>
Ejemplo uso:	<fwk:file property="fitxer" mode="E,E,E"key="field.importacioFitxers.fitxer" styleClass="LABEL" isRequired="true"/>

Tag que genera el código equivalente a un input type=file.

16.4.23 Tag Modo

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:mode>

El tag de modo permite indicar que una región de la página JSP solo será generada para un modo específico de formulario.

```
<fwk:mode value="edit">
<!-- *************** LIBRO ABORDO ************************* -->
<layout:panel key="panels.libroAbordo" styleClass="FORM" width="100%">
<fwk:file property="libroAbordoFile" />
</layout:panel>
</fwk:mode>
```

Código 16-15 Ejemplo de uso del tag mode

En el caso mostrado, solo se mostrará el componente para adjuntar un fichero si el formulario se encuentra en modo edición.

Atributos posibles del tags

Atributo	Descripción
value	Para qué valor de modo de formulario se muestra el código contenido entre el inicio de tag y fin de tag Valores posibles: **edit**, **create**, **inspect**

16.4.24 Tag Línea

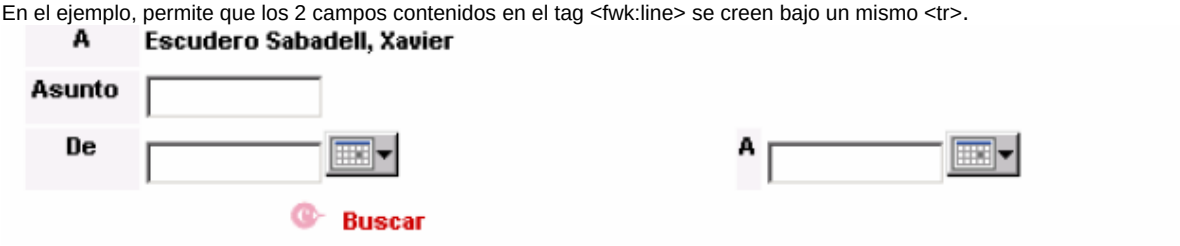
Como se ha comentado en el tag <fwk:form> éste genera de inicio y fin un tag html 'table' y los componentes contenidos generan '<tr><td>'. Esta particularidad que permite poder despreocuparse de la creación del layout de los componentes, hace que los componentes se ubiquen automáticamente uno por fila.
Si queremos en una misma fila mostrar varios componentes haremos uso del tag <fwk:line>.

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:line>

```
<fwk:text key="field.avis.titol" property="titol"
isRequired="false" mode="E,E,E" styleClass="LABEL" size="10" maxlength="10"/>
<fwk:line>
<fwk:date key="field.avis.usuariOrig" property="dataInf"
isRequired="false" mode="E,E,E" styleClass="LABEL"
size="10" maxlength="10"/>
<fwk:date key="field.avis.usuariDest" property="dataSup"
isRequired="false" mode="E,E,E" styleClass="LABEL"
size="10" maxlength="10"/>
</fwk:line>
```

Código 16-16 Código ejemplo de uso del tag line

En el ejemplo, permite que los 2 campos contenidos en el tag <fwk:line> se creen bajo un mismo <tr>.



16.4.25 Tags de Menú

Los tags de menú permiten generar de manera dinámica un menú con las opciones de la aplicación a las que un usuario tiene acceso. Para ello, hacen uso de la capa de seguridad del framework y si la url asociada a la opción está permitida para el usuario conectado, la opción aparecerá, en caso contrario no aparecerá en el menú.
El Framework proporciona 2 tipos de tags, que difieren en la presentación, pero no en la forma de definición de los menús:

- Tag Folders. Se presentan los menús con iconificación [+] para abrir y cerrar submenús
- Tag Expandable. Se presentan los menús con bordes, y al pulsar una opción se expande hacia abajo para presentar sus opciones.

Para definir los menús (y hacer uso de cualquiera de los 2 tags proporcionados), es necesario definir un fichero XML con el formato expuesto a continuación:

```
<menus>
<menu key="menu.usuarios.misDatos" url="/mantUsuario.do?reqCode=edit.Self"/>
<menu key="menu.administracion">
<menu key="menu.dominios" url="/listDominio.do?reqCode=newSearch">
<menu key="menu.alta" url="/mantDominio.do?reqCode=create"/>
</menu>
<menu key="menu.grupos" url="/listGrupo.do?reqCode=newSearch">
<menu key="menu.alta" url="/mantGrupo.do?reqCode=create"/>
</menu>
<menu key="menu.usuarios" url="/listUsuario.do?reqCode=newSearch">
<menu key="menu.alta" url="/mantUsuario.do?reqCode=create"/>
</menu>
</menu>
</menus>
```

Código 16-17 Definición de un fichero XML de menú

Para cada opción de menu se define un tag <menu> con la siguiente información:

- key. Clave del fichero de recursos para mostrar el literal de la opción
- url (opcional). Url a la que acceder cuando se pulse la opción. En caso de haber indicado url desplegará las opciones contenidas
- tags <menu> anidados. Permite definir que un menú contiene otros menús.

Tag Menú Folders

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:menuFoldersFromXML>
Ejemplo de uso:	<fwk:menuFoldersFromXML xmlFilename="/WEB-INF/menu.xml"/>

Atributo	Descripción
xmlFilename	Ruta al fichero xml que contiene las definiciones de los menús

```
<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>
<fwk:html>
<fwk:menuFoldersFromXML xmlFilename="/WEB-INF/menu.xml"/>
</fwk:html>
```

Código 16-18 Ejemplo de incorporación del tag menú a la página de menú

Tag Menú Expandable

Uso Librería:	*<%@ taglib uri="/WEB-INF/framework.tld" prefix="fwk" %>*
Definición tag :	<fwk:menuFromXML>
Ejemplo de uso:	<fwk:menuFromXML xmlFilename="/WEB-INF/menu.xml"/>

Atributo	Descripción
xmlFilename	Ruta al fichero xml que contiene las definiciones de los menús

Capítulo 17.

17. Servicio de Validación

Cuando se realiza una aplicación, existe siempre inherente 2 tipos de validaciones:

- Validación de datos

  Incluye el tratamiento de la entrada correcta de tipo de datos, en la que no influye el estado de la aplicación.

  Esta validación debe tener en cuenta, entre otros:

  a) Si un campo es obligatorio
  b) Si un campo cumple una expresión regular (formato de nif, código postal ,etc.)
  c) Otras validaciones

- Validación funcional

  Es aquella validación que depende del estado de la aplicación, y de los datos de negocio.

  Esta validación queda fuera del ámbito de este proyecto, pues su tratamiento se realiza mediante el lanzamiento y tratamiento de excepciones de negocio.

De ahora en adelante, únicamente nos referiremos a la validación de entrada de datos. Este tipo de validación se resuelve en el framework mediante el uso de 'Struts Validator'. Éste permite poder definir mediante el uso de XML cómo deben realizarse las validaciones de forma dinámica, sin que por ello deba codificarse ningún código javascript en nuestras páginas JSP (de este forma obtenemos un código mucho más claro y una desconexión entre la capa de presentación y de validación).
Lo más importante del uso de la validación es que ésta se realiza tanto en la parte cliente como en la parte servidor sin tener por ello qué codificar 2 veces. La importancia de que se realice la validación tanto en la parte cliente como en la servidora reside en qué en el navegador se puede desactivar el uso de JavaScript y eso podría conllevar a problemas si no se realizase de nuevo la validación en la parte servidor. Ahora bien, esta ventaja queda reducida en el framework, pues en caso de que el usuario desactivara el uso de Javascript no sería posible el uso de determinados servicios y envío de formularios.

17.1 Configuración Inicial

Para poder usar la validación, es necesario que en el fichero /WEB-INF/struts-config.xml se configure un nuevo plug-in según se muestra en el siguiente código:

```
   <?xml version="1.0" encoding="ISO-8859-1" ?>
   <action-mappings>
   <!-- ========== Plug Ins Configuration ================================== -->
   <!-- Add multiple validator resource files by setting the pathname property -->
   <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathname" value="/WEB-INF/validator-rules.xml"/>
   <set-property property="pathname" value="/WEB-INF/validation.xml"/>
   </plug-in>
</struts-config>
```

Código 17-1 Configuración plugin validación

El primer fichero (validator-rules.xml) define las validaciones genéricas que pueden realizarse en el framework: requerido, expresión regular, rango, etc.
El fichero 'validation.xml' define las validaciones a realizar para cada uno de los formularios a validar en la aplicación.

17.2 Formularios

Dependiendo de si usamos formularios dinámicos o no, la declaración del formulario será diferente. La clase del framework BaseActionForm, extiende

17.2.1 Formulario Estático

Si usamos formularios no dinámicos (método recomendado para compatibilidad con los servicios del framework), nuestras clases formulario deben heredar de la clase 'com.juoc.struts.base.BaseActionForm'.

```
public class VehiculoActionForm extends BaseActionForm implements java.io.Serializable
```

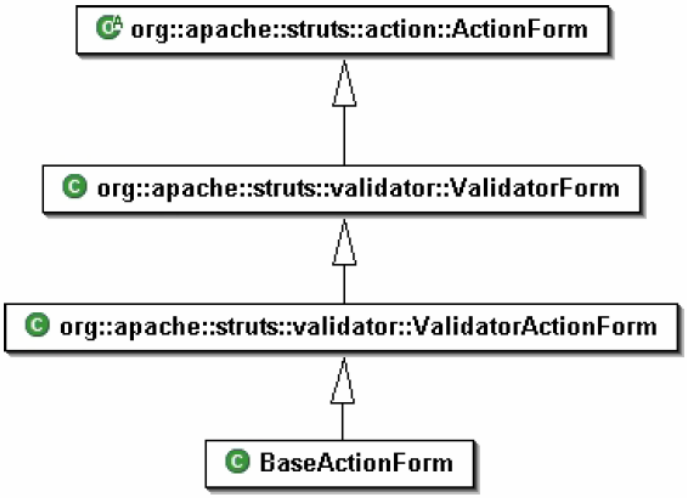


Figura 17-1 Jerarquía del ActionForm del framework (BaseActionForm)

17.2.2 Formulario Dinámico

Los formularios dinámicos definidos deben tener como type 'org.apache.struts.validator.DynaValidatorForm'.

```
<form-bean
name="nombreFormulario"
dynami
c="true" type="org.apache.struts.validator.DynaValidatorForm">
<form-property ....>
...
</form-bean>
```

No se recomienda su uso.

17.3 Configuración de Validadores

El framework proporciona en el fichero 'validation-rules.xml' la generación del código para cada tipo de validación posible.
Actualmente se ofrecen entre otros:

- required. Para generar el código necesario de comprobación de obligación de un campo
- mask. Para general el código necesario de comprobación según expresión regular.
- etc.

En caso de que se necesite crear un nuevo código de validación, es conveniente definir un nuevo tipo. Un ejemplo claro es la comprobación de igualdad de valores entre 2 campos, cuando el usuario introduce el nuevo password y debe reintroducir otra vez el nuevo password, donde estos 2 deben coincidir para continuar con el proceso.

17.4 Definición de las validaciones a realizar para los formularios

En el fichero validation.xml tenemos 2 secciones ampliamente diferenciadas que se engloban bajo el tag raíz <form-validation>:

- Definición de constantes

  En esta sección se definen constantes (expresiones regulares, por lo general) que pueden ser accedidas desde cualquier punto de las definiciones de los formularios.

- Definición de los formularios a validar

  En esta sección se especifican qué validaciones realizar para cada campo de cada uno de los formularios.

```
<form-validation>
   <global>
         <constant>
               <constant-name>anyFiscal</constant-name>
               <constant-value>^([0-9][0-9][0-9][0-9])$</constant-value>
         </constant>
   </global>
   <formset>
         <form name="avisForm">
               <field property="titol" depends="required">
                     <arg0 key="field.avis.titol"/>
               </field>
               …
         </form>
   </formset>
</form-validation>
```

Código 17-2 Esquema definición validaciones

Validación según idioma

En caso de que se requieran de diferentes tipos de validación según el idioma escogido (por ejemplo debido a fechas, formatos de moneda, etc.), es necesario definir varios formsets..

Para especificar un conjunto de validaciones para formularios generales sin tener en cuenta el idioma:

```
<formset>
      Especificación de cada formulario (ver punto 7.3)
</formset>
```

Ahora bien, para especificar validaciones específicas de un idioma:

```
<formset language="xx">
</formset>
```

, donde ‘xx’ representa la codificación ISO del lenguaje según el estándar especificado en:

http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt

Además, si queremos poder especificar un país dentro del lenguaje la sintaxis sería:

```
<formset language=”xx” country=”yy”>
</formset>
```

, donde ‘yy’ representa la codificación ISO del país según el estándar especificado en:

http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

17.4.1 Declaración de los Formularios

Para cada formulario a validar debemos definir un tag <form> con el siguiente atributo:

name	Nombre del formulario tal y como se ha definido en el fichero struts-config.xml

Internamente a él, definiremos un tag <field> para cada campo a validar.

field

property	Nombre del campo del formulario

depends

Especifica las validaciones que deberá superar el valor del campo para ser considerado válido.

- required. El campo es obligatorio, no puede ser vacío
- date. El valor del campo es una fecha
- mask. El valor del campo debe cumplir la expresión regular especificada en la variable mask (<var-name>mask<var-name>)
- integer. El valor del campo debe ser un número entero
- range. El valor del campo debe estar comprendido dentro de un cierto rango
- minlength. Mínima longitud que debe tener el campo
- maxlength. Máxima longitud que puede tener el campo
- byte
- short
- long
- double
- range
- email
- creditcard

```
<field property="idImpost" depends="required">
<arg0 key="field.calculQuotaCambra.impost"/>
</field>
```

Además, se puede especificar una dependencia de validación que se haya definido en algún constant del apartado <global>. Ejemplo: depends='matricula'

arg0...arg N

Especifican mediante el atributo 'key' el valor a utilizar en cada uno de los argumentos del mensaje que se mostrará en caso de que el campo no sea correcto.

Para 'required', 'short', 'byte', 'integer', 'long', 'double', 'float', 'date', 'email', 'creditcard', indicaremos únicamente 'arg0', y para su atributo 'key' el mensaje a incorporar

```
<field property="idImpost" depends="required">
<arg0 key="field.calculQuotaCambra.impost"/>
</field>
errors.required={0} es un campo obligatorio
```

En este caso, se mostraría 'Impost es un campo obligatorio'.

Para 'minlength','maxlength' sus mensajes vienen definidos por 2 argumentos, uno que indique el campo y otro que indique la longitud mínima o máxima que debe cumplirse.

```
<field property="anyFiscal" depends="required, integer, maxlength,
minlength">
<arg0 key="field.calculQuotaCambra.anyFiscal"/>
<arg1 name="maxlength" key="${var:maxlength}" resource="false"/>
<arg1 name="minlength" key="${var:minlength}" resource="false"/>
<var>
<var-name>maxlength</var-name>
<var-value>4</var-value>
</var>
<var>
<var-name>minlength</var-name>
<var-value>4</var-value>
</var>
</field>
```

En este caso, anyFiscal debe cumplir una longitud mínima de 4 y una longitud máxima de 4. Para ello en el arg1 se han especificado para la longitud mínima:

```
<arg1 name="minlength" key="${var:minlength}" resource="false"/>
```

Mediante el atributo name estamos haciendo referencia a que el argumento 1, se refiere a la dependencia 'maxlength'. Esto permite que podamos tener varias dependencias con varios argumentos 1, uno para cada tipo de dependencia.

Con '${var:minlength}' se está indicando que el valor del argumento del mensaje viene definido por la variable contenida en la definición (<var></var>) con nombre 'minlength'; y el atributo 'resource="false" ' está indicando que no debe ir al fichero de recursos para obtener el literal, sino que la propia key define el literal.

En el caso de 'range', tendremos 3 argumentos, con la misma filosofía que la definida arriba en el caso de 'minlength' y 'maxlength'.

```
<field property="anyFiscal" depends="required, integer, maxlength,
minlength">
<arg0 key="field.calculQuotaCambra.anyFiscal"/>
```

```
<arg1 name="maxlength" key="${var:maxlength}" resource="false"/>
<arg1 name="minlength" key="${var:minlength}" resource="false"/>
<var>
<var-name>maxlength</var-name>
<var-value>4</var-value>
</var>
<var>
<var-name>minlength</var-name>
<var-value>4</var-value>
</var>
</field>
```

Los mensajes a mostrar se definen en el fichero de recursos:

```
##############################################################################
# Mensajes validacion (validation-rules.xml)
##############################################################################
errors.byte={0} tiene que ser un número
errors.creditcard={0} no es un número de tarjeta de crédito válido
errors.date={0} no es una fecha correcta
errors.double={0} tiene que ser un número
errors.email={0} no una dirección de correo válida
errors.float={0} tiene que ser un número
errors.integer={0} tiene que ser un número
errors.invalid={0} es inválido
errors.long={0} tiene que ser un número
errors.maxlength={0} no puede tener un tamaño mayor que {1} caracteres
errors.minlength={0} no puede tener un tamaño menor a {1} caracteres
errors.range={0} no se encuentra en el rango permitido de {1} a {2}.
errors.required={0} es un campo obligatorio
errors.short={0} tiene que ser un entero
```

Código 17-3 Mensajes multiidioma de validación

17.4.2 Declaración de Validaciones Generales

En la sección <global> podemos definir constantes que se reutilicen en el fichero de validación.
El uso principal de esta sección es definir nombres de constantes para expresiones regulares a comprobar para varios campos de la aplicación (matrículas de vehículos, códigos postales, etc.).

```
<form-validation>
<!--
Declaraciones globales de expresiones regulares que se utilizaran mas ade
-->
<global>
<constant>
<constant-name>matricula</constant-name>
<constant-value>(^([A-NO-Za-no-z]){1,2}-([0-9]){4}-([A-NO-Za-no-z])
{1,2}$)|(^([A-NO-Za-no-z]){1,2}([0-9]){4}
([A-NO-Za-no-z]){1,2}$)
</constant-value>
</constant>
<constant>
<constant-name>anyFiscal</constant-name>
<constant-value>^([0-9][0-9][0-9][0-9])$</constant-value>
</constant>
..
</global>
```

Código 17-4 Ejemplo de definición de una constante expresión regular

Posteriormente, desde cualquier formulario puede definirse para un determinado campo:

```
<.... depends="${nombreConstante}">

 <form name="searchPlantillaForm">
 <field property="anyFiscal" depends="mask">
 <arg0 key="field.plantilla.anyFiscal"/>
 <var>
 <var-name>mask</var-name>
 <var-value>${anyFiscal}</var-value>
 </var>
 </field>
</form>
```

Código 17-5 Ejemplo de uso de constantes de validación desde un formulario

17.5 Código de validación en la página JSP

Con el código especificado en el punto anterior estamos indicando cómo debe generarse dinámicamente el código de validación Javascript en el cliente y Java en el servidor. Ahora bien, en la página JSP es necesario indicar la llamada al código de validación creado antes de enviar el formulario.

```
<body>
...
<html:form action="procesarFormulario" onsubmit="return validateNombreForm(this);
<!-- A partir de aquí es donde se generará el código JavaScript de validación del ->
<html:javascript formName="nombreForm"/>
</body>
```

Código 17-6 Indicación de la validación en la página JSP

Donde nombreForm es el nombre de formulario que se habrá indicado en el procesamiento de la petición dentro del fichero struts-config.xml

Capítulo 18.

18. Aplicación de Ejemplo

Para comprobar como facilita el desarrollo de aplicaciones empresariales el Framework Juoc, se han creado una serie de páginas de lo que sería una aplicación de test que realiza la presentación de árticulos promocinables dentro de folletos publicitarios para la red de departamento de recambios de una cadena de automoviles. Esta aplicación de test consiste en la lista de articulos que tiene un folleto que presenta una promoción con atributos de estos articulos, con pantalla de buscadores, menus, etc.

La finalidad de la aplicación ha consitstido en verificar como de complejo, util, y sencillo resulta la creación de aplicaciones con Juoc.

Se ha hecho uso de la internalización, validación de formularios, tiles para la presentación de marco de trabajo de la aplicación y de los tags de creación de las páginas web.

En la siguiente figura se puede obserbar la distribución de directorios donde se incluyen las clases java (de deficición de VO, excepciones, Forms, Action), páginas jsp, ficheros de recursos de imágenes, properties, javascript, y los ficheros de configuración:

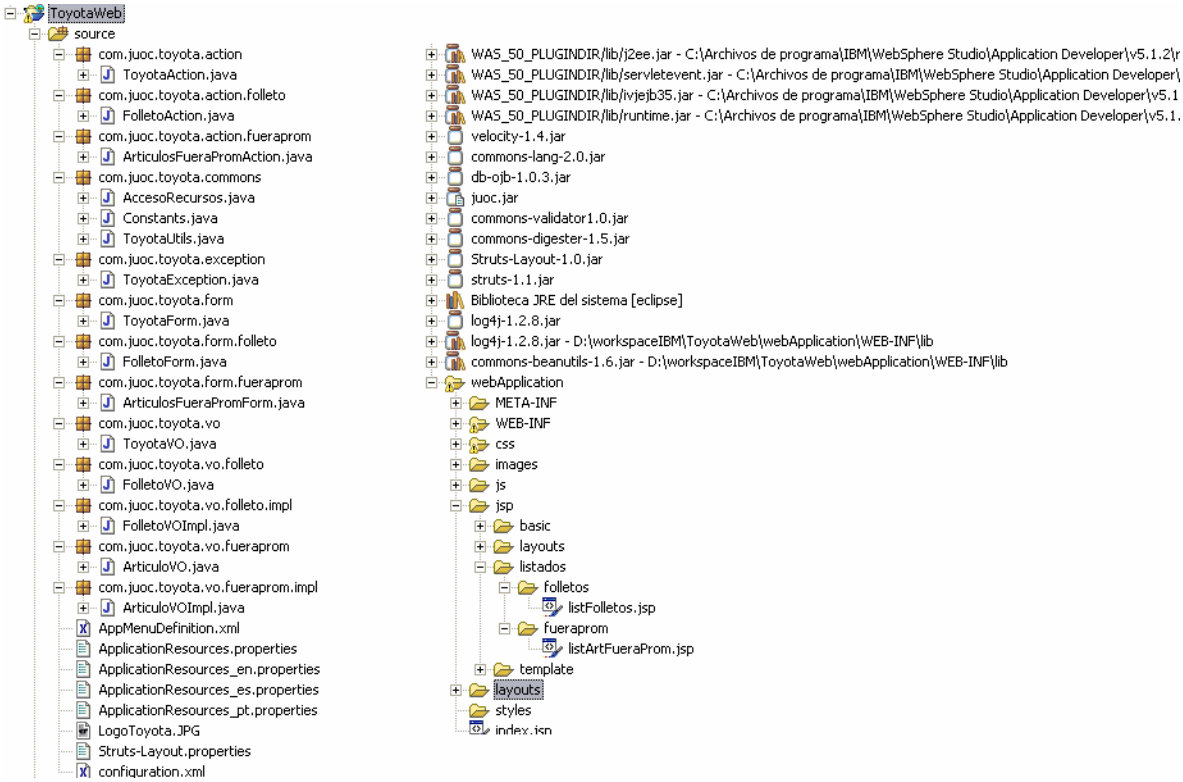


Figura 18-1 Estructura de ficheros de la aplicación de test.

El código que contienen las clases java resulta sencillo. Solo se ha programado lo caracteristico de la propia aplicación la parte del flujo de la aplicaciones lo resuelve Juoc. Se han definido los VO necesarios para tratar la información de los folletos y de los árticulos.

Los jsp que se han creado utilizan los tags del framework con lo que la creación de jsp resulta sencilla el código introducido es poco y facilmente entendible. Las pantallas muestran un aspecto estandar y con calidad en el acabado a base del javascript que incluye la creación de los tags del framework y las hojas de estilos de la aplicación.

La apliación no interactua con ningún sistema gestor de BBDD puesto que las funcionalidades presentandas por el framework no contemplan este ambito, al ser un framework de presentación. Una posibilidad de incluir accesso a BBDD de una forma sencilla consiste en utilizar, ibatis. Ibatis se ocupa de la capa de Persistencia (se sitúa entre la lógica de Negocio y la capa de la Base de Datos). asocia objetos de modelo (JavaBeans) con sentencias SQL o procedimientos almacenados mediante ficheros descriptores XML, simplificando la utilización de bases de datos. La utilización de Ibatis supone incluir los jar que cuelgan del proyecto `org.apache.ojb.`

No se ha entregado el acceso a BBDD dado que por cuestiones de tiempo no se ha podido llegar a hacer funcionar correctamente.

Capítulo 19.

19. Futuras líneas de desarrollo/mejoras en el Framework.

El desarrollo de un Framework de Presentación para usos empresariales comprende un extenso desarrollo de facilidades y servicios. En este proyecto fin de carrera se ha desarrollado una base, que ofrece los servicios básicos y a partir de los cuales se pueden ir añadiendo muchos más. Inicialmente dentro de la arquitectura que se quería desarrollar se incluian los siguientes servicios que no se han podido desarrollar dada la magnitud que tendría el proyecto:

Tiles con Pestañas

Este layout permite el uso de pantallas accesibles mediante pestañas

Tags

Creación de Tags de combos (inclusion de depedencia entre combos para cargar dinamicamente los mismos).

Servicio de Listados

Permite definir informes complejos en formato HTML a partir de un fichero de definición del informe en formato XML. Permite la definición de listados resultado paginados, donde se permite al usuario navegación por página (adelante, atrás, primera, última) y acceso directo a número de página, ordenación ascendente/descendente por cada una de las columnas, y selección del número de filas a mostrar por cada página.

Servicio de Autorización

Permite la entrada del usuario a determinadas URLs, así como los tipos de acceso permitidos en cada una de ellas. En conexión con el mecanismo de tags de presentación muestra la página en un modo de edición o consulta según estos tipos de acceso.

Servicios de trazas

Basado en log4j, resulta util en el seguimiento de los errores y Auditorias. Su configuración se realizaría mediante un fichero donde se permite la de definición de trazas por niveles (DEBUG < INFO < WARN < ERROR < FATAL). La idea sería que se dispusiera de diferentes Tipologías de Salidas de log: base de datos, fichero,consola, correo incluso con posibilidad de salida multiidiomaServicios.

Servicio de Excepciones

La gestión de excepciones es un aspecto muy importante y crítico. Se definría una interface basada en multiidioma con la deficion del patrón de uso de las excepciones: captura nivel inferior y pasar a excepción nivel actual. Así una excepción del Framework permitíria indicar:

a)Causa raíz que ha provocado la excepción (clave del fichero de recursos multidioma). Permite que mediante el Servicio de Log podamos conocer cual ha sido la causa real de la excepción

b)Anidamiento de excepciones. Éstas son las que finalmente por mecanismo de internacionalización se mostrarán al usuario por pantalla.

Tendríamos una presentación internacionalizada de la excepción realizada en la capa de presentación (integración con Struts: paso a ActionErrors).

Como se puede observar estos servicios son extremadamente utiles y necesarios.

Capítulo 20.

20. Conclusiones.

Podemos concluir que la creación de un Framework de Presentación, en este caso para entornos empresariales, es una tarea muy atractiva , pero costosa desde el punto de vista de todo lo que se puede llegar a ofrecer para ayudar en la parte de la presentación de Web Sites.

La parte de análisis de requisitos , es decir lo que queríamos ofrecer, se baso en la idea más sencilla y útil: de la multitud de aplicaciones que se pueden crear, se eligió las aplicaciones empresariales. En este tipo de aplicaciones la linealidad de desarrollos hace sencillo extraer características comunes y por tanto desarrollar herramientas que faciliten estos desarrollos. Aunque las características sean comunes, estandarizar la parte de usabilidad en este tipo de aplicaciones es muy importante y compleja ya que el usuario final de las mismas, las utiliza en el horario de su jornada laboral, es decir, durante muchas horas al día. Esto implica que los interfaces no deben provocar fatiga visual ni mental, el comportamiento en el uso de los mismos debe facilitar al usuario lo máximo posible las tareas laborales que realiza con ellos. La eficiencia, eficacia, estilo, facilidad de uso, coherencia de contenidos etc. es fundamental.
Así en este punto la definición de una buena guía de estilos visuales, como deben ser los componentes, que características deben poseer desde el punto de vista visual son tareas a las que responden departamentos de maquetación. Una vez elegido como tiene que ser el aspecto visual, reuniones con arquitectos técnicos darán las características de comportamiento informático que tendrán dichos componentes, estableciendo sus propiedades, métodos, y uso.
En esta etapa nosotros mismos hemos elaborado una serie componentes, estableciendo su aspecto visual y su comportamiento.

Desde el punto de vista tecnológico la idea de utilizar la tecnología Struts como punto de partida, se baso en lo maduro de la misma. Hemos profundizado mucho en ella, aprendiendo muchísimo sobre como Struts es un buen Framework de la capa de presentación, de ahí que esta tecnología haya sido la opción de desarrollo durante muchos años en el sector empresarial. Struts ha resultado una tecnología fácilmente extensible, su base es idónea para empezar a desarrollar todo el abanico de posibilidades que se nos puedan llegar a ocurrir. De esta manera se han desarrollado un conjunto de tags, ampliado la arquitectura base adaptándola al desarrollo de aplicaciones empresariales, ofrecer servicios configurables mediante ficheros XML (validaciones) etc. Si bien se podría estimar que en las horas de codificación haya existido un porcentaje de desarrollo en el lenguaje Java, como principalmente se ofrecen componentes a la capa visual, el resultado es que extender una tecnología como Struts comprende un gran porcentaje de codificación mediante Javascript embebido en código Java, para poder ofrecer los componentes visuales de JSP. Este tipo de programación embebida puede llegar a resultar compleja y “frustrante” al ser JavaScript un lenguaje de script costoso de debuggear en un entorno embebido.
Por contrapartida el desarrollador que utiliza los tags no tiene que programar nada en JavaScript, mas haya de alguna situación muy especifica propia de su aplicación, pero el control de los componentes, visualización, comportamiento, se encarga el propio Framework a través de las propiedades, métodos de los mismos, y con las hojas de estilos. Descargar al desarrollador de toda la parte de JavaScript es fundamental ya que la capa de codificación en este lenguaje de script tiene debe que ser lo más delgada posible.
Las otras tecnologías estudiadas JSF y Spring podrían haber resultado más sencillas a la hora de crear el Framework, en base a que parece que tienen menos carga de JavaScript.

El entorno de desarrollo escogido ha sido IBM WebSphere Studio Application Developer. Ofrece un entorno de trabajo parecido a Eclipse pero con más pluggings que facilitan las tareas de desarrollo. Aunque ha costado mucho estabilizarlo y en muchas ocasiones los problemas se solucionaban reiniciando el entorno, sin una aparente causa lógica de porque se habían solucionado.

Siempre se ha tenido presente durante el proyecto la magnitud de opciones que se podían ofrecer. De esta manera el enfoque ha consistido en ofrecer un conjunto amplio de funcionalidades, a modo de paquete básico, dado la magnitud de desarrollo que puede conllevar realizar un conjunto de funcionalidades completa. En la última entrega hubo que añadir un tag de menú, y también durante esta entrega se intento entregar el Servicio de Listados que incluía un Tag de Tabla, pero el desarrollo requerido era muy extenso, abandonando el intento de entrega. Durante todo el desarrollo de framework nos hemos enfrentando a decisiones de que hacer y que posponer como punto futuro.
Así el resultado final ha sido un paquete básico en el que se han adaptado las clases de flujo de Struts añadiendo más funcionalidad a las mismas, con servicios de multiidoma, validación, marco de trabajo y un

conjunto extenso de tags para la creación de JSP. Se han desarrollado los servicios de la forma más configurable posible mediante archivos XML, hojas de estilos y extensas propiedades sobre los componentes. Respondiendo de esta manera a los objetivos del proyecto.
La parte de conexión de BBDD ha sido un punto poco tratado que se ha intentado resolver en la última entrega. Para el desarrollo del Framework no es un punto decisivo pero para una ejecución lo más real posible es un punto imprescindible. El Framework Ibatis apareció como una posible solución rápida de conexión pero existieron problemas de configuración que impidieron el arranque del servicio.

Este proyecto fin de carrera ha sido muy ameno, llegar a entender el corazón de una tecnología y adaptarla a las necesidades particulares ha sido muy cautivador. Ser capaz de ofrecer un conjunto de facilidades para que se desarrollen portales Web empresariales siempre ha sido una idea muy atractiva con la que me he sentido muy motivado. A contenido: estudio de tecnologías actuales, propuestas, pensar en maquetacion, en java, en BBDD (en última instancia) , hacer algo nuevo, definir los limites de lo componentes, crearlos, probarlos, afinarlos etc. Aunque siempre ha existido la sensación de tener que dejar algunas cosas por la falta de tiempo , con todo lo que se ha realizado han faltado combos, tablas, las opciones de seguridad, conexión con BBDD etc.

La envergadura del proyecto siempre ha sido un aliciente para el trabajo dedicado, pero ha sido algo infinito que había que acotar. El reto inicial ha sido de una dimensión muy grande, pero anima a abrir líneas de trabajo en el futuro, puesto que se pueden hacer muchas más cosas interesantes.

Bibliografía.

J2EE

Josep Maria Camps i Riba, J2EE Una plataforma de componentes distribuida.pdf, P06/81059/01155
Materiales de la asignatura Ingeniería del software de componentes y de sistemas distribuidos de la OUC:

Patrones de Diseño

Benet Campderrich Falgueras, Patrones de Diseño, P04/81058/00845
Materiales de la asignatura de Ingeniería del Software Orientado al objeto de la UOC.
Sun, Core J2EE Patterns,
[http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html]

Struts

Chuck Cavaness , Jakarta struts.pdf, O'Reilly.

Java Server Faces

Sun, The Java EE 5 Tutorial,
[http://java.sun.com/javaee/5/docs/tutorial/doc/JavaEETutorial.pdf, Capitulo 10]

Spring Web MVC Framework

Rod Johnson, The Spring Framework - Reference Documentation,
[http://www.springframework.org/docs/reference/mvc.html]

Usabilidad

Nielsen, Jakob (2000). Designing Web Usability. New Riders.
Preece, Jenny (1996). Human Computer Interaction. Addison-Wesley.
ISBN: 0201627698
Sun Guía de estilo para la web.
[http://www.sun.com/styleguide/tables/Printing_Version.html]

Arquitectura Struts

[http://struts.apache.org/]

Servicios de Pantallas

Web Tiles Original [http://www.lifl.fr/~dumoulin/tiles/]
Documentación Struts Tiles [http://struts.apache.org/1.2.7/userGuide/dev_tiles.html]

Servicios de Tags

Tags de Layout heredados y redefinidos para el framework [http://struts.application-servers.com/]
Sun Blueprints: Fast Lane Reader [http://java.sun.com/blueprints/patterns/FastLaneReader.html]
MSDN Magazine: Exchanging Data Over the Internet Using XML [http://msdn.microsoft.com/msdnmag/issues/0400/cutting/default.aspx]
XMLHttp en Mozilla y Netscape [http://developer.mozilla.org/en/docs/XML_Extras]

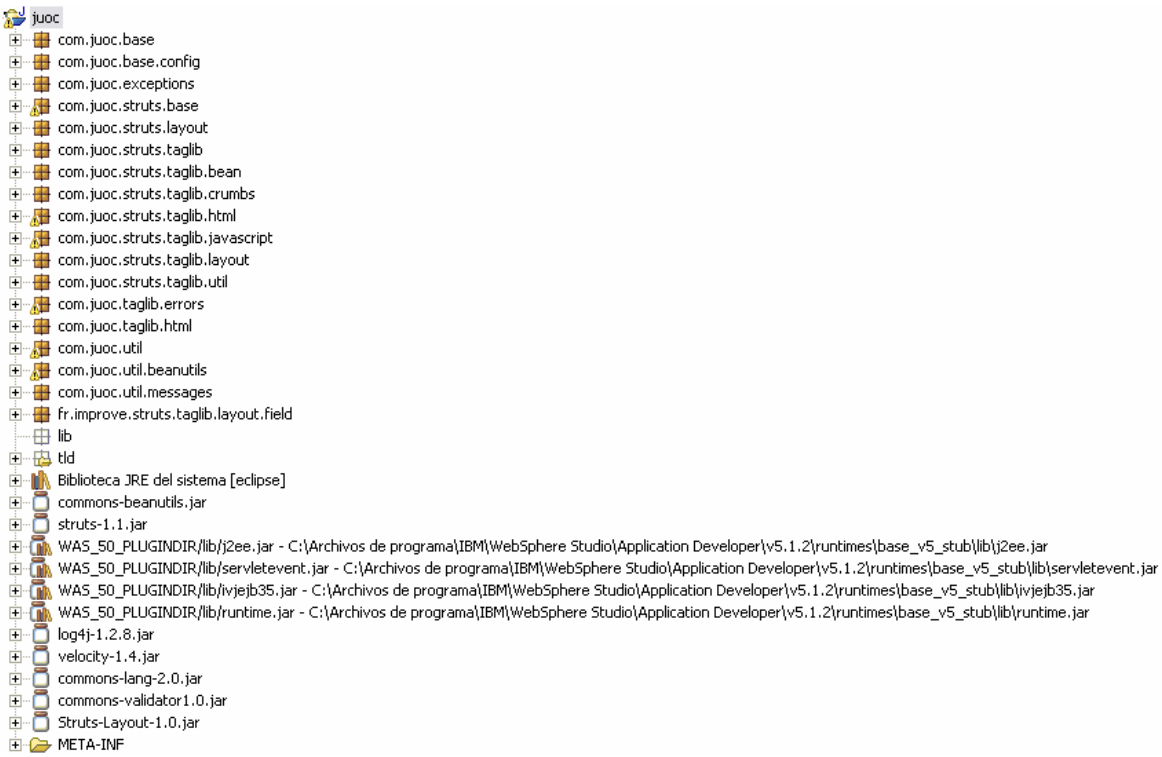
Servicios de Validacion

Documentación Struts Validator [http://struts.apache.org/1.2.7/userGuide/dev_validator.html]

Anexos

En está sección incluimos el código fuente del framework bajo un proyecto de IBM WebSphere junto con los jar de los que depende. También se incluye el codigo empaquetado en forma de jar bajo el fichero juoc.jar .

En la siguiente imagen se puede ver la distribución de las clases que conforman el framework junto con los jar de los que depende:



Tambíen incluimos el EAR de la aplicación de test.