Disseny i implementació d'un marc de treball (framework) de presentació per aplicacions J2EE: Jewel Framework

Juan Manuel López Closa

Enginyeria en Informàtica

Josep Maria Camps Riba

16/01/2012

Dedicatòria i agraïments

Aquest projecte final posa punt i final a la meva etapa com estudiant universitari, que no hauria pogut dur a terme si no fos pel suport d'algunes persones:

En primer lloc, agrair als meus pares el seu suport durant tota la meva vida, sobretot en moments d'exàmens i entregues de treballs. Sense la seva ajuda no podria haver dedicat el temps que he hagut de dedicar a la carrera.

A la meva parella, Maite S., pel seu suport i la seva companyia durant moltes tardes d'estudi conjunt en aquests darrers tres anys.

També als meus amics de sempre, en especial tres d'ells: Lluis S., Joan C. i Miquel A., que han estat amb jo en els moments bons i en els dolents, encara que no sempre els he pogut dedicar tot el temps que es mereixen.

No oblidar-me de les dues empreses per les que he fet feina, SM2 Baleares i Brújula, que han fet possible que creixi com a professional i m'han ajudat a compatibilitzar feina i estudis. En especial, a Dani C., qui em va introduir al món de J2EE i em va dedicar molt de temps a resoldre els meus dubtes de principiant.

Finalment, agrair a tots els professors, tutors i consultors que he tingut tots aquests anys, que m'han donat els consells necessaris per seguir endavant, així com a la UOC, que m'ha permès acabar el cicle superior a la vegada que he fet feina.

Resum del projecte

El present projecte tracta sobre el desenvolupament d'un marc de treball o framework per la capa de presentació d'aplicacions J2EE.

En un primer bloc, s'analitzen els frameworks més utilitzats al món empresarial a dia d'avui: Struts 2, Spring MVC i JavaServer Faces. S'estudia com funcionen i les característiques principals de cada un, comparant-les per veure quines són comuns a tots i es consideren imprescindibles per un framework d'aquest tipus. A més, es fa un anàlisi exhaustiu dels patrons J2EE i les bones pràctiques que poden aportar al projecte.

El segon bloc del projecte és el desenvolupament de Jewel Framework, el marc de treball que s'ha creat. Primer, s'analitza quines són les funcionalitats que ha de complir el framework i es fa un disseny de l'arquitectura i estructura que segueix. Després, s'implementa seguint el disseny i tenint cura que es compleixin tots els requisits, tant funcionals com de qualitat del producte. Finalment, es dissenya i desenvolupa una aplicació d'exemple que demostra la usabilitat i bon funcionament de Jewel.

Paraules clau: j2ee, framework, marc de treball, capa de presentació, mvc, jewel.

Àrea: J2EE

Índex de continguts

Índex de figures

1. Introducció

En el món empresarial cada vegada és més comú trobar-se amb aplicacions web en comptes de les clàssiques aplicacions d'escriptori. Aquestes aplicacions web tenen l'avantatge que, al executar-se sobre un servidor d'aplicacions, el client hi accedeix mitjançant un navegador web, com Internet Explorer o Mozilla Firefox. Així, l'aplicació no depèn del sistema operatiu i, a més, s'hi pot accedir des de qualsevol ordenador que hi tingui permís sense instal·lar cap client.

Entre les diverses plataformes per desenvolupar aquest tipus d'aplicacions destaca Java Enterprise Edition, una solució creada per Sun Microsystems[1]. Java EE té l'avantatge de ser multiplataforma, lliure, tenir una àmplia comunitat d'usuaris que hi treballa i disposar de moltes especificacions com EJB i Servlets.

A l'hora de programar, ja sigui en Java, .NET o altres, és comú l'ús de marcs de treball o frameworks, que són un conjunt de llibreries i utilitats que faciliten i agilitzen el desenvolupament d'aplicacions informàtiques mitjançant l'ús d'un codi de més alt nivell. Així, els programadors i altres membres d'un projecte informàtic poden dedicar més temps a altres tasques com la captura de requeriments.

Una aplicació web es divideix en tres capes:

- Capa de presentació: és la part que veu l'usuari. Aquesta capa s'encarrega de generar la interfície d'usuari a partir de la informació que proporciona la capa de negoci.
- Capa de lògica de negoci: és la part que implementa la problemàtica solucionada per l'aplicació. És la capa intermediària entre les altres dues.
- Capa de persistència o de base de dades: és la responsable de comunicar la capa de negoci amb la base de dades.

El present projecte té per finalitat l'anàlisi, disseny i desenvolupament d'un framework per la capa de presentació d'aplicacions Java EE, el qual agilitzi i simplifiqui el procés de creació d'aplicacions web. Aquest framework estructurarà el codi en tres components: Model, Vista i Controlador, seguint el patró MVC. A més, disposarà d'un conjunt d'utilitats que facilitaran el desenvolupament.

La part visual d'una aplicació web consta de diverses pàgines web que contenen text, formularis, imatges, etc. La feina de programar tot això pot ser tediosa i repetitiva pel programador, per això, el framework intentarà reduir la càrrega d'aquesta part perquè es pugui centrar més en altres aspectes que necessitin de més atenció com la lògica de negoci.

En una primera fase s'avaluaran i compararan els frameworks actuals més utilitzats dins l'àmbit professional: JSF, Struts i Spring; en les seves versions més recents. Els tres segueixen una estructura MVC que és la més emprada en aplicacions distribuïdes ja que segueix els patrons que dicta Java EE. S'estudiarà el motiu pel que utilitzen aquesta arquitectura i les avantatges que comporta.

Una vegada finalitzat l'estudi dels frameworks, s'analitzaran els requisits que haurà de complir el nou framework, bastant-se en les característiques que tenen en comú, a la vegada que s'imiten els punts forts i s'eviten els punts febles.

[1] Sun Microsystems forma part d'Oracle Corporation des del 2009.

Posteriorment, es farà el disseny en base als requisits marcats. Es seguiran estàndards de disseny i es farà ús dels patrons Java EE per tal de desenvolupar un framework de qualitat.

Finalment, es desenvoluparà el nou framework, que s'enfocarà i adaptarà a un ús per aplicacions no molt complexes. D'aquesta manera s'aconseguirà un framework lleuger i optimitzat, per contra d'altres que tenen masses funcions i opcions de configuració que poden arribar a afectar el rendiment.

El framework a desenvolupar tindrà molt en compte la complexitat del codi generat, és a dir, que l'avantatge de treballar amb un codi de més alt nivell no suposi un deteriorament del rendiment degut a la introducció de moltes operacions o codi innecessari.

El desenvolupament del nou framework es farà de manera incremental, és a dir, primerament es desenvoluparà el nucli del framework i posteriorment s'aniran afegint funcionalitats i mòduls. Dins cada increment es seguiran les següents fases: desenvolupament, proves unitàries, integració i proves d'integració.

A més, de cara a facilitar el seu ús, es desenvoluparà un complement o *plugin* per l'IDE Eclipse que agilitzarà el procés de configuració del framework i la creació de nous components. Per certes configuracions dels components es farà ús d'anotacions dins les classes Java, cosa que reduirà la quantitat d'arxius de configuració necessaris i millorarà la llegibilitat del codi font.

En finalitzar el desenvolupament del framework i l'eina IDE, es generarà una bona documentació que serveixi als principiants a començar a desenvolupar aviat i bé. Per descriure la funcionalitat de les classes i els seus mètodes es farà ús de Javadoc, l'estàndard de Java per la documentació de classes. També es generarà una altra documentació més directe per el framework i l'eina IDE.

Per acabar, i amb l'objectiu de demostrar la utilitat i el bon funcionament del nou framework, es farà una aplicació que emprarà totes les seves característiques.

2. Objectius generals i específics

L'objectiu global del projecte és posar-se en la situació d'un arquitecte de software que proporciona eines d'ajuda al desenvolupador de programari, en comptes de la d'aquest darrer, que programa aplicacions per un client fent ús d'aquestes eines. Així, es té l'oportunitat de fer una eina que facilitarà el desenvolupament de futures aplicacions amb tot allò que necessitarem i farem ús.

Els objectius específics es poden entendre des de dos punts de vista:

Estudi dels frameworks actuals

Amb el primer punt, es pretén obtenir uns coneixements amplis dels frameworks més utilitzats per desenvolupar la capa de presentació d'aplicacions web, així com millorar coneixements de Java EE i del patró MVC que utilitzen. Investigant i analitzant el seu funcionament es profunditzarà sobre la importància de l'ús de patrons per solucionar diverses problemàtiques.

Desenvolupament d'un nou framework

Del producte final s'espera que sigui un framework no molt complex, sinó fàcil d'aprendre, utilitzar i desenvolupar-hi. A la vegada, ha de ser escalable, és a dir que es pugui millorar, ampliar i reutilitzar en un futur. En aquest aspecte serà imprescindible el manteniment d'un codi llegible i ben documentat, fent ús de javadoc, i proporcionar una documentació completa i ben estructurada.

A més, el nou framework no ha de ser gaire pesat, és a dir, l'usuari no ha de veure penalitzat la reducció d'esforç a l'hora de programar amb un augment del temps d'execució.

En quan a l'eina d'ajuda que s'integrarà amb Eclipse IDE, s'espera que agilitzi la configuració i la generació automàtica de codi del framework, facilitant l'aprenentatge i l'ús d'aquest.

Finalment, la creació d'una petita aplicació web que utilitzi el framework desenvolupat pretendrà demostrar la utilitat i la facilitat d'ús d'aquest, així com de l'eina per l'IDE creada. La mesura de la quantitat de codi auto generat a les pàgines i del temps de carrega, permetran avaluar l'eficàcia del framework.

3. Planificació

Per dur a terme la planificació es descompondrà el projecte en tasques que aniran molt relacionades amb les fites del projecte. L'ordre de les tasques és important i es pot veure definit al diagrama de Gantt.

3.1. Diccionari EDT

Codi: 2	**Seguiment del projecte**	**De 23/09/2011**	**Fins 16/01/2012**
Descripció	Seguiment i control dia a dia de les tasques per detectar possibles desviaments i poder-los tractar.		
Durada	115 dies		
Resultats	- MS Project actualitzat		

Codi: 4	**Pla de treball**	**De 23/09/2011**	**Fins 03/10/2011**
Descripció	Elaboració del pla de treball.		
Durada	11 dies		
Resultats	- Pla de treball		

Codi: 4.1	**Lectura i comprensió de l'enunciat**	**De 23/09/2011**	**Fins 25/09/2011**
Descripció	Lectura de l'enunciat del projecte final i comprensió dels conceptes exposats a l'enunciat.		
Durada	11 hores		
Resultats	-		

Codi: 4.2	**Planificació temporal**	**De 26/09/2011**	**Fins 29/09/2011**
Descripció	Identificació de les tasques a realitzar, organització i priorització de les tasques i generació del MS Project		
Durada	10 hores		
Resultats	- EDT - Calendari de fites - Document MS Project		

Codi: 4.3	**Document del Pla de Treball**	**De 29/09/2011**	**Fins 01/10/2011**
Descripció	Elaboració de les seccions Descripció, Objectius i Planificació del Pla de Treball		
Durada	8 hores		
Resultats	- Document del Pla de Treball		

Codi: 4.4	**Repàs i maquetació final**	**De 02/10/2011**	**Fins 03/10/2011**
Descripció	Repàs del Pla de Treball i maquetació final. Preparació de l'enviament de l'entrega.		
Durada	6,5 hores		
Resultats	- Document del Pla de Treball repassat - Planificació MS Project repassada - Enviament preparat		

Codi: 6	Anàlisi de frameworks	De 03/10/2011	Fins 05/11/2011
Descripció	Estudi i anàlisis dels frameworks més utilitzats al mercat.		
Durada	31 dies		
Resultats	- Document d'anàlisis dels frameworks		

Codi: 6.1	Estudi de patrons	De 03/10/2011	Fins 11/10/2011
Descripció	Estudi dels patrons utilitzats per els frameworks a analitzar. Estudi dels patrons J2EE i MVC.		
Durada	8 dies		
Resultats	- Document d'anàlisis dels patrons		

Codi: 6.2	Estudi dels frameworks	De 11/10/2011	Fins 03/11/2011
Descripció	Anàlisi i comparatives dels frameworks més utilitzats al món empresarial: JSF, Struts i Spring.		
Durada	21,3 dies		
Resultats	- Document d'anàlisis dels frameworks - Documents de proves dels frameworks		

Codi: 6.3	Documentació	De 03/10/2011	Fins 05/11/2011
Descripció	Finalitzar la documentació de les tasques 6.1 i 6.2.		
Durada	6 hores		
Resultats	- Document d'anàlisis dels frameworks		

Codi: 8	Desenvolupament del framework	De 05/11/2011	Fins 01/01/2012
Descripció	Implementació del framework.		
Durada	55 dies		
Resultats	- Llibreries (JAR) del framework - Documentació i manuals d'usuari		

Codi: 8.1	Anàlisi del framework	De 05/10/2011	Fins 11/11/2011
Descripció	Abast de la funcionalitat i requeriments del framework a implementar.		
Durada	6 dies		
Resultats	- Document d'abast - Document de requeriments		

Codi: 8.2	Disseny del framework	De 11/11/2011	Fins 27/11/2011
Descripció	Diagrames per al disseny del framework		
Durada	16 dies		
Resultats	- Diagrama de classes UML - Diagrama de seqüència		

Codi: 8.3	Implementació del framework	De 27/11/2011	Fins 01/01/2012
Descripció	Implementació del framework, proves i generació de les llibreries.		
Durada	33,6 dies		
Resultats	- Codi font del framework - Documents de proves - Llibreries JAR		

Codi: 10	Desenvolupament eina IDE	De 26/12/2011	Fins 31/12/2011
Descripció	Anàlisis, disseny, implementació i proves d'una eina d'ajuda per Eclipse IDE.		
Durada	4 dies		
Resultats	- Eina per Eclipse IDE - Documentació i manuals d'usuari		

Codi: 11	Desenvolupament d'aplicació d'exemple	De 31/12/2011	Fins 09/01/2012
Descripció	Anàlisi, Disseny, implementació i proves d'una aplicació que utilitzi el framework desenvolupat i en demostri les característiques.		
Durada	9,67 dies		
Resultats	- Codi font de l'aplicació - Aplicació web (fitxer WAR) - Documentació de l'aplicació		

Codi: 12	Seguiment i actualització de la memòria	De 06/10/2011	Fins 09/01/2012
Descripció	Elaboració de la memòria del projecte.		
Durada	94 dies		
Resultats	- Memòria del projecte		

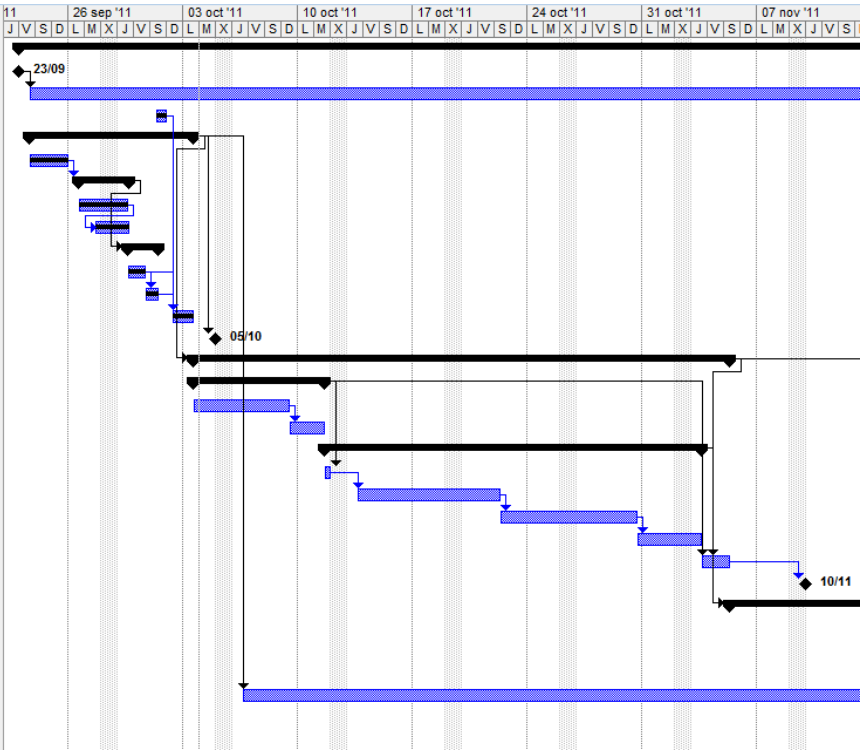
Codi: 13	Elaboració de la presentació	De 09/10/2011	Fins 14/01/2012
Descripció	Elaboració de la memòria del projecte.		
Durada	12 hores		
Resultats	- Document de presentació		

3.2. Fites del projecte

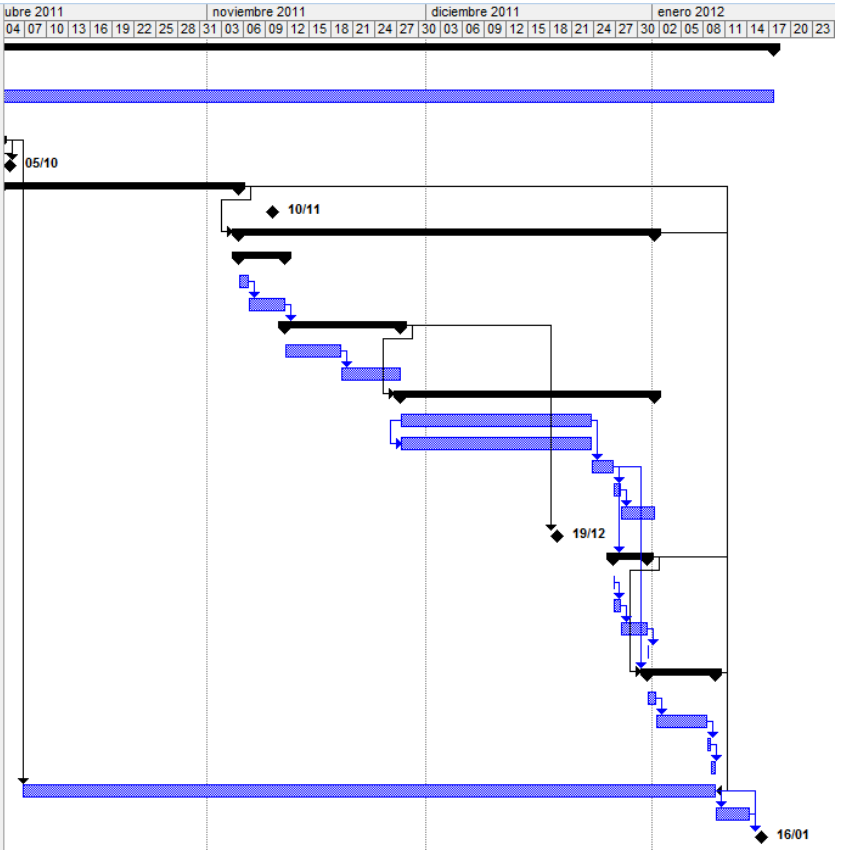
Data	Codi - Descripció
23/09/2011	1 – Inici de projecte
05/10/2011	5 – Entrega PAC1: Pla de treball
10/11/2011	7 – Entrega PAC2: Anàlisis de frameworks
19/12/2011	9 – Entrega PAC3: Disseny del framework
16/01/2011	14 – Entrega final del projecte

3.4. Diagrama de Gantt

EDT	Nom de la tasca	Durada	Inici	Fi
0	**Projecte Final de Carrera**	**114,17 días**	**vie 23/09/11**	**mar 17/01/12**
1	Inici de projecte	0 días	vie 23/09/11	vie 23/09/11
2	Seguiment del projecte	114,17 días	vie 23/09/11	mar 17/01/12
3	Tutoria amb el tutor	2,5 horas	sáb 01/10/11	sáb 01/10/11
4	**Pla de treball (PAC1)**	**10,83 días**	**vie 23/09/11**	**lun 03/10/11**
4.1	Lectura i comprensió de l'enunciat	11 horas	vie 23/09/11	dom 25/09/11
4.2	**Planificació temporal**	**2,33 días**	**lun 26/09/11**	**jue 29/09/11**
4.2.1	Definir tasques	6 horas	lun 26/09/11	jue 29/09/11
4.2.2	Prioritzar i organitzar tasques	4 horas	mar 27/09/11	jue 29/09/11
4.3	**Document del pla de treball**	**2,67 días**	**jue 29/09/11**	**sáb 01/10/11**
4.3.1	Descripció	4 horas	jue 29/09/11	vie 30/09/11
4.3.2	Objectius	4 horas	vie 30/09/11	sáb 01/10/11
4.4	Repàs i maquetació final	6,5 horas	dom 02/10/11	lun 03/10/11
5	Entrega del pla (PAC1)	0 días	mié 05/10/11	mié 05/10/11
6	**Anàlisi de frameworks (PAC2)**	**31,33 días**	**lun 03/10/11**	**sáb 05/11/11**
6.1	**Estudi de patrons**	**8 días**	**lun 03/10/11**	**mar 11/10/11**
6.1.1	Estudi de patrons Java EE	16 horas	lun 03/10/11	dom 09/10/11
6.1.2	Estudi del patró MVC	8 horas	dom 09/10/11	mar 11/10/11
6.2	**Estudi de frameworks**	**21,33 días**	**mar 11/10/11**	**jue 03/11/11**
6.2.1	Identificació dels frameworks a analitzar	2 horas	mar 11/10/11	mar 11/10/11
6.2.2	Anàlisi de característiques	27 horas	jue 13/10/11	sáb 22/10/11
6.2.3	Proves i valoracions dels frameworks	27 horas	sáb 22/10/11	dom 30/10/11
6.2.4	Comparativa	8 horas	dom 30/10/11	jue 03/11/11
6.3	Documentació	6 horas	jue 03/11/11	sáb 05/11/11
7	Entrega Anàlisi (PAC2)	0 días	jue 10/11/11	jue 10/11/11
8	**Desenvolupament de framework propi**	**55,67 días**	**sáb 05/11/11**	**dom 01/01/12**
9	Entrega Disseny (PAC3)	0 días	lun 19/12/11	lun 19/12/11
10	**Desenvolupament eina IDE**	**4 días**	**lun 26/12/11**	**sáb 31/12/11**
11	**Desenvolupament d'una aplicació**	**9,67 días**	**sáb 31/12/11**	**lun 09/01/12**
12	Seguiment i actualització de la memòria	94 días	jue 06/10/11	lun 09/01/12
13	Elaboració de la presentació	12 horas	lun 09/01/12	sáb 14/01/12
14	Entrega Final	0 días	lun 16/01/12	lun 16/01/12



EDT	Nom de la tasca	Durada	Inici	Fi
0	**Projecte Final de Carrera**	**114,17 días**	**vie 23/09/11**	**mar 17/01/12**
1	Inici de projecte	0 días	vie 23/09/11	vie 23/09/11
2	Seguiment del projecte	114,17 días	vie 23/09/11	mar 17/01/12
3	Tutoria amb el tutor	2,5 horas	sáb 01/10/11	sáb 01/10/11
4	**Pla de treball (PAC1)**	**10,83 días**	**vie 23/09/11**	**lun 03/10/11**
5	Entrega del pla (PAC1)	0 días	mié 05/10/11	mié 05/10/11
6	**Anàlisi de frameworks (PAC2)**	**31,33 días**	**lun 03/10/11**	**sáb 05/11/11**
7	Entrega Anàlisi (PAC2)	0 días	jue 10/11/11	jue 10/11/11
8	**Desenvolupament de framework propi**	**55,67 días**	**sáb 05/11/11**	**dom 01/01/12**
8.1	**Anàlisis del framework**	**6 días**	**sáb 05/11/11**	**vie 11/11/11**
8.1.1	Abast del framework	6 horas	sáb 05/11/11	dom 06/11/11
8.1.2	Definició de requeriments	12 horas	dom 06/11/11	vie 11/11/11
8.2	**Disseny del framework**	**16 días**	**vie 11/11/11**	**dom 27/11/11**
8.2.1	Diagrama de classes (UML)	24 horas	vie 11/11/11	sáb 19/11/11
8.2.2	Diagrames de seqüència	24 horas	sáb 19/11/11	dom 27/11/11
8.3	**Implementació del framework**	**33,67 días**	**dom 27/11/11**	**dom 01/01/12**
8.3.1	Implementació del framework	75 horas	dom 27/11/11	vie 23/12/11
8.3.2	Proves unitaries	75 horas	dom 27/11/11	vie 23/12/11
8.3.3.	Proves finals	12 horas	vie 23/12/11	lun 26/12/11
8.3.4	Generació dels JAR entregables	2 horas	lun 26/12/11	mar 27/12/11
8.3.5	Documentació del framework	12 horas	mar 27/12/11	dom 01/01/12
9	Entrega Disseny (PAC3)	0 días	lun 19/12/11	lun 19/12/11
10	**Desenvolupament eina IDE**	**4 días**	**lun 26/12/11**	**sáb 31/12/11**
11.1	Anàlisis eina IDE	1 hora	lun 26/12/11	lun 26/12/11
11.2	Disseny eina IDE	2 horas	lun 26/12/11	mar 27/12/11
11.3	Desenvolupament eina IDE	8 horas	mar 27/12/11	sáb 31/12/11
11.4	Proves eina IDE	1 hora	sáb 31/12/11	sáb 31/12/11
11	**Desenvolupament d'una aplicació**	**9,67 días**	**sáb 31/12/11**	**lun 09/01/12**
12.1	Disseny de l'aplicació	3 horas	sáb 31/12/11	dom 01/01/12
12.2	Desenvolupament de l'aplicació	21 horas	dom 01/01/12	dom 08/01/12
12.3	Proves	3 horas	dom 08/01/12	dom 08/01/12
12.4	Generació del WAR entregable	2 horas	dom 08/01/12	lun 09/01/12
12	Seguiment i actualització de la memòria	94 días	jue 06/10/11	lun 09/01/12
13	Elaboració de la presentació	12 horas	lun 09/01/12	sáb 14/01/12
14	Entrega Final	0 días	lun 16/01/12	lun 16/01/12



Anàlisi comparatiu dels frameworks actuals

En aquest punt comença l'anàlisi dels frameworks més utilitzats al món empresarial. El primer a analitzar són els patrons J2EE, molt utilitzats a les aplicacions web i als frameworks analitzats. Després es seguirà amb l'anàlisi de Struts 2, Spring MVC i JavaServer Faces.

4. Patrons J2EE

Un patró és un conjunt de disseny i bones pràctiques per solucionar un problema que es repeteix en un determinat context. Els patrons serveixen per no tornar a pensar la solució a un problema que ja ha estat resolt moltes vegades de la mateixa manera. Per a que una solució es consideri un patró ha d'haver-se provat la seva eficàcia en diversos projectes.

A J2EE es descriuen una sèrie de patrons dividits en tres àrees segons la seva aplicació: presentació, negoci i integració. El següent apartat es centrarà en els patrons J2EE a la capa de presentació (Figura 1).

4.1. Patrons de la capa de presentació

Els patrons de la capa de presentació són aquells que es centren en millorar i facilitar com s'implementa la lògica de presentació. Cada patró es troba dividit en dos apartats: el problema que resol i la solució al problema. A més, alguns patrons presenten un tercer apartat, on s'expliquen les estratègies que es poden seguir per implementar-lo.

4.1.1. Filtre d'intercepció *(Intercepting Filter)*

Problema: Necessitem interceptar i manipular la petició i la resposta, abans i després de ser processades, per poder determinar com continua la petició i realitzar certes accions.

Solució: Tenir un gestor de filtres o *Filter Manager* que s'encarregui de crear els filtres i delegar el control de la petició o resposta al filtre adequat. Solen estar presents en processos d'autenticació, registre d'accions o *log*, encriptació, etc.

Els filtres s'han de poder afegir o llevar sense haver de canviar el codi font i ser independents entre ells, de manera similar a com ho fa l'especificació de Servlet 2.3 o superior.

4.1.2. Controlador central *(Front Controller)*

Problema: Necessitem un punt d'accés centralitzat on capturar les peticions per tal de no duplicar codi comú a cada acció.

Solució: Utilitzar un controlador central (o més d'un, agrupant accions amb característiques comuns) que capturi totes les peticions i les redirigeixi al gestor d'accions adequat.

A més de gestionar les accions, el controlador central és responsable de gestionar les vistes, és a dir, trobar i presentar la vista adequada a la petició.

Estratègies d'implementació

- Controlador Servlet. El controlador central és un Servlet.

- Controlador JSP. El controlador central és un JSP. Es una estratègia desaconsellada ja que mescla les capes vista i controlador del patró MVC
- Ordre i controlador. El controlador, normalment un Servlet, utilitza un ajudant per trobar l'acció o ordre a executar.

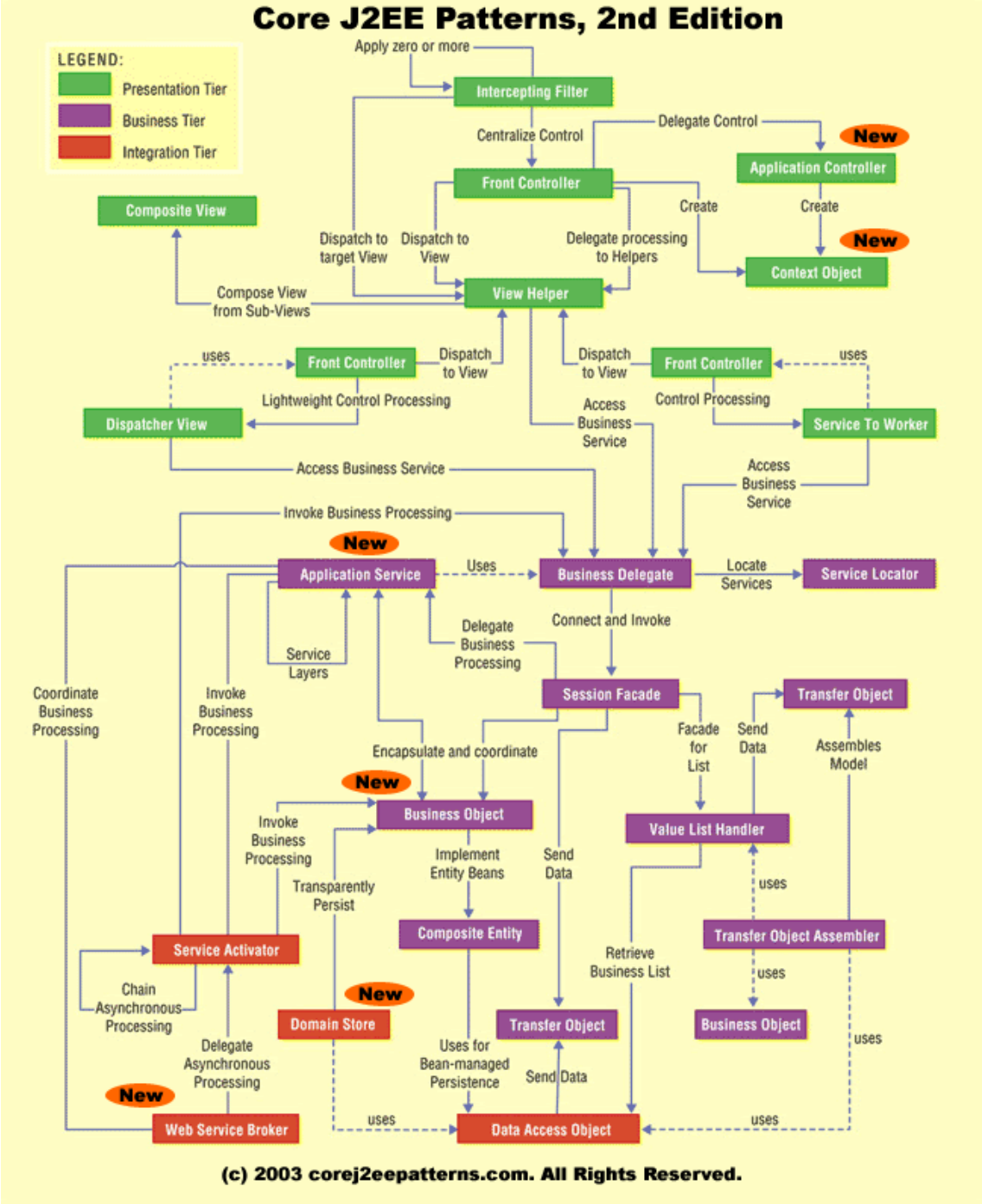


Figura 1. Relació dels patrons J2EE

4.1.3. Objecte de context *(Context Object)*

Problema: Evitar l'ús d'un sistema de protocol específic, és a dir, no utilitzar un protocol com HTTP més enllà de la primera capa.

Solució: Encapsular la informació de la petició en un objecte de context independent del protocol, que s'utilitzarà al llarg del cicle de petició. Això facilita les proves de l'aplicació i redueix el tràfic per la xarxa, ja que s'utilitza un objecte controlat en comptes de tota la petició.

Estratègies d'implementació

- Utilitzar un *Map*. És la implementació més bàsica al ser pobrament tipada, és a dir, no s'utilitzen objectes concrets.
- Utilitzar un POJO[2] per encapsular la petició. És una implementació tipada i amb mètodes d'ajuda, però el desenvolupament és més feixuc.
- Utilitzar una mescla de POJO i *Map*. Utilitza un objecte POJO amb mètodes d'ajuda per les propietats comuns a totes les peticions i un *Map* per les propietats que canvien a cada petició. És el que fa servir Struts 2 quan implementa els *ActionForm*.
- Utilitzar una objecte auto convertible. Es fa ús de *reflection*[3] i llibreries d'utilitats com BeanUtils[4] per convertir automàticament una petició a un JavaBean[5], relacionant els paràmetres de la petició a propietats del JavaBean.

4.1.4. Controlador d'aplicació *(Application Controller)*

Problema: Es vol centralitzar i modularitzar la gestió d'accions i la gestió de vistes, evitant que el controlador central creixi massa al haver de gestionar, per si sol, aquestes funcionalitats.

Solució: Separar la gestió d'accions i la gestió de vistes del controlador central mitjançant un gestor o ajudant per cada funcionalitat.

Estratègies d'implementació

- Gestor d'accions. Obté l'acció que correspon a la petició a partir d'un mapeig d'accions per a que el controlador frontal l'executi. Normalment, les accions implementen una interfície amb un mètode d'execució `execute()`.
- Gestor de vistes. Obté la vista on delegar la visualització a partir de la petició i un mapeig de vistes, que conté la relació entre el nom de la vista lògic i el recurs físic.
- Gestor de transformacions. Obté la vista i utilitza una eina de transformació per generar la visualització.
- Navegació i control de flux. S'utilitza per pantalles que han de seguir un determinat ordre d'execució o presentació. Per exemple, comprovar una condició prèvia, com que l'usuari estigui autenticat, o comprovar que s'hagi introduït unes dades per poder accedir a una vista.

[2] Plain Old Java Object. Es refereix a una classe simple que no depèn de cap llibreria externa i proporciona mètodes per accedir a les seves propietats.
[3] reflection. És el procés de consultar i modificar l'estructura del programa en temps d'execució. Per exemple, accedint a mètodes dinàmicament o recórrer les propietats d'una classe sense saber de quina classe es tracta.
[4] BeanUtils. Llibreria d'Apache amb utilitats per gestionar Beans.
[5] JavaBean. Classe Java que segueix certes convencions: http://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html

4.1.5. Ajudant de vista *(View Helper)*

Problema: Es vol separar completament la vista de la lògica de presentació per facilitar la reutilització, el manteniment i la modularitat.

Solució: Utilitzar Vistes o *Views,* per encapsular la lògica de visualització, i Ajudants o *Helpers* com POJO, etiquetes personalitzades o llibreries d'etiquetes; per encapsular la lògica de presentació.

Estratègies d'implementació

- Basada en plantilles. Es separa el rol del dissenyador web i del programador. S'utilitzen llibreries com JSTL.
- Vista basada en controlador. Utilitza un Servlet per presentar la vista. És la pitjor estratègia, ja que mescla llenguatge d'etiquetes dins el codi Java.
- Ajudant JavaBean. S'utilitza un JavaBean dins la pàgina JSP, pel que no acaba de separar del tot la vista de la lògica.
- Etiquetes personalitzades. Ús d'etiquetes personalitzades que, encara que són molt flexibles i reutilitzables, afegeixen bastanta complexitat.
- Arxiu d'etiquetes. Soluciona el problema de substituir codi Java per etiquetes amb la mateixa funcionalitat, que no és el que pretén el patró, utilitzant arxius d'etiquetes on s'implementa la visualització i es pot reutilitzar a diverses pàgines JSP.

4.1.6. Vista composta *(Composite View)*

Problema: Evitar la duplicació de codi construint una vista modular amb components atòmics, com capçaleres, peus, taules, etc.

Solució: Utilitzar un gestor de vista que utilitzi una plantilla per saber com col·locar el contingut.

Estratègies d'implementació

- Gestor JavaBean. El JavaBean implementa la lògica de control de les vistes i la composició. No és massa elegant ja que utilitza codi scriptlet[6].
- Gestor estàndard amb etiquetes. A cada pàgina JSP s'inclou l'estructura base mitjançant etiquetes JSP. Presenta el problema que, en cas d'haver un canvi de composició, s'hauran de modificar totes les pàgines una per una.
- Gestor personalitzat d'etiquetes. És l'estratègia més flexible però més complexa, ja que utilitza etiquetes personalitzades. La idea és tenir una plantilla base i a cada pàgina definir que s'utilitzarà aquesta plantilla i assignar un contingut a cada zona. És el que utilitza la llibreria UIComposition del framework JSF.
- Transformador XSLT. Utilitza una plantilla XSL per transformar un model XML.

4.1.7. Servei al treballador *(Service to Worker)* i Distribuïdor de Vista *(Dispatcher View)*

Aquests dos patrons descriuen una combinació del patró controlador central i del patró ajudant de vista.

[6] Scriptlet. Fragment de codi Java dins una pàgina JSP.

Problema: Centralitzar el control d'accés i recuperació de la vista, per evitar codi duplicat a les vistes i no mesclar la lògica de negoci amb la presentació.

Solució: Combinar un controlador i un servei per despatxar vistes per preparar la presentació.

Exemple de servei al treballador:
`http://some.server.com/servlet/Controller?action=login`

Exemple de distribuïdor de vista:
`http://some.server.com/servlet/Controller?next=login.jsp`

La diferència entre els dos patrons és que el primer té un control més alt de la vista, ja que es basa en l'acció i el seu resultat per determinar el model de presentació. Aquesta correspondència es pot fer mitjançant un arxiu de configuració XML, com fa Struts. En canvi, el segon patró, determina la vista directament a partir de la petició, és a dir, només ha de redirigir a la vista determinada pel paràmetre *next*.

4.2. Males pràctiques

A més dels patrons del punt anterior, es recomana seguir un conjunt de bones pràctiques o, el que és el mateix, evitar una sèrie de males pràctiques que es solen produir a l'hora de desenvolupar aplicacions web.

Ús de *scriptlets* a les pàgines JSP

No és recomanable l'ús de codi Java dins les pàgines JSP, ja que es mescla la lògica amb la presentació. En comptes de *scriptlets* es recomana l'ús del patró *View Helper* per substituir aquest codi per etiquetes, sempre i quan no es faci com s'explica al següent apartat.

Substitució literal de scriptlets per etiquetes

L'ús del patró View Helper no es basa en substituir un scriptlet per una etiqueta que faci la mateixa funcionalitat. Per exemple, no s'ha de substituir el Codi d'exemple 1 per el Codi d'exemple 2:

```
<% for (Objecte o : llistatObjectes ) {
...
} %>
```

Codi d'exemple 1. Scriptlet

```
<c:forEach var="o" items="llistatObjectes">
...
</c:forEach>
```

Codi d'exemple 2. Etiqueta JSTL

D'aquesta manera, només canviem el llenguatge i no ens serveix per reutilitzar codi. L'ús correcte seria desenvolupar una etiqueta que permetés recorrer el llistat i presentar els resultats:

```
<util:taulaObjectes items="llistatObjectes" class="llistat" />
```

No validar el reenviament de formularis

És un fet freqüent que un client reenviï un formulari al refrescar una pàgina, fet que pot ocasionar dades duplicades o errors. Això s'ha de controlar, per exemple, amb la validació d'una mostra o *token* sincronitzada.

Aquesta validació consisteix en generar una mostra durant el procés de petició per després validar-la en la següent petició. D'aquesta manera, si rebem una mostra que no coincideix amb la mostra guardada en sessió al servidor vol dir que s'ha reenviat la petició i, per tant, hem d'invalidar la segona petició. Aquest mètode és el que utilitza Struts 2.

5. Struts 2 Framework

Struts 2 és un framework de codi lliure creat per Apache Foundation per facilitar el desenvolupament d'aplicacions web que segueixin l'arquitectura MVC. La seva darrera versió, la 2.2.3.1, va ser llançada per setembre de 2011 i és la que s'analitza en aquest projecte.

5.1. Arquitectura

Aquest framework, que entra dins la categoria dels frameworks orientats a accions, es basa en el patró MVC per separar la vista, el controlador i el model. La vista queda implementada per el resultat o *Result*, el controlador per la classe *FilterDispatcher* i per les classes que implementen la interfície *Action*. La capa del model es deixa lliure d'implementació i es pot integrar amb tecnologies com EJB, DAOs o JavaBeans amb accés JDBC o Hibernate, etc.

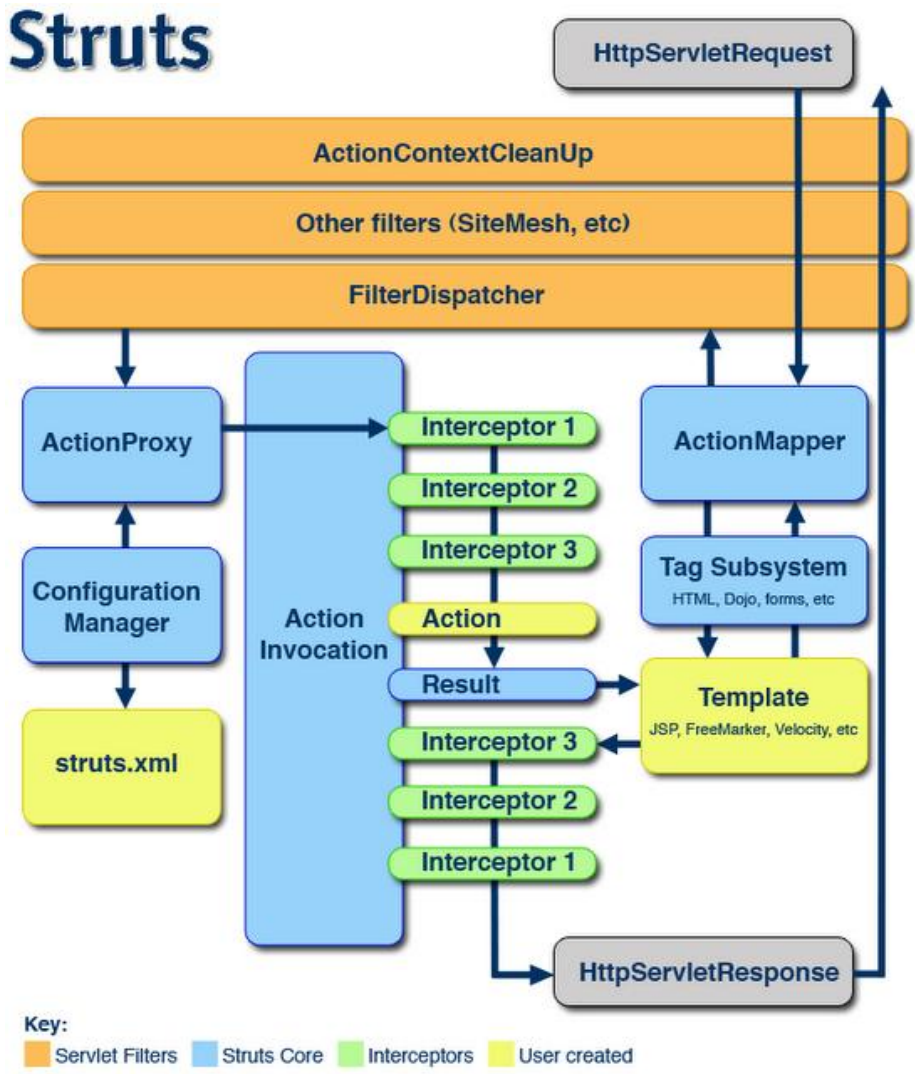


Figura 2. Arquitectura Struts 2

A la Figura 2 es mostra la arquitectura de Struts 2, on es pot veure tots els seus components i quin és el cicle que segueix una petició. A continuació es descriuen els principals components i com intervenen en el cicle de vida d'una petició:

1. ***FilterDispatcher***

És la classe que fa de controlador central i el primer component que interactua amb la petició.

2. ***ActionMapper***

És la interfície utilitzada pel *FilterDispatcher* per trobar l'acció a executar. A partir de la petició *HttpServletRequest* retorna un objecte *ActionMapping* que conté l'acció corresponent a la URL de la petició.

Juntament amb *FitcherDispatcher*, implementen el patró de controlador d'aplicació.

3. ***ActionProxy***

És la interfície en la que delega el control el *FilterDispatcher* una vegada trobada l'acció a executar. S'encarrega de consultar la configuració, amb l'ajuda de *ConfigurationManager*, i crear l'objecte *ActionInvocation* que executarà l'acció.

4. ***ConfigurationManager***

És l'objecte que representa la configuració del framework. Per defecte, es carrega la configuració de l'arxiu *struts.xml* a través de la classe *StrutsXmlConfigurationProvider* que implementa la interfície *ConfigurationProvider*.

5. ***ActionInvocation***

És la classe que implementa el patró *Command* de J2EE i, per tant, s'encarrega d'executar l'acció *Action*. Una vegada executada, cerca el resultat associat a l'acció i especificat a la configuració, i l'executa. Aquest resultat pot ser una plantilla JSP o FreeMarker a renderitzar.

6. ***Interceptors***

Sempre s'executen abans i després de l'acció, encara que no hagin de fer res. Són reutilitzables, gràcies a que estan separats del codi de l'acció, i solen tenir finalitats de validació, registre d'accions, depuració, etc.

7. ***Action***

És tota classe que implementa la interfície *Action* i realitza una tasca o funció dins l'aplicació. S'encarrega d'invocar els mètodes de la lògica de negoci. Struts 2 proporciona algunes implementacions com *ActionSupport*, que proporciona mètodes per les accions més comuns: afegir missatges al context, localitzar els missatges, validació i execució de l'acció.

8. ***Result***

És una interfície que representa el resultat d'una acció, ja sigui mostrar una pàgina web, enviar un correu electrònic, etc. Struts 2 proporciona implementacions per alguns tipus de resultats com XSLT, Velocity, arxius, portlets[7], etc. que contenen mètodes adequats al tipus de resposta.

9. ***Template***

Són les pàgines que implementen la vista i renderitzen la resposta a l'usuari. Poden ser JSP i utilitzar etiquetes pròpies de Struts 2 i llenguatges d'expressions.

10. **ActionContextCleanUp**

És un filtre opcional (discontinuat a la nova versió) que s'encarrega de netejar el context, en comptes de ser el *FilterDispatcher* qui ho faci, i facilitar la integració amb SiteMash.

[7] Un portlet és un component modular que s'afegeix a una pàgina d'un portal, com si es tractés d'una finestra d'un escriptori, i no és accessible directament. Dins una pàgina pot haver més d'un portlet, que poden tenir finalitats distintes i no estar relacionats entre ells.

11. ObjectFactory

És una classe que s'utilitza al llarg de tot el framework de Struts 2 per instanciar els objectes del nucli: interceptors, accions, resultats, etc.

5.2. Cicle de vida d'una petició

Suposant una acció per mostrar un llistat de productes <host>/llistatproductes.action, on el mapeig a Struts 2 és mitjançant el patró *.action, obtenim les següents fases:

1. El navegador envia la petició llistatproductes.action al servidor.
2. El *FilterDispatcher* rep la petició i determina l'acció "llistatproductes" a través de l'*ActionMapper*.
3. S'apliquen els interceptors per funcionalitats com validacions, pujada d'arxius i control de navegació.
4. El mètode de l'acció trobada al punt 2 és executat i s'obté el llistat de productes del model.
5. El resultat, *Result*, és renderitzat al navegador a través del formulari del llistat de productes.

5.3. Característiques del framework

A continuació es procedirà a analitzar una sèrie de característiques i funcionalitats del framework que permetrà comparar-lo amb Spring i JSF.

5.3.1. Internacionalització i18n

Struts 2 dóna suport a la internacionalització (i18n) mitjançant l'ús d'arxius de propietats i d'etiquetes UI com l'etiqueta `<s:i18n>` i l'etiqueta `<s:text>`, que proporcionen accés a dades i maquetació d'aquestes des de la vista.

Per definir els missatges s'han de crear arxius de propietats, que segueixen unes certes regles de nomenclatura per tal que Struts 2 els trobi segons el següent ordre de cerca:

1. <nom de la classe *Action*>.properties dins la mateixa carpeta que la classe *Action* on s'utilitzen els missatges
2. <nom de la interfície>.properties dins la mateixa carpeta que la interfície
3. <classe base>.properties
4. <paquet>.properties dins la carpeta del paquet
5. Arxius de propietats globals

En el cas d'utilitzar una propietat que no sigui global (punts 1-4), la classe *Action* ha d'estendre la classe *ActionSuport*, que proporciona el mètode *getText* i habilita la cerca de l'arxiu de propietats apropiat.

En cas dels arxius de propietats globals, s'ha de definir la constant `struts.custom.i18n.resources` a l'arxiu de configuració *struts.xml*, indicant la ruta cap a l'arxiu de propietats a l'atribut *value*.

Per canviar d'idioma l'aplicació només cal declarar un paràmetre *request_locale* amb el codi del llenguatge a canviar i cridar a una classe *Action*. L'interceptor *I18nInterceptor* del

framework s'encarregarà de canviar l'idioma. Es pot veure un exemple de la funcionalitat en el següent codi:

```
<s:url id="localeEN" namespace="/" action="locale" >
   <s:param name="request_locale" >en</s:param>
</s:url>
<s:a href="%{localeEN}" >English</s:a>
```

Codi d'exemple 3. Selecció d'idioma a Struts 2

5.3.2. Proves

Es poden realitzar proves unitàries a les accions mitjançant JUnit[8]. Per això, Struts 2 utilitza el nucli de Spring i el paquet *test* d'aquest, que conté els objectes Mock[9] utilitzats per simular la petició i el pas de paràmetres que faria l'usuari.

A més, Struts 2 proporciona una extensió del framework per integrar-se amb JUnit que conté la classe *StrutsTestCase*, de la que hem d'estendre els tests de l'aplicació. Aquesta classe proporciona el mètode *getActionProxy*, que s'utilitza per executar l'acció a provar i fa ús dels objectes *Mock* de Spring per simular la petició, resposta i context.

Tot i així, es pot testejar l'aplicació sense fer ús de JUnit, ja que les accions són classes POJO simples que es poden instanciar, establir les propietats i executar obtenint el resultat a testejar.

5.3.3. Ajax

Struts 2 proporciona una llibreria d'extensió que conté classes i etiquetes per utilitzar DOJO[10]. Aquesta llibreria proporciona diverses utilitats Ajax, com validació i actualització de contingut, mitjançant l'ús d'etiquetes que generen les peticions i les respostes en JSON.

Algunes de les etiquetes que proporciona la llibreria són `<s:autocompleter>`, per generar un camp autocompletat; `<s:tree>`, per generar un arbre de resultats; `<s:div>`, per actualitzar un contingut sense haver de refrescar la pàgina.

5.3.4. Validació de formularis

Struts 2 facilita la validació de formularis, que es pot especificar utilitzant arxius XML de validació *<ActionClassName>-validation.xml* o mitjançant anotacions a les propietats de les accions. Tant al validar mitjançant XML com mitjançant anotacions, s'ha d'indicar quines regles de validació han de complir les propietats o el mètode d'execució de l'acció.

A més, el procés de validació es pot fer mitjançant Ajax, i els missatges poden ser localitzats i incorporar paràmetres, el que aporta bastanta flexibilitat a l'hora de validar formularis i mostrar els missatges.

[8] JUnit és un framework que facilita la realització de proves unitàries o en grup d'aplicacions Java.
[9] Un objecte *Mock* és aquell que simula el comportament d'un objecte real. Per exemple, Spring utilitza la classe *MockHttpServletRequest* per simular el comportament d'un *HttpServletRequest*.
[10] DOJO és una biblioteca Javascript que facilita l'ús d'AJAX dins aplicacions web.

5.3.5. Configuració del framework

La configuració principal de l'aplicació es fa mitjançant un arxiu XML anomenat *struts.xml* situat a la carpeta WEB-INF de l'aplicació web. En aquest arxiu es configuren els següents paràmetres:

- Els paquets que disposa l'aplicació. Cada paquet agrupa un conjunt d'accions amb característiques comunes com interceptors, tipus de resultats, etc.
- Les accions de l'aplicació, indicant-hi la classe que la implementa i els resultats que pot generar.
- Constants de l'aplicació.
- Beans que utilitza la aplicació, com factories d'objectes o configuracions.

A més, Struts 2 permet l'ús d'anotacions per configurar les accions, els interceptors, els validadors i els tipus de conversió. Per fer ús de les anotacions cal incloure al projecte l'extensió *Convention-Plugin*.

Per a configurar aspectes del framework en si, es pot utilitzar un arxiu de propietats *struts.properties*, on s'especifiquen propietats com la codificació, configuració de la cache, etc. Aquestes propietats sobreescriuen les opcions per defecte de l'arxiu *struts-default.properties* (del fitxer *struts2.jar*).

5.3.6. Sistemes de presentació

Struts 2 no només permet presentar la vista mitjançant pàgines JSP, sino que permet utilitzar motors de plantilles com FreeMarker, Velocity, XSLT o l'extensió pròpia de Struts 2: *struts-tiles*. El tipus de presentació s'ha d'especificar a la configuració del resultat de l'acció (mitjançant l'arxiu *struts.xml* o anotacions a la classe).

A més del sistema de plantilles, el framework proporciona un sistema de temes que serveixen per personalitzar la manera en que es renderitzen les etiquetes de Struts 2. Per crear un nou tema es sol estendre el tema XHTML que incorpora el framework i personalitzar les plantilles que es vulgui modificar mitjançant l'ús de FreeMarker o Velocity. L'ús d'un tema o un altre ve donat per l'atribut *theme* que tenen les etiquetes del framework, ja sigui escrit directament o amb el valor d'una variable de sessió.

5.3.7. Documentació

Apache proporciona una documentació molt àmplia a la seva pàgina web composta per:

- API del framework
- Guia per al desenvolupador
- Guia per desenvolupar d'etiquetes
- Guia per desenvolupar extensions
- Guia per l'arquitecte, on explica l'arquitectura de Struts
- Guies de migració
- Altres documentacions aportades per la comunitat
- Codis d'exemple

A més, conta amb molts de projectes lliures i molt de suport de la comunitat, pel que és fàcil trobar manuals que expliquen pas a pas com implementar certes funcionalitats.

6. Spring MVC Framework

Spring Framework és un framework de codi lliure creat per la comunitat Spring Source. La seva primera versió va sorgir a partir del codi publicat al llibre *Expert One-on-One J2EE Design and Development* de Rod Johnson al 2002. La versió actual, que serà l'analitzada al present projecte, és la 3.1 i va ser llançada a l'agost de 2011.

6.1. Inversió de control (IoC)

Spring va popularitzar la tècnica d'Inversió de Control, que canvia el model de programació tradicional per un model on s'inverteix el rol del programador. En comptes de ser aquest qui s'encarrega d'especificar i controlar el flux de l'aplicació, és una entitat externa o contenidor qui s'encarrega de gestionar les accions i cridar les respostes especificades per el programador.

És a dir, en comptes de ser l'usuari qui crida a un mètode d'una llibreria, és la llibreria qui s'encarrega d'executar el codi de l'usuari quan es produeixen certes accions. A Spring, aquesta tasca la du a terme un contenidor, que gestiona les instancies dels objectes de l'usuari.

Per exemple, tenim el següent codi on el programador decideix quan ha de demanar el nom i quan ha de demanar la pregunta:

```
puts 'What is your name?'
name = gets
process_name(name)
puts 'What is your quest?'
quest = gets
process_quest(quest)
```

Codi d'exemple 4. IOC - Model de programació tradicional

En canvi, en el següent codi es fa ús de la inversió de control i és una entitat externa, el sistema de finestres, qui s'encarrega de cridar els mètodes `process_name` i `process_quest` quan l'usuari abandona un camp, és a dir, quan es produeix l'esdeveniment *FocusOut*.

```
require 'tk'
root = TkRoot.new()
name_label = TkLabel.new() {text "What is Your Name?"}
name_label.pack
name = TkEntry.new(root).pack
name.bind("FocusOut") {process_name(name)}
quest_label = TkLabel.new() {text "What is Your Quest?"}
quest_label.pack
quest = TkEntry.new(root).pack
quest.bind("FocusOut") {process_quest(quest)}
Tk.mainloop()
```

Codi d'exemple 5. IOC - Model de programació amb IOC

Un altre exemple d'ús de IOC és el cas dels EJB, on la interfície defineix mètodes com `ejbPassivate` i `ejbActivate`, que l'usuari implementa però no té el control de quan es criden, si no que és el contenidor de EJB qui el du.

6.2. Arquitectura

Spring és un framework orientat a accions amb una arquitectura dividida en 7 capes o mòduls, el que permet agafar el que realment interessa per al projecte. A diferència d'alguns frameworks, com Struts, que només implementen la capa de presentació i deixen al programador la tasca d'integrar un framework per la capa de negoci, Spring sí proporciona mòduls per la implementació d'aquesta capa. El present projecte es centrarà en l'estudi del mòdul Web, referenciat com a Spring MVC.

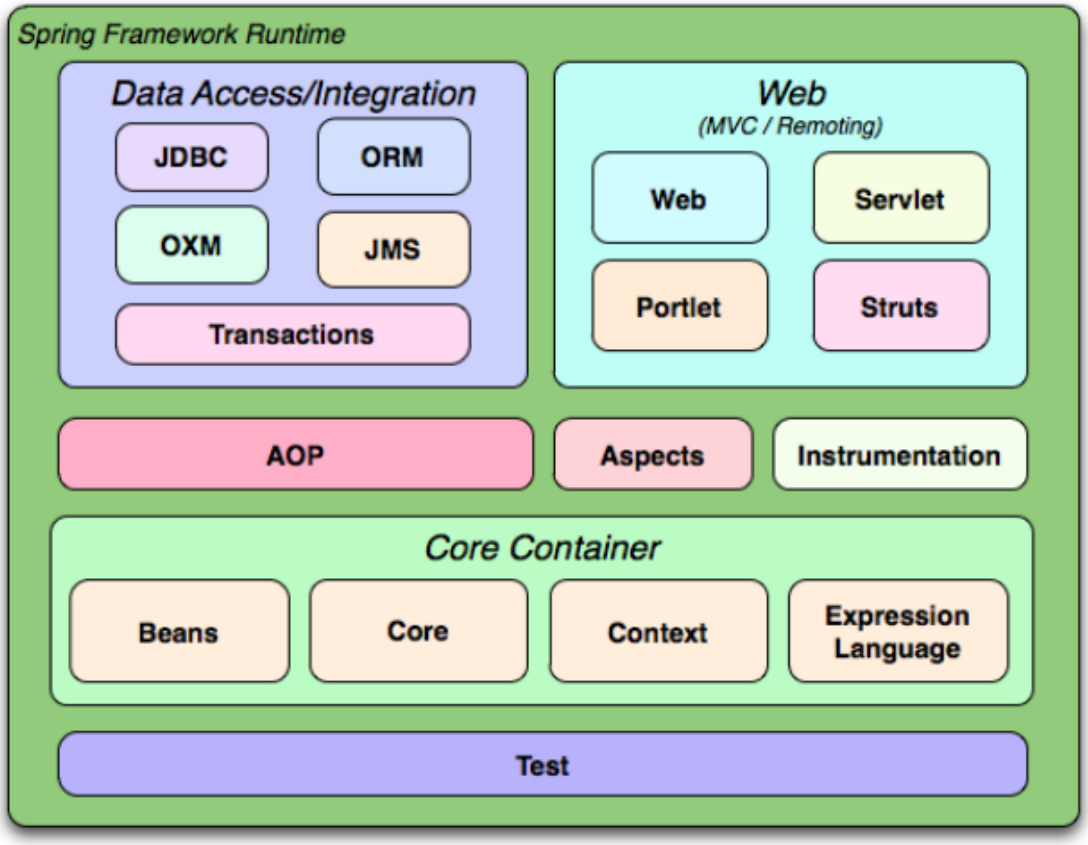


Figura 3. Mòduls de Spring

El nucli de Spring, també conegut com el contenidor IoC, és el mòdul principal que proporciona les funcionalitats bàsiques del framework i implementa la tècnica d'inversió de control mitjançant els paquets *org.springframework.beans* i *org.springframework.context*.

En concret, la interfície *BeanFactory* s'encarrega d'instanciar i configurar els objectes de l'aplicació. La implementació més utilitzada és *XmlBeanFactory*, que permet configurar tots aquests objectes i les seves relacions en XML.

Spring Web

Spring Web proporciona un MVC flexible i configurable amb les següents característiques principals:

- És orientat a accions, al igual que Struts 2
- Fa una forta divisió del model i la vista
- Proporciona interceptors
- No obliga a utilitzar JSP per a la vista, permetent tecnologies com Velocity, XML, etc.

- Permet utilitzar altres frameworks MVC com Struts o JSF per la vista
- Facilita les proves al utilitzar POJOs per implementar les funcionalitats
- Permet configurar el framework mitjançant anotacions en comptes d'arxius XML

Un component essencial a Spring MVC són els controladors, que s'encarreguen d'interpretar les peticiones rebudes i presentar el model amb la vista adequada. Això es fa amb un POJO simple, que ha d'implementar la interfície *Controller* o estendre algunes de les implementacions que proporciona Spring:

- *Controller* i *AbstractController* per tasques senzilles. Només cal implementar el mètode `handleRequestInternal(HttpRequest, HttpResponse)` i retornar un objecte *ModelAndView*.
- *SimpleFormController* per controlar formularis.
- *MultiActionController* per tenir diversos mètodes dins el mateix controlador, mapant les distintes peticions o accions a cada mètode. Els mètodes han de tenir la següent estructura:

```
public [ModelAndView | Map | void] nombreMetodo(Httpreq,
HttpResp [, Exception | AnyObjeto]);
```

- *UrlFilenameController* per accedir directament a un arxiu a partir de la URL.
- *ParametrizableViewController* per indicar la vista on es redirigirà l'acció en comptes d'escriure-la directament al codi.

L'objecte *ModelAndView*, que s'utilitza als mètodes dels controladors, conté el model i el nom lògic de la vista que s'han de presentar com a resposta a una petició. Això també es pot fer mitjançant un mapeig a un objecte *Map*.

6.3. Cicle de vida d'una petició

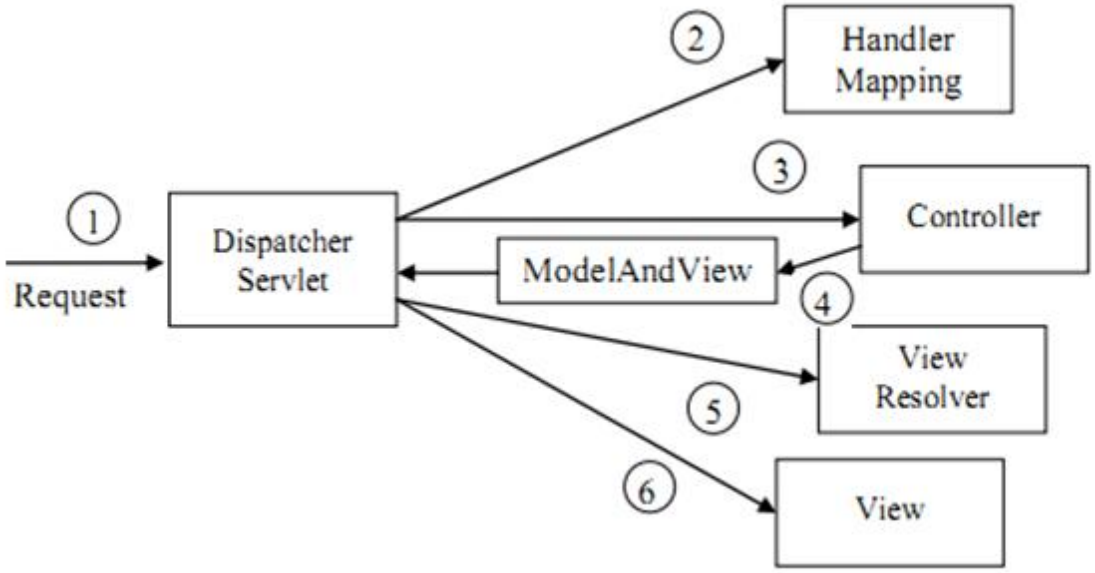


Figura 4. Cicle de vida d'una petició a Spring MVC

1. El *DispatcherServlet*, que actua de controlador central, rep una petició.
2. El *HandlerMapping* retorna un controlador a partir de la URL i els mapejos establerts.

3. El *DispatcherServlet* cedeix el control al controlador retornat, que executa la tasca.
4. El controlador retorna un objecte *ModelAndView* amb el model i la vista adequats a la petició.
5. En cas que el *ModelAndView* tingui un nom lògic per la vista, s'utilitza el *ViewResolver* per cercar l'objecte *View*. En cas contrari, el *DispatcherServlet* pot anar directament al següent pas.
6. Finalment, el *DispatcherServlet* s'encarrega de presentar la vista.

6.4. Característiques del framework

Les següents característiques analitzades marquen els punts que s'utilitzaran per comparar Spring MVC amb els altres frameworks.

6.4.1. Internacionalització i18n

Spring proporciona eines per internacionalitzar les aplicacions de manera senzilla. Per això, s'ha d'indicar on hi haurà els arxius de propietats que contenen els missatges mitjançant el següent codi a l'arxiu de configuració de Spring:

```
<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
   <property name="basename" value="messages" />
</bean>
```

Codi d'exemple 6. Configuració de la internacionalització a Spring

A més, es necessita indicar com es determinarà en quin idioma mostrar els missatges i la configuració per canviar l'idioma escollint una de les següents implementacions del *LocaleResolver*:

- *AcceptHeaderLocaleResolver*: Inspecciona la capçalera *accept-language* de la petició.
- *CookieLocaleResolver*: Cerca l'idioma a una galeta o *cookie.*
- *SessionLocaleResolver*: Cerca l'idioma a la sessió associada a l'usuari.
- *LocaleChangeInterceptor*: S'afegeix un interceptor al *HandlerMapping* per detectar un paràmetre associat a l'idioma.

6.4.2. Proves

Els controladors de Spring són POJOs simples, pel que es poden provar instanciant-los i executant els seus mètodes directament, sense necessitat d'utilitzar cap contenidor ni el framework mateix.

També es poden utilitzar objectes *Mock* per imitar la capa de persistència i provar la capa de negoci de l'aplicació amb dades creades per el test. Spring proporciona un conjunt de paquets que contenen objectes *Mock*: JNDI, Web i Portlet; a més d'un paquet d'utilitats per les proves.

Per realitzar les proves dels controladors de Spring MVC s'utilitzen les classes *ModelAndViewAssert*, *MockHttpServletRequest* i *MockHttpSession*, incloses al paquet *org.springframework.test.web*, en combinació amb JUnit, TestNG o altres frameworks de proves.

6.4.3. Ajax

Spring MVC proporciona Ajax de manera transparent a l'hora de programar el controlador, és a dir, no diferencia una petició Ajax d'una petició normal, pel que el mateix mètode pot ser utilitzat per ambdós tipus de peticions.

L'únic que cal afegir al codi és l'anotació @ResponseBody al tipus retornat per el mètode i l'anotació @RequestBody als paràmetres del mètode, per tal d'indicar a Spring que converteixi la petició i la resposta a JSON, sempre i quan es tracti d'una petició asíncrona. Al rebre una resposta en format JSON, el seu tractament mitjançant Javascript és fàcil, sobretot si s'utilitza un framework com jQuery o Dojo.

6.4.4. Validació de formularis

Spring MVC segueix l'especificació JSR-303[11] per la validació de Beans mitjançant anotacions. Per indicar al controlador que ha de validar un model, només cal indicar-ho mitjançant l'anotació @Valid a l'argument *model* del mètode. Al Bean, de la mateixa manera que fa Struts 2, s'ha de posar les regles de validació mitjançant anotacions a les propietats.

També es permet la validació mitjançant la interfície *Validator*, que proporciona el mètode `validate(Object obj, Errors e)` i és on s'ha d'implementar la lògica de validació.

Per mostrar els missatges d'error, Spring MVC proporciona l'etiqueta `<form:errors />` que pot ser utilitzada per mostrar els errors d'un camp en particular o del formulari sencer.

6.4.5. Configuració del framework

Spring MVC, al igual que Struts 2, s'integra incorporant un Servlet, en concret el *DispatcherServlet*, a l'arxiu web.xml de l'aplicació i mapant-lo a un patró URL.

La configuració del framework dins l'aplicació es fa mitjançant un arxiu XML que ha de seguir la nomenclatura *<servlet-name>-servlet.xml*. Dins aquest arxiu es configuren els Beans que s'utilitzaran, que poden ser controladors, interceptors o classes que afegeixen funcionalitat al framework, com l'extensió per utilitzar anotacions.

Les anotacions que proporciona Spring MVC serveixen per configurar la navegació, els controladors, etc. destacant les següents:

- @Controller, per indicar que la classe és un controlador
- @RequestMapping, on s'indica quina URL captura el controlador o el mètode del controlador
- @SessionAttributes, per declarar variables de sessió

6.4.6. Sistemes de presentació

Spring MVC separa les tecnologies de presentació de la resta del framework utilitzant *ViewResolvers*, que s'utilitzen per renderitzar la vista en funció de la tecnologia utilitzada.

[11] http://jcp.org/en/jsr/detail?id=303

A més de JSP simples o combinats amb JSTL, es permet l'ús de plantilles Velocity, XSLT, Freemark o la integració amb la llibreria Tiles, molt similar a la de Struts. En el cas d'utilitzar XSLT cal que el controlador transformi el model a DOM[12], estenent la classe *AbstractXsltView*.

A l'hora de veure documents, Spring proporciona classes abstractes amb mètodes per construir un document d'un tipus i mostrar-lo sense gaire esforç. En el cas de PDF s'ha d'estendre la classe *AbstractPdfView* i en cas de documents Excel s'ha d'estendre la classe *AbstractExcelView*, per documents generats amb la llibreria POI; o la classe *AbstractJExcelView*, per documents generats amb la llibreria JExcelApi.

Una altra característica del framework és que permet l'ús de temes, que es poden utilitzar per oferir plantilles i estils personalitzats als usuaris de l'aplicació. Per utilitzar els temes s'ha d'indicar a la configuració una o més implementacions de *ThemeResolver*, que s'encarregaran de resoldre quin tema utilitzar, i un arxiu de propietats per cada tema, on s'especifiqui el CSS i altres propietats pròpies del tema.

6.4.7. Documentació

La comunitat de Spring proporciona una documentació molt extensa, en format HTML o PDF, per cada un dels seus mòduls a la seva pàgina web. La documentació, que inclou fragments de codi per facilitar la comprensió, explica com utilitzar cada una de les seves característiques i l'arquitectura que utilitza el framework. A més, s'hi pot trobar l'API en Javadoc i un fòrum on es pot demanar dubtes a la comunitat.

Al tractar-se d'un framework molt utilitzat i que ja du temps al mercat, es troben molts projectes lliures i molta informació a pàgines de desenvolupadors que comparteixen els seus coneixements.

[12] Document Object Model és un model estàndard per representar documents HTML i XML mitjançant objectes, junt amb una interfície per accedir i manipular aquests objectes.

7. JavaServer Faces Framework

JavaServer Faces o JSF és el framework estàndard de Java per desenvolupar aplicacions web. Es tracta d'una especificació, la JSR-314, que proporciona un conjunt de APIs per representar components en una interfície d'usuari dins un entorn web i manejar el seu estat, esdeveniments, validacions, etc. La seva darrera versió és la 2.0 i fou llançada a l'agost de 2009.

Com a especificació, conta amb diverses implementacions, una oficial de Java i altres implementacions fortament lligades al servidor d'aplicacions que s'utilitzi: MyFaces de Apache, RichFaces de Jboss o IceFaces de ICESoft. Aquestes implementacions, a més d'implementar la funcionalitat bàsica especificada a l'estàndard, amplien el conjunt de components i afegeixen funcionalitats.

7.1. Arquitectura

JSF, al igual que Struts, es basa en el model-2 de l'arquitectura MVC, amb la diferència que, en comptes d'utilitzar pàgines (normalment JSP) per la vista, utilitza un arbre de components. L'ús de components permet que el framework sigui independent de protocols específics i de marca, ja que es poden renderitzar en funció del dispositiu on es visualitza. Encara que, la majoria implementacions, s'orienten a l'ús del protocol HTTP i el llenguatge de marca HTML.

L'arquitectura MVC que proposa JSF, representa a la Figura 5, està formada per les següents parts:

Controlador

Es composa d'un controlador central, un o més arxius de configuració i un conjunt de capturadors d'accions. El controlador central està implementat per un Servlet, anomenat *FacesServlet*, que rep totes les peticions dirigides al framework.

Model

Es tracta d'un framework purament per la capa de presentació, pel que la capa del Model la deixa a l'elecció del programador, que pot utilitzar Hibernate, iBatis o qualsevol altre framework de persistència.

Vista

Es composa principalment de l'arbre de components, que es pot renderitzar en HTML o un altre llenguatge de marques, depenent de l'aplicació i del dispositiu client. L'arbre de components es basa en el patró de components, utilitzat també per la biblioteca gràfica Swing de Java, i proporciona una interfície *UIComponent* que han d'implementar tots els components, com *UIForm* o *UIInput*.

Aquest arbre de components és accessible durant qualsevol fase del cicle de vida mitjançant el mètode `getViewRoot()` del context de JSF: *FacesContext*.

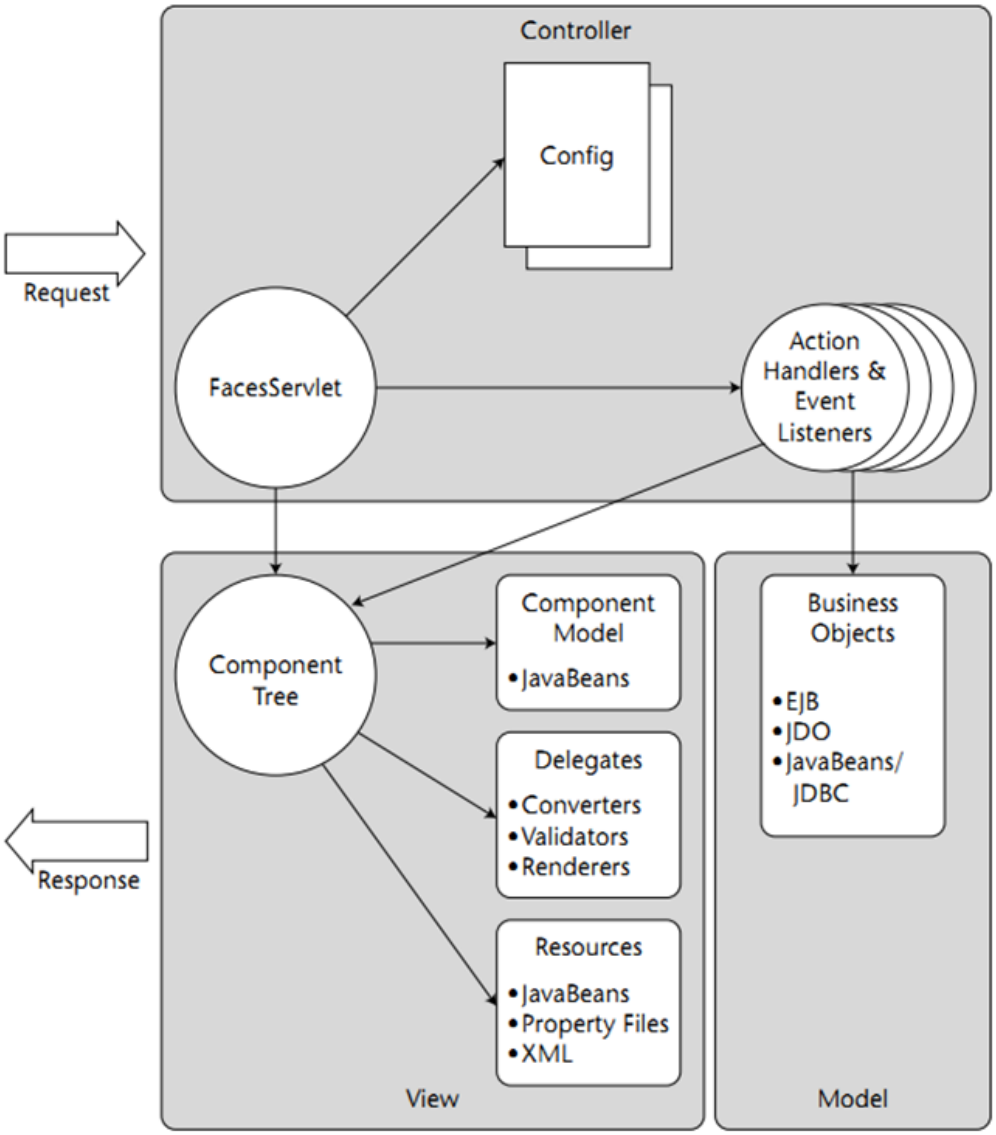


Figura 5. Arquitectura JavaServer Faces

7.2. Cicle de vida d'una petició JSF

Existeixen quatre escenaris per aquest procés en el que el controlador frontal de JSF, el Servlet *FacesServlet*, processa una petició i retorna una resposta:

- Petició Faces genera una resposta Faces. Segueix totes les fases del cicle de vida.
- Petició No-Faces genera una resposta Faces. Es produeix, per exemple, quan un usuari fa clic a un enllaç dins una pàgina sense components JSF que redirigeix a una pàgina amb components JSF. En aquest cas, es passa directament a la fase *Render Response*, ja que no hi ha cap arbre de components en sessió per reconstruir.
- Petició Faces genera una resposta No-Faces. Es dona, per exemple, quan es redirigeix a una altra aplicació o es genera una resposta sense components JSF com un arxiu, un XML, etc. En aquest cas, es salta la fase *Render Response*, ja que es transfereix el control a una altra aplicació.
- Petició No-Faces genera resposta No-Faces. Es tramita la resposta mitjançant un Servlet que no és del framework.

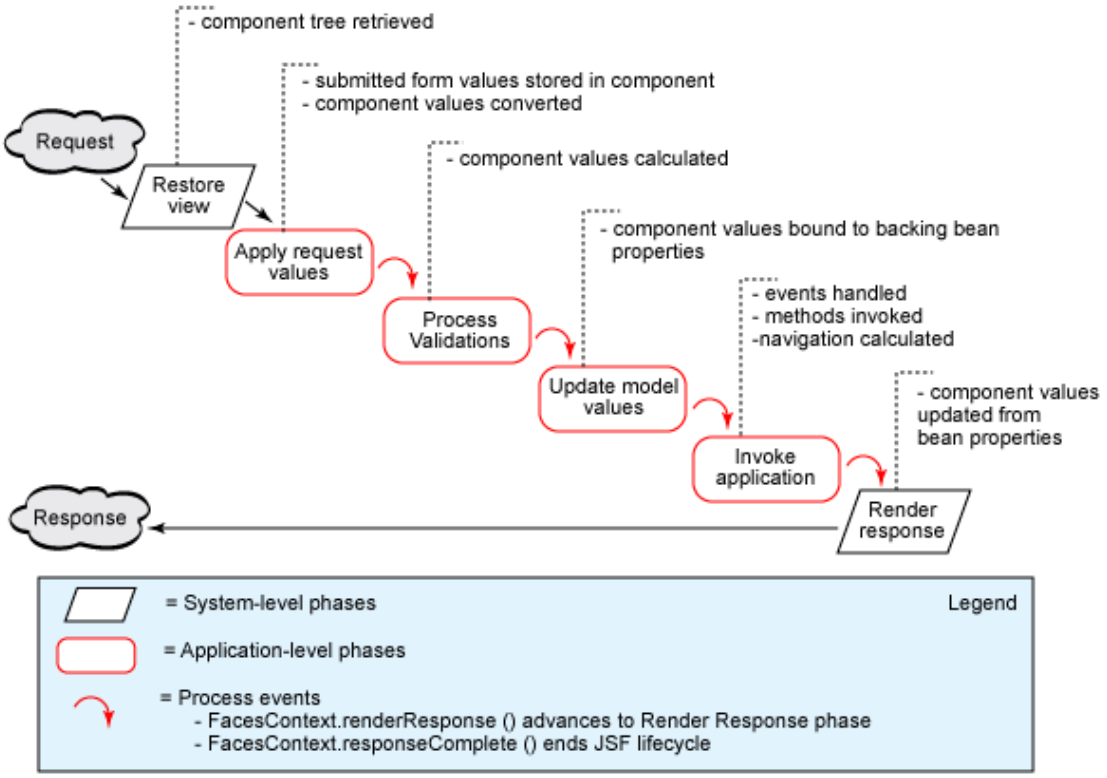


Figura 6. Cicle de vida d'una petició JSF

1. Restaurar la vista.
   Primerament, es comprova el paràmetre inicial en el context extern per verificar que l'estat de l'arbre s'ha guardat prèviament a la petició o la sessió. Aquest paràmetre és l'identificador de la vista i ve determinat pel nom de la pàgina. Si existeix, es restaura l'arbre, i si no existeix, es crea un nou arbre que es guarda a la propietat *viewRoot* del context *FacesContext*.
   Després, es registren els escoltadors i els validadors, així com l'idioma, que, com s'explicarà més endavant, s'obté a partir del context o de la petició.
   Si la petició no conté dades POST o GET, es passa directament a la fase 6 invocant el mètode *renderResponse()* del *FacesContext*.
2. Aplicar els valors de la petició.
   S'actualitzen els components de l'arbre amb les dades de la petició mitjançant el mètode *processDecode* del component arrel, que crida el mateix mètode dels fills recursivament. Aquest mètode s'encarrega de convertir el paràmetre en un objecte.
   Durant aquesta fase es posen a la coa els esdeveniments lligats als components.
3. Processar les validacions
   Es valida cada component a partir dels objectes *Validator* associats, que poden ser proporcionades pel framework o pel programador.
   Si un component provoca un error de validació s'afegeix un missatge a la cua d'errors i es salta a la fase 6, mostrant els errors de validació a la mateixa vista que ha generat la petició.
4. Actualitzar el model
   S'actualitzen les propietats dels controladors o *managed beans* que estiguin associades a un component.

5. Invocar l'aplicació
   L'*ActionListener* s'encarrega de processar la petició, generalment llançada per components *UICommand*, i en determina el controlador que conté el mètode a executar.
   En aquesta fase es determina la vista lògica que s'haurà de renderitzar mitjançant les regles de navegació configurades a l'arxiu *faces-config.xml*.
6. Renderitzar la resposta
   Renderitza l'estat actual de l'arbre.

7.3. Característiques del framework

A continuació s'analitzen les principals característiques que es consideren importants a un framework de presentació i que serviran per comparar-lo amb Struts 2 i Spring MVC.

7.3.1. Internacionalització i18n

JavaServer Faces permet implementar la internacionalització indicant els idiomes permesos, l'idioma per defecte i els arxius de propietats, que contenen els missatges en cada idioma, a l'arxiu de configuració del framework mitjançant les etiquetes *locale-config* i *resource-bundle*:

```
<application>
    <locale-config>
        <default-locale>en</default-locale>
    </locale-config>
    <resource-bundle>
        <base-name>com.mkyong.welcome</base-name>
        <var>msg</var>
    </resource-bundle>
</application>
```

Codi d'exemple 7. Internacionalització a JSF

A diferència de Struts 2 i Spring MVC, JSF no proporciona cap forma de gestionar l'idioma en que es visualitza la petició JSF. En cas de voler aquesta funcionalitat, molt comú a les aplicacions web, el programador ha d'implementar un Controlador de sessió que tingui un llistat amb els possibles idiomes de l'aplicació i una variable que emmagatzemi l'idioma seleccionat per l'usuari.

7.3.2. Proves

A l'hora de realitzar proves dels controladors, ja sigui amb JUnit, TestNG o un altre framework de proves, es pot instanciar directament el controlador des del test, ja que són simples POJOs i disposen dels mètodes per posar cadascuna de les propietats que utilitzen.

En cas de necessitar el context de JSF (*FacesContext* o *ExternalContext*), l'arbre de components (*UIViewRoot*) o altres objectes propis d'una petició JSF, es necessitarà utilitzar un objecte *Mock* durant el test, de manera similar a Spring o Struts. Hi han diverses llibreries que proporcionen aquests objectes *Mock*, com el projecte *jsf-mock* de JBoss Comunity.

També es pot utilitzar el framework de proves JSFUnit de Jboss, que s'integra amb JUnit 4 o Arquillian i permet fer proves completes d'integració i unitàries a aplicacions JSF. Aquest

framework de proves, permet validar els controladors, la navegació, l'arbre de components d'una vista, els missatges d'error associats a entrades de formularis, etc.

7.3.3. Ajax

JSF 2 incorpora suport per utilitzar Ajax de manera oficial, a diferència de la versió 1.2 que no en proporcionava i s'havia d'utilitzar una implementació com RichFaces o Ajax4faces. Per enviar una petició Ajax cal enllaçar un esdeveniment i un escoltador a un component mitjançant l'etiqueta `<f:ajax>`:

```
<h:inputText value="#{bean.text}">
    <f:ajax event="keyup" render="text count"
listener="#{bean.textListener}"/>
</inputText>
<h:outputText id="text" value="#{bean.text}"/>
<h:outputText id="count" value="#{bean.count}"/>
```

Codi d'exemple 8. Ús d'Ajax a JSF

Al Codi d'exemple 8 es crida a un mètode escoltador *textListener*, que rep un argument de tipus *AjaxBehaviorEvent* amb les dades de l'esdeveniment. Aquest mètode actualitzarà les variables *text* i *count* del controlador *bean* i, en haver-se executat, s'actualitzaran els valors dels camps *text* i *count* sense actualitzar tota la pàgina.

L'atribut *render* de l'etiqueta admet identificadors de components o una sèrie de valors definits pel framework, que permeten especificar quin conjunt de components s'actualitzarà en haver-se executat l'esdeveniment.

7.3.4. Validació de formularis

JSF utilitza l'estàndard JSR-303 per dur a terme les validacions dels *Beans* mitjançant anotacions. Per indicar que un camp s'ha de validar a través de JSR-303 cal indicar-ho amb l'etiqueta `<f:validateBean />` dins el component a validar. Així, indicarem al framework que comprovi les regles especificades al controlador mitjançant anotacions:

```
@NotNull
@Size(min=6, max=30)
private String nom;
```

Codi d'exemple 9. Validació mitjançant anotacions JSR-303

També es permet especificar les validacions que ha de complir un component utilitzant els seus atributs. La majoria de components d'introducció d'informació, com `<h:inputText />`, tenen atributs de validació com *required* i *requiredMessage*, que permeten indicar si el component és obligatori i el missatge que es mostrarà en cas de no tenir valor.

Finalment, JSF permet crear validadors personalitzats implementant la interfície *Validator* i indicant al component a validar quina classe el validarà utilitzant l'etiqueta `<f:validator validatorId="com.exemple.EmailValidator"/>`

Per mostrar els missatges d'error només cal incorporar l'etiqueta `<h:messages />` a la vista i JSF s'encarregarà de mostrar els missatges d'error produïts per la validació o conversió de

components. També es poden mostrar missatges personalitzats creant un objecte *FacesMessage* amb el missatge a mostrar i afegint-lo al context.

7.3.5. Configuració del framework

Un projecte que utilitzi JSF ha de declarar el Servlet que fa de controlador central del framework, *FacesServlet*, a l'arxiu *web.xml* i relacionar-hi els patrons URL que utilitzaran peticions JSF (Codi d'exemple 10). A més, hi ha una sèrie de paràmetres del framework que es poden configurar al mateix arxiu mitjançant variables de context.

```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

Codi d'exemple 10. Configuració del framework JSF

Els components del framework, com controladors i regles de navegació, es poden definir a un arxiu anomenat *faces-config.xml* o mitjançant anotacions als propis controladors. D'una manera o una altra, cal especificar quins són els controladors de l'aplicació i l'àmbit d'aquests:

- Àmbit de petició: Les propietats del controlador només tindran validesa durant la petició, és a dir, el seu valor es perdrà una vegada hagi finalitzat aquesta.
- Àmbit de sessió: Les propietats romandran vives durant la sessió de l'usuari.
- Àmbit d'aplicació: Les propietats tenen el mateix valor a tota l'aplicació, és a dir, són compartides per tots els usuaris que hi accedeixin.
- Àmbit de vista: Les propietats tenen el mateix valor mentre l'usuari estigui a la mateixa vista.
- Àmbit personalitzat: Permet utilitzar un àmbit personalitzat per el desenvolupador.
- Sense àmbit: El controlador és instanciat cada vegada que se'l referencia. Té sentit si es tracta d'un controlador que sempre s'utilitza des d'un altre controlador.

La classe que s'encarrega d'emmagatzemar la informació continguda a l'etiqueta `<application>` de l'arxiu *faces-config.xml* és *Application*, de la qual només hi ha una instancia a l'aplicació. Per tant, és la que proporciona informació sobre els recursos de llenguatge, qui captura les peticions, quins són els controladors, etc.

7.3.6. Sistemes de presentació

JSF 2 té com a principal tecnologia de presentació el sistema de plantilles *Facelets*, que utilitza a través de la llibreria *facelets* i un conjunt d'etiquetes. Tot i així, es poden utilitzar les llibreries de components dins pàgines JSP.

Facelets es basa en documents XHTML, on s'hi declaren els components de JSF que formen l'arbre de components per una determinada vista. En aquests documents no s'hi pot escriure codi Java, pel que queda un document net on només hi han etiquetes XML.

Aquest sistema utilitza el patró de vista composta, ja que proporciona etiquetes que faciliten l'ús de plantilles. Així, tenim l'etiqueta `<ui:insert>`, que s'utilitza a la plantilla per indicar on es situarà cada element, i les etiquetes `<ui:composition>` i `<ui:define>`, que s'utilitzen a les pàgines per indicar quina plantilla utilitzen i quin element defineixen, respectivament.

7.3.7. Documentació

Es pot trobar bastanta documentació al lloc oficial d'Oracle:

- API de referència
- Manuals
- Manuals per la creació de projectes d'exemple
- Enllaços a la implementació oficial de JSF: Mojarra
- Enllaços a fòrums i xats IRC de la comunitat de desenvolupadors

A més, conta amb molts de manuals complets a Internet i diversos llibres, tant en format electrònic com en paper, que detallen com funciona i proporcionen exemples de desenvolupament d'aplicacions pas a pas.

8. Comparativa dels frameworks analitzats

Els tres frameworks analitzats tenen bastants punts en comú, com s'ha pogut veure a les característiques analitzades individualment i el cicle de vida que segueix cadascun. En els següents punts es farà una comparació directa de les característiques analitzades als capítols anteriors, així com del cicle de vida, de la forma que implementen els patrons J2EE i la popularitat que tenen.

8.1. Característiques implementades

Els tres implementen les següents característiques de forma similar:

- Internacionalització i18n mitjançant arxius de propietats i etiquetes. Struts 2 i Spring MVC proporcionen interceptors que permeten canviar l'idioma fàcilment, mentre que a JSF s'ha de crear un controlador per implementar aquesta funcionalitat.
- Integració amb frameworks de prova com JUnit o provant directament els controladors, ja que els tres utilitzen POJOs simples.
- Peticions asíncrones incorporant una llibreria Ajax pròpia o adaptada, i etiquetes perquè el seu ús sigui més fàcil.
- JSF i Spring MVC permeten la validació de formularis i propietats dels controladors mitjançant l'especificació JSR-303. Struts 2 permet la validació de forma similar, especificant les regles a arxius XML o amb anotacions, però sense seguir aquesta especificació.

  Els tres frameworks permeten la visualització dels missatges de validació a la vista amb la incorporació d'una etiqueta.
- La configuració dels tres frameworks es fa afegint el controlador central a l'arxiu *web.xml* i un arxiu XML per configurar controladors, regles de navegació, etc. A més, els tres permeten l'ús d'anotacions per minimitzar l'extensió de l'arxiu de configuració, inclús fent innecessari el seu ús.
- Es troba força documentació, tant oficial com de la comunitat d'usuaris, així com exemples i aplicacions lliures.

En l'aspecte en que difereixen més és en el sistema de presentació, ja que JSF és un framework orientat a components i, per tant, tracta el desenvolupament de forma similar als frameworks per aplicacions d'escriptori com Swing o Awt. En canvi, Struts i Spring MVC es centren només en facilitar la relació vista-controladors i la seva navegació. Tot i aquesta diferència, els tres presenten facilitats per l'ús de plantilles.

8.2. Arquitectura

Els frameworks analitzats disposen d'un cicle de vida molt similar, que es pot reflectir en el següent esquema:

1. El controlador central rep la petició.
2. Es determina l'acció que ha de processar la petició, utilitzant un ajudant que té el mapeig entre URL i controladors (patró "Controlador d'aplicació")
3. S'executen els filtres o interceptors abans de l'acció, com la validació de Beans.
4. S'executa l'acció.
5. S'executen els filtres o interceptors després de l'acció, com al punt 3.

6. Es determina quina vista s'ha de presentar, segons la petició i els resultats de l'acció (patró "Controlador d'aplicació")
7. Es renderitza la vista.

Aquest esquema segueix, en gran mesura, el que proposa el patró J2EE de "Servei al treballador" (Veure capítol Servei al treballador *(Service to Worker)* i Distribuïdor de Vista *(Dispatcher View)*):

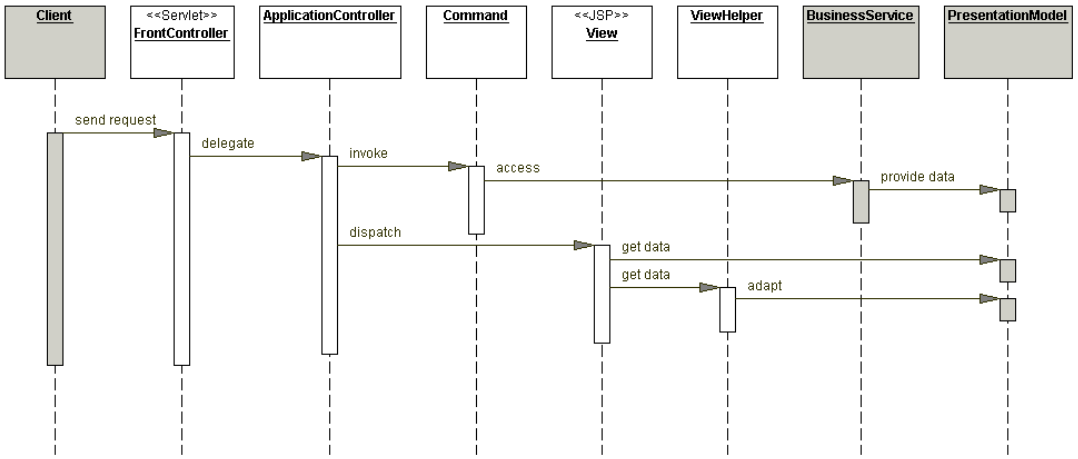


Figura 7. Diagrama de flux del patró "Servei al treballador"

A més d'aquest patró, els tres frameworks implementen, en major o menor fidelitat, els següents patrons:

- Filtre d'intercepció. Ho utilitzen per fer validacions dels Beans, la internacionalització, etc. abans i després de processar una acció.
- Controlador central. Queda implementat pel *FilterDispatcher* a Struts 2, el *DispatcherServlet* a Spring MVC i el *FacesServlet* a JSF.
- Objecte de context. Cap dels tres frameworks obliga a utilitzar d'objectes com *HttpServletRequest* o *HttpServletResponse*, ja que s'encarreguen de mapar-los a *Maps* propis i assignar-los directament a variables del controlador o del context del framework.
- Controlador d'aplicació. Els tres frameworks utilitzen capturadors o *Handlers* per obtenir quina acció s'ha d'executar i quina vista s'ha de mostrar.
- Ajudant de vista. L'ús d'etiquetes personalitzades és utilitzat per els tres frameworks, sobretot per JSF que, al utilitzar *Facelets*, permet que es puguin construir les vistes sense utilitzar cap llenguatge de marca com HTML. A més, tots tres utilitzen llenguatge d'expressions per accedir als Beans més fàcilment.
- Vista composta. Tots tres frameworks permeten utilitzar plantilles, ja sigui a través d'etiquetes, com JSF, o de descripcions XML, com Struts 2.

8.3. Popularitat

A l'hora de comparar la popularitat d'un framework es tindrà en compte els següents aspectes:

- Ofertes de feina que cerquen gent amb coneixements sobre el framework.
- Nombre de cerques que es realitzen a Google sobre el frameworks.

- Nombre de preguntes a la web Stackoverflow[13], una reconeguda pàgina on desenvolupadors de tot tipus de tecnologies demanen i contesten dubtes.

8.3.1. Ofertes de feina

Per realitzar la comparació d'ofertes de feina utilitzarem dues pàgines, referents en quan a currículums i ofertes de feines: LinkedIn[14] i Indeed.com[15].

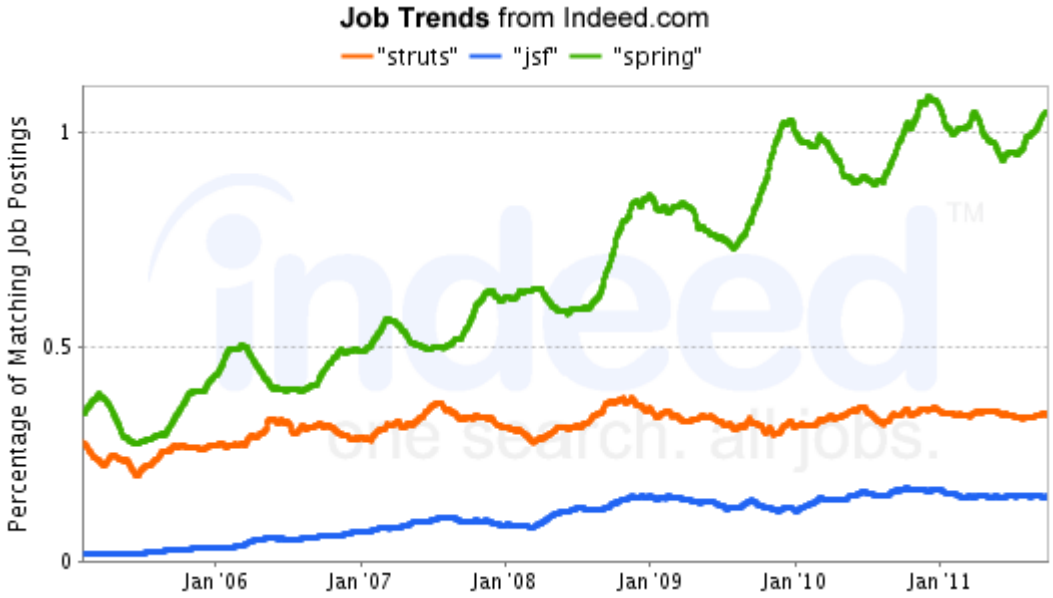


Figura 8. Ofertes a Indeed segons framework

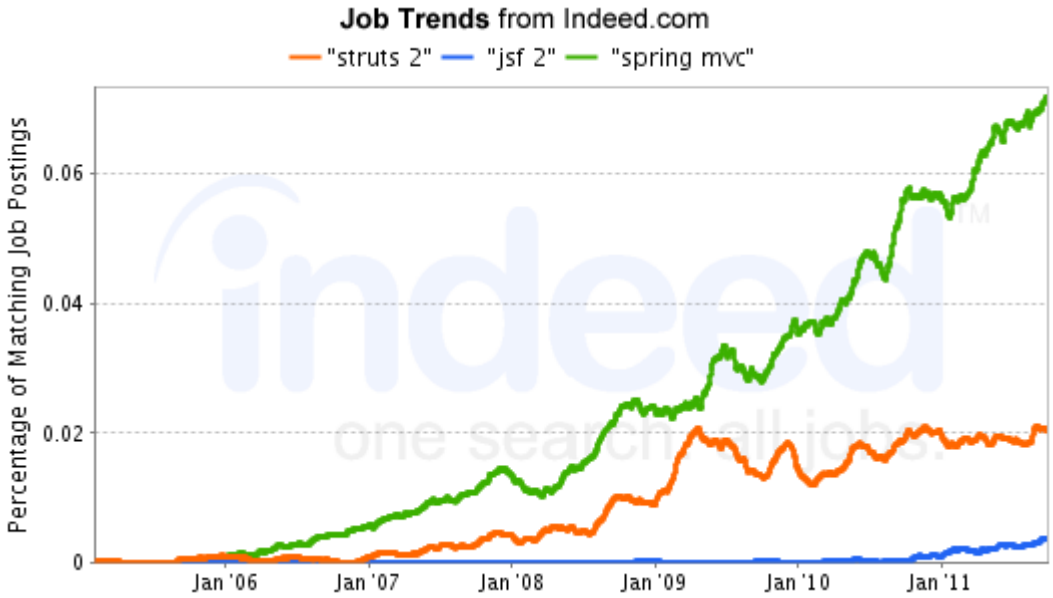


Figura 9. Ofertes a Indeed segons framework i versió

[13] http://stackoverflow.com/tags
[14] http://www.linkedin.com/jsearch
[15] http://www.indeed.com/jobtrends/

A LinkedIn es presenta la següent relació d'ofertes de feina[16]:

Framework	Espanya	Estats Units
Struts	5 ofertes	296 ofertes
Spring	8 ofertes	857 ofertes
JSF	4 ofertes	95 ofertes

Figura 10. Ofertes a LinkedIn segons framework

Com es pot veure a les tres figures, Spring presenta un nombre molt més elevat d'ofertes, pel que sembla que les empreses l'utilitzen molt més que altres dos frameworks. JSF, al contrari, té poca demanda de professionals.

8.3.2. Nombre de cerques

Per comparar el nombre de cerques s'utilitzarà la utilitat Google Trends[17].

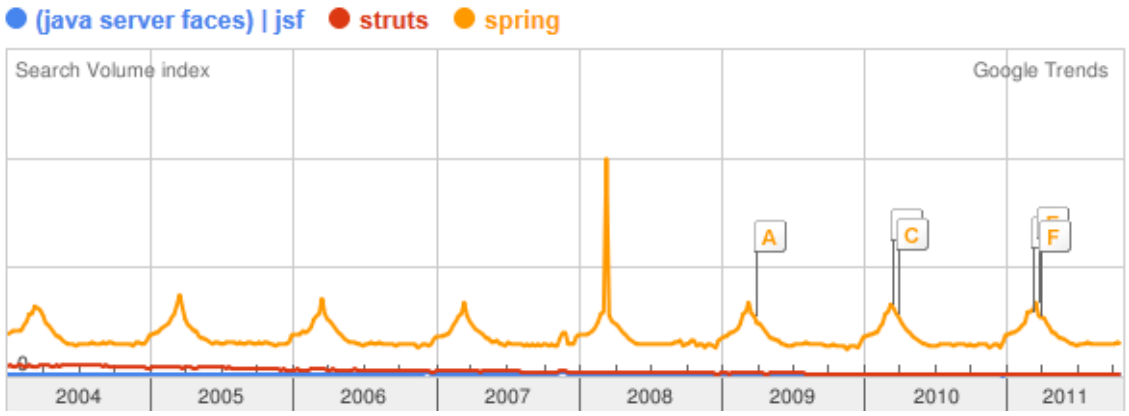


Figura 11. Cerques a Google segons framework

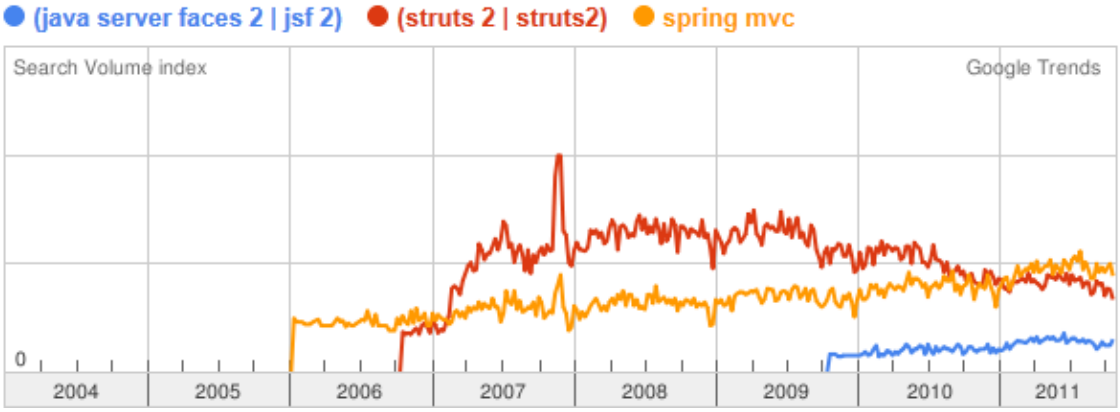


Figura 12. Cerques a Google segons framework i versió

Com queda reflectit a les dues figures, Spring és més cercat a Google que els altres dos frameworks. JSF torna a ser el framework que menys representació té.

[16] Estadístiques extretes el 6 de novembre de 2011
[17] http://www.google.es/trends

8.3.3. Preguntes a StackOferflow

Framework	Nombre de preguntes		
	Totals	Darrera setmana	Darrer mes
Struts 2	2071	24	99
Spring MVC	3750	57	223
JSF 2	2483	69	69

Figura 13. Nombre de preguntes a StackOverflow segons framework i versió

A partir de les dades de la taula deduïm que Spring és el framework que més s'utilitza, ja que és sobre el que més es demana. En aquesta comparativa, és Struts 2 el framework que sembla ser el menys utilitzat.

8.3.4. Conclusió

De les comparatives anteriors podem extreure la conclusió que Spring és el framework més popular i JSF el menys popular. Això no vol dir que un sigui millor que un altre, sino que hi ha més gent que l'ha elegit, ja sigui perquè fa més temps que és al mercat, perquè proporciona més funcionalitat, perquè és el que millor funciona o perquè és el més fàcil d'aprendre.

9. Conclusió de l'anàlisi comparatiu

En aquest anàlisi he pogut observar que aquests frameworks tenen molt en comú i, per tant, penso que són un exemple a seguir en l'anàlisi i desenvolupament del meu framework. En concret, trobo que un bon framework per la capa de presentació ha de tenir les 7 característiques que s'han analitzat en més profunditat.

Principalment, la diferència més gran entre els tres frameworks és que JSF s'orienta a components, fet que dificulta l'aprenentatge, ja que un desenvolupador acostumat a treballar directament en Servlets i JSP (com és el meu cas) no veu trivial el pas a components. Això és degut a que el patró de components és més propi d'aplicacions d'escriptori que d'aplicacions web.

Tot i així, és cert que una vegada entès i après com s'utilitza JSF, el desenvolupament d'una aplicació sol ser més ràpid, ja que proporciona molts tipus de components que s'utilitzen a les aplicacions (calendaris, taules de dades, etc.) i s' estalvia la feina d'implementar-los o cercar llibreries de tercers.

També cal dir que m'ha ajudat molt el fet d'estudiar els patrons J2EE, ja que, tot i aplicar (sense saber-ho) alguns d'ells, són pràctiques que ajuden a tenir aplicacions amb un codi estructurat, net i clar. El framework a desenvolupar farà un ús intensiu dels patrons analitzats.

Disseny del framework Jewel

El framework a dissenyar en aquesta part, Jewel Framework (**J**ava **E**asy **We**b **L**ight Framework), té com a principal objectiu facilitar el desenvolupament d'aplicacions web en J2EE seguint el model MVC i una sèrie de patrons J2EE estudiats a l'anterior part. També s'utilitzarà els coneixements adquirits durant l'anàlisi dels frameworks actuals.

Abans de començar amb el disseny cal definir què contindrà i què no contindrà el framework mitjançant la descripció de l'abast.

10. Abast de Jewel

A partir de la comparativa i l'anàlisi dels frameworks dels capítols anteriors es determinen quines són les característiques més importants en un framework per la capa de presentació. Tenint en compte això i la durada del projecte, Jewel implementarà les següents característiques:

- Arquitectura MVC basada en accions: Aquesta característica, comú a tots els frameworks analitzats, serà implementada de forma similar a JSF o Spring: un controlador central o Servlet rebrà les peticions, les tractarà i invocarà el controlador d'aplicacions perquè executi les accions definides pel programador a les classes controladores.
- Internacionalització i18n: La internacionalització d'aplicacions és un punt molt important que implementen tots els frameworks analitzats i que Jewel donarà suport amb les següents funcionalitats:
    - Lectura de missatges localitzats a arxius de propietats.
    - Presentació dels missatges en l'idioma seleccionat mitjançant l'ús d'etiquetes personalitzades.
    - Utilització d'un filtre propi del framework per facilitar el canvi d'idioma de l'aplicació. A més, es proporcionarà una etiqueta que implementarà la funcionalitat per la selecció d'idioma a l'aplicació.
- Configuració mitjançant XML i anotacions: La configuració principal del framework es farà mitjançant un arxiu XML basat en una definició d'esquema XML o XSD[18] proporcionada amb el framework. D'altra banda, la configuració dels controladors, les seves accions i els filtres es farà mitjançant anotacions a les pròpies classes Java que desenvolupi el programador.
- Llibreria d'etiquetes: Es proporcionarà un conjunt d'etiquetes per facilitar la creació de les vistes i cobrir aspectes com els següents: tractar les accions mitjançant formularis o enllaços, la internacionalització de texts, presentar llistats de resultats fàcilment, etc.

[18] XML Schema Definition (XSD): Llenguatge d'esquema utilitzat per definir l'estructura i les restriccions a un arxiu.

- Validació de formularis al costat del client: Les etiquetes proporcionades faran ús d'atributs on es podrà definir la validació de camps obligatoris al enviar un formulari.
- Proves de les aplicacions: Els controladors d'accions seran POJOs simples amb anotacions, pel que les es podran provar fàcilment amb frameworks com JUnit. A més, l'ús d'un objecte de context facilitarà les proves sense la necessitat d'un servidor d'aplicacions.
- Vistes compostes: Es podrà utilitzar plantilles amb l'ajuda de la llibreria Apache Tiles, que es podrà incorporar al framework fàcilment. S'haurà d'indicar quin tipus de vista presenta una acció al programa-la, tal i com es veurà més endavant.
- Suport de peticions AJAX-JSON: Es permetrà definir accions que retornin una resposta JSON, pel que serà fàcil pel programador afegir funcionalitats AJAX a l'aplicació mitjançant frameworks Javascript com JQuery. Jewel només proporcionarà la facilitat per retornar un resultat JSON i una etiqueta que generi la URL on efectuar la petició a l'acció Ajax.
- Protecció davant reenviament de formularis: El framework proporcionarà un filtre, que es podrà activar o desactivar, per comprovar si un formulari s'està reenviant. Això es fa seguint l'estratègia "validació de mostra sincronitzada" que s'ha vist a l'apartat "1.2. Males pràctiques".
- Seguretat: El programador podrà definir quins rols tenen i no tenen permís per executar les accions. Per comprovar-ho s'utilitzarà els mètodes de la petició que implementen JAAS[19].
- Independència de servidor d'aplicacions: Jewel no ha de dependre del servidor d'aplicacions on s'utilitzi, mentre aquest tingui suport per JEE6 i Java 1.6+. Per tant, es podrà utilitzar en Jboss, Tomcat, Glassfish, etc.
- Documentació: Jewel comptarà amb una documentació completa o guia pel desenvolupador que explicarà com utilitzar el framework. A més, el codi del framework estarà documentat mitjançant un Javadoc extens i de qualitat.

D'altra banda, és important dir que el framework a desenvolupar no tindrà les següents característiques:

- Llibreria Javascript: Jewel utilitzarà jQuery per implementar algunes funcionalitats com l'entrada de dades de tipus data mitjançant un calendari o *datepicker*. Per tant, si l'usuari utilitza jQuery es considera innecessari afegir més funcionalitats Javascript de la que aporta aquest framework.
- Validació mitjançant JSR-303: Jewel només proporciona validació de camps al costat del client. En cas de voler validació al costat del servidor, recomanat per gairebé tota aplicació, ho haurà d'implementar el propi programador de la manera que consideri més oportuna.
- Configuració de controladors i accions mitjançant XML: Es consideren les anotacions una font de configuració molt més neta i clara que els arxius XML, per això Jewel es

[19] JAAS – Java Authentication and Authorization Service. Framework de seguretat estàndard de Java per controlar la seguretat centrada en usuaris i rols. Jboss, Tomcat i Glassfish ho suporten per el control d'accés a usuaris i permisos.

limitarà a utilitzar un únic arxiu de configuració XML per la configuració general del framework.

- Navegació controlada: No es podrà definir el flux de navegació per controlar de quina vista a quines vistes es pot navegar, ja que no es considera una característica essencial i el programador ho pot controlar fàcilment des de les accions.
- Entrada i registre d'usuaris: Jewel no determina com s'autenticaran els usuaris a l'aplicació, ni si hi haurà rols o no. L'únic que permetrà és la definició de permisos d'execució per les accions.

El producte final constarà de les següents parts:

- Fitxer JAR[20] que contindrà totes les classes i recursos necessaris per utilitzar Jewel.
- Fitxer ZIP amb les dependències necessàries i opcionals per utilitzar Jewel que es descriuen al capítol 12.1.
- Fitxer ZIP amb els recursos necessàries per executar Jewel com el TLD de la llibreria d'etiquetes i XSD per la configuració XML.
- Fitxer ZIP amb el Javadoc del framework.
- Fitxer ZIP amb el codi font del framework.
- Fitxer WAR[21] amb l'aplicació d'exemple i el seu codi font.

[20] JAR - Java Archive: Arxiu comprimit en format ZIP que conté els arxius .class d'un projecte Java, la seva metadata i els recursos que necessita.
[21] WAR - Web application Archive: Arxiu comprimit en format ZIP que conté un conjunt d'arxius JSP, Servlets, classes Java i altres recursos que formen una aplicació web que serà desplegada a un servidor d'aplicacions com Jboss o Tomcat.

11. Arquitectura de Jewel

Jewel seguirà una arquitectura MVC de model 2, al igual que els frameworks analitzats, que separarà clarament la Vista del Controlador i el Model. A més, seguirà alguns dels patrons J2EE analitzats al primer capítol.

11.1. Patró MVC

Aquest framework proporcionarà l'ajuda necessària per implementar, de forma separada, les parts de Vista i Controlador del patró MVC. En quan al Model, es deixarà a l'elecció del programador, podent utilitzar EJBs, DAOs amb Hibernate, etc.

Jewel permetrà implementar la Vista en arxius JSP o plantilles Apache Tiles. Però també facilitarà una interfície que el programador pot implementar per utilitzar altres tecnologies de presentació. Això és possible gràcies que a que les accions que el programador faci han de retornar, a més del model que s'utilitzarà i el nom lògic per la vista, el tipus de vista, que pot ser un dels tipus proporcionats per Jewel o un de personalitzat que implementi la interfície org.jewel.core.*ResultDispatcher*.

Jewel proporcionarà els següents tipus de presentadors de resultats:

- Resultats en pàgines JSP: La vista quedarà implementada per un arxiu JSP, la ruta del qual s'haurà d'indicar al resultat de l'acció. Un cop processada l'acció, Jewel s'encarregarà de redirigir al JSP amb el model obtingut.
- Resultats en plantilles Tiles: La vista quedarà implementada per una plantilla seguint l'estructura que Apache Tiles proposa. El nom lògic de la vista i el model s'ha d'indicar al resultat de l'acció.
- Resultat JSON[22]: El resultat de la petició es presenta en format JSON, és a dir, es retornarà una resposta JSON que contindrà el model obtingut a l'acció. Aquest tipus de resposta és adequat a peticions Ajax des de Javascript.

La part Controladora estarà formada pel nucli de Jewel i les classes que haurà d'implementar el desenvolupador en base a un conjunt de regles de desenvolupament descrites al capítol 3.

11.2. Patrons J2EE

Jewel farà ús de bona part dels patrons estudiats al primer capítol, ja que proporcionen bones pràctiques que serviran per estructurar el framework.

Filtre d'intercepció

El framework permetrà manipular una petició mitjançant filtres que es podran executar abans i/o després d'executar una acció. S'implementarà una estratègia personalitzada, però més tancada que la proposada pel patró, ja que aquests filtres s'executaran sobre el propi framework i només tindran accés a l'objecte de context. En el cas que es vulgui tenir accés a tota la petició ja existeixen els filtres de J2EE que es poden configurar a l'arxiu web.xml.

[22] JSON – JavaScript Object Notation: Format d'intercanvi de dades molt utilitzat en peticions Ajax JavaScript, com alternativa a XML, degut a la seva lleugeresa i simplicitat.

El controlador d'aplicació s'encarregarà d'inicialitzar el gestor de filtres, implementat per la classe *org.jewel.core.FilterManager*, i invocar-lo abans i després de cada acció. El gestor de filtres guardarà un llistat dels filtres actius, tant els creats per l'usuari com els del propi framework.

Es permetrà activar i desactivar els filtres des de l'arxiu de configuració de Jewel, seguint la bona pràctica del patró per evitar haver de tocar el codi Java. L'ordre en que s'executaran els filtres també es podrà definir a la configuració.

Controlador central

La classe *org.jewel.core.JewelServlet* del nucli actua de punt d'accés per totes les peticions que rep el framework. Aquesta classe implementa l'estratègia controlador Servlet estenent la classe *HttpServlet* de J2EE i delega el control al controlador d'aplicació per executar l'acció i per presentar la vista.

Objecte de context

Aquest patró ve implementat per la classe *org.jewel.core.RequestContext*, que conté les dades que es necessitaran durant l'execució de l'acció: l'idioma actual, el token i els paràmetres enviats des de la vista.

L'estratègia d'implementació serà mixta entre les estratègies POJO i Map, és a dir, la classe *RequestContet* tindrà un mapa de paràmetres variables a cada petició i una sèrie de paràmetres fixes, com l'idioma, el token i la URL de destí.

Aquest objecte de context permetrà que el controlador d'aplicació es pugui executar independentment del protocol que s'utilitzi, facilitant així les proves de les accions.

Controlador d'aplicació

Aquest és el patró que es relaciona amb els patrons anteriors: objecte de context i controlador central. El controlador central es comunica amb el controlador d'aplicació, implementat per la classe *org.jewel.core.ApplicationController*, passant-li l'objecte de context.

Per gestionar les accions, aquest patró implementarà l'estratègia de capturador d'accions sense processador d'accions. En aquesta estratègia el controlador d'aplicació es comunica amb el gestor d'accions, implementat per la classe *org.jewel.core.ActionManager*, obtenint l'acció adequada a la petició i executant-la.

Per tal de gestionar les vistes s'implementarà l'estratègia de capturador de vistes, on el controlador d'aplicació es comunica amb el gestor de vistes, implementat per la classe *org.jewel.core.ViewManager*, per obtenir la classe que implementa la interfície *ResultDispatcher* i executar-la per redirigir a la vista adequada segons el resultat de l'acció.

Ajudant de vista

Per tal d'implementar aquest patró i ajudar al programador a separar la vista de la lògica de presentació es proporcionarà una llibreria d'etiquetes, seguint l'estratègia ajudant d'etiquetes personalitzades. Aquestes etiquetes tenen una doble finalitat: per un costat, evitar l'ús de scriptlets a les vistes facilitant el posterior manteniment d'aquestes; per l'altre, facilitar l'ús del framework i tasques com la validació de formularis.

Un exemple de la primera finalitat són les etiquetes `jwl:dataTable` i `jwl:dataList`, que permetran renderitzar una taula i un llistat de resultats a partir d'un objecte `java.util.Collection`. Un exemple de la segona finalitat són les etiquetes `jwl:commandLink` i `jwl:commandButton` que permeten preparar la vista per comunicar-se directament amb les accions al executar un formulari o un enllaç.

Vista composta

Aquest patró permet al programador evitar la tasca de mantenir codi duplicat a les vistes, que normalment es produeix a les capçaleres, peus, menús, etc. Jewel implementa aquest patró mitjançant la llibreria Apache Tiles, de manera que es podrà utilitzar arxius JSP simples o plantilles definides a l'arxiu de configuració d'aquesta llibreria: *tiles-config.xml*.

Servei al treballador

S'utilitzarà el patró de servei al treballador amb comanda, on la comanda o acció l'implementa el desenvolupador utilitzant un mètode marcat amb l'anotació @Action dins un POJO amb l'anotació @Controller, com es veurà al capítol 12.3. Aquest mètode és l'encarregat de comunicar-se amb la capa de negoci (la qual és de lliure implementació).

L'acció determinarà quina és la vista que mostrarà el resultat i el seu tipus, pel que una mateixa acció pot derivar a diferents vistes en funció del codi executat i els paràmetres.

11.3. Diagrama de classes

El framework es dividirà en dues parts principals: el nucli i la llibreria d'etiquetes, implementades pels paquets *org.jewel.core* i *org.jewel.taglib* respectivament. A més, el framework disposarà d'altres paquets que serviran per ajudar a la funcionalitat dels anteriors:

- *org.jewel.annotations*: Conté les anotacions que farà servir el desenvolupador per definir i configurar els filtres, els controladors i les accions d'aquests.
- *org.jewel.exceptions*: Conté les excepcions que es poden produir al framework.
- *org.jewel.util*: Conjunt de classes amb mètodes útils per la resta de framework.

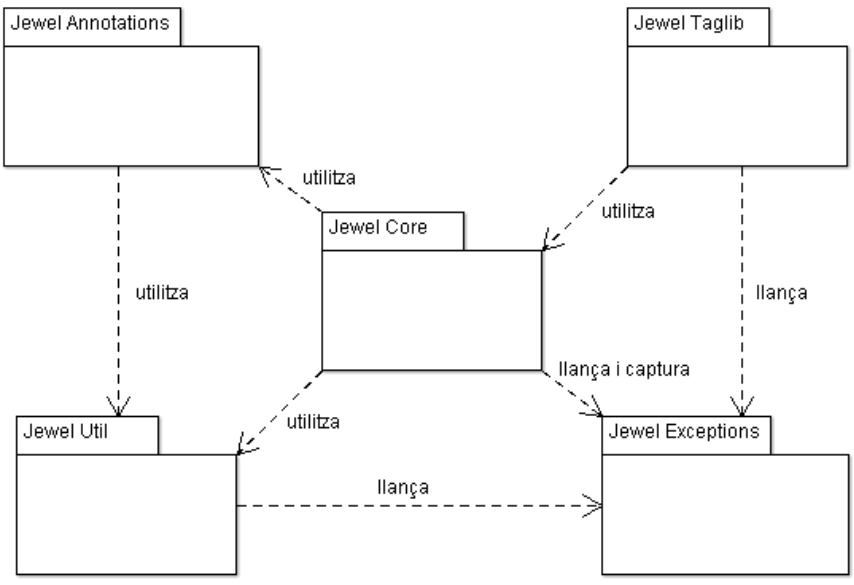


Figura 14. Diagrama de classes: Relació entre els paquets

11.3.1. Nucli del framework

El nucli de Jewel està format per aquelles classes imprescindibles i que posseeixen la funcionalitat principal del framework. Cal destacar la classe *JewelServlet*, que implementa el controlador central i és el punt d'entrada al framework, i la classe *ApplicationController*, que implementa el controlador d'aplicació. Aquestes dues classes són les que relacionen, utilitzen i invoquen les demès classes, tal i com es veu a la Figura 15.

JewelServlet

Aquesta classe fa la funció de controlador central i estén la classe *javax.servlet.http.HttpServlet* de J2EE. S'encarrega d'instanciar el controlador d'aplicació quan s'inicialitza el Servlet i de processar les peticions quan les rep. Al processar una petició crea l'objecte de context i invoca el controlador d'aplicació per executar l'acció i, posteriorment, per presentar la vista.

ApplicationController

És la classe que implementa el patró controlador d'aplicació i s'encarrega d'inicialitzar el framework, gestionar les accions, gestionar els filtres i gestionar les vistes. Segueix el patró Singleton[23], ja que només cal tenir una instància de l'objecte per tota l'aplicació.

Per gestionar les accions es comunica amb el gestor d'accions, per recuperar l'acció a executar i poder-la executar; el gestor de vistes, per recuperar la classe que implementa *ResultDisptacher* i s'encarregarà de processar i presentar el resultat; i el gestor de filtres, que s'encarregarà d'executar els filtres actius abans i després de processar l'acció.

A més, s'encarrega d'obtenir els paràmetres de l'objecte de context i assignar-los als arguments del mètode que invoca l'acció o als controladors de sessió. Es pot veure el flux que segueix al processar una acció a la Figura 19.

ApplicationConfiguration

Representa la configuració de l'arxiu de configuració *jewel-config.xml* del framework. S'inicialitza quan es configura el controlador d'aplicació la primera vegada que el controlador central el sol·licita a la seva inicialització. Conté una instancia de la classe *LocaleConfiguration* on guarda la informació de la configuració d'idiomes a l'aplicació.

ViewManager

És l'encarregat de gestionar la presentació de les vistes obtenint la classe que implementa *ResultDispatcher* i el programador ha indicat com a resultat de l'acció.

[23] Singleton: és un patró de disseny que evita la creació d'objectes d'una classe des de fora de la mateixa i només deixa tenir una instància de la classe a tota l'aplicació. Això s'aconsegueix fent el constructor privat i permetent només crear i obtenir aquesta única instància des del mètode getInstance() d'aquesta mateixa classe.

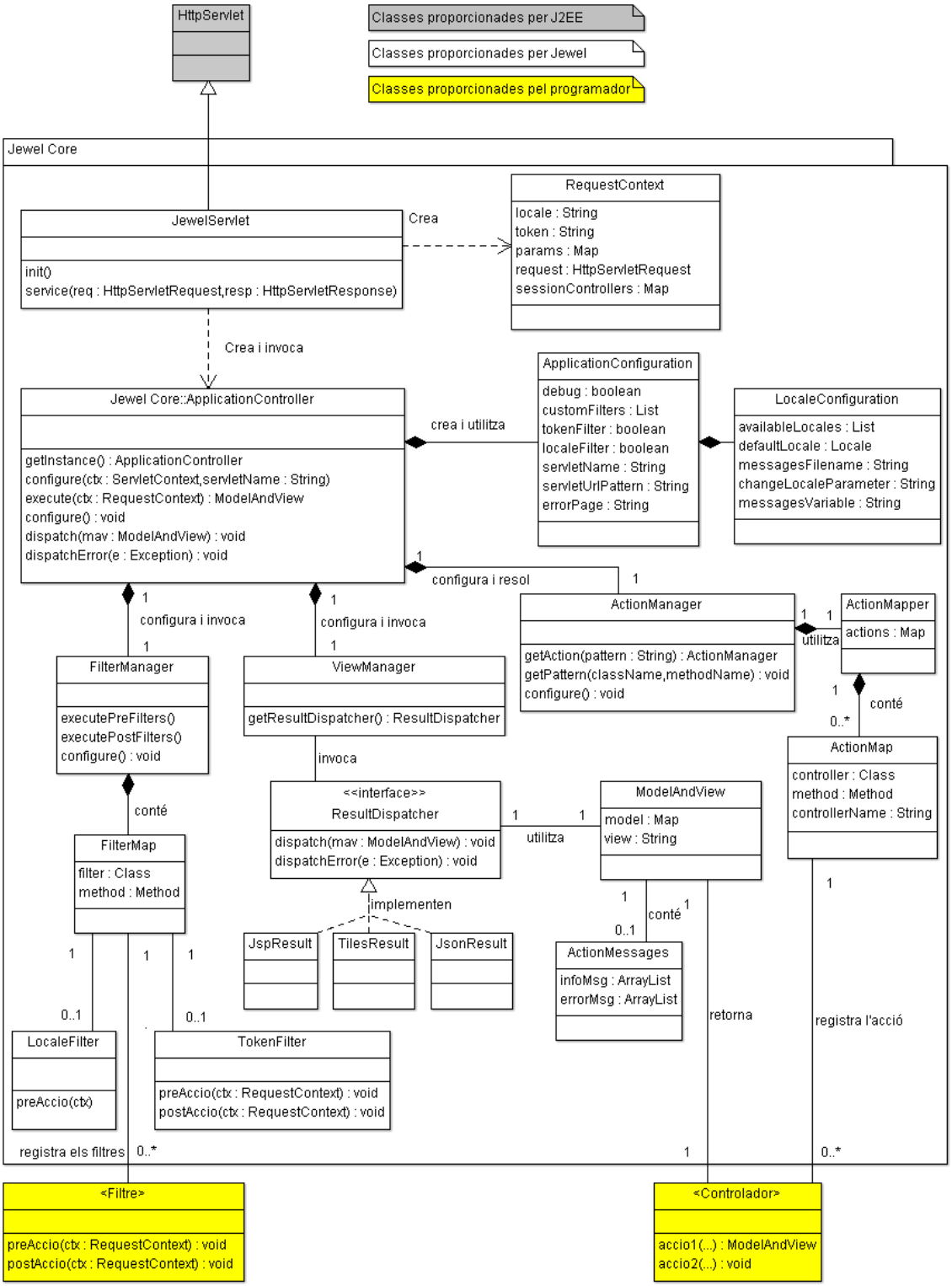


Figura 15. Diagrama de classes: Nucli del framework

ActionManager

S'encarrega de gestionar les accions de l'aplicació i s'instanciarà al moment de configurar-se el controlador d'aplicació. Al inicialitzar-se, cerca totes les accions de l'aplicació, és a dir, tots els mètodes anotats com @Action que estiguin a una classe controladora (anotada com @Controller). Per registrar una acció, es guardarà la relació entre el mètode i el patró URL que s'utilitzarà per cridar aquesta acció.

Quan es llença el procés de petició, el controlador d'aplicació consultarà al gestor d'accions quina acció compleix el patró URL de la petició i recuperarà el mètode i la classe a executar.

FilterManager

És l'encarregat de gestionar els filtres de l'aplicació, tant els inclosos al propi framework com els creats per l'usuari. Ambdós tipus de filtres es podran activar i desactivar des de l'arxiu de configuració, permetent l'opció d'auto-escanejar els filtres de l'usuari. Si es deixa a Jewel la missió de cercar els filtres de l'usuari es cercaran, en ordre indeterminat, totes les classes amb l'anotació *@Filter* i els seus mètodes anotats com *@PreAction*, per executar-se abans de l'acció, o *@PostAction*, per executar-se després de l'acció.

Aquesta classe guarda un llistat de filtres amb els seus mètodes que s'executen abans de l'acció i un altre llistat per els filtres amb els seus mètodes que s'executen després. Aquests llistats són els que s'utilitzaran als mètodes `executePreFilters()` i `executePostFilters()` que es cridaran per executar els filtres seguint el diagrama de seqüència de la Figura 20.

Jewel incorpora dos filtres:

- LocaleFilter: Cerca un paràmetre a la petició que indica que l'usuari ha canviat l'idioma, i si no coincideix amb l'idioma actual fa els canvis pertinents perquè a partir d'aquest moment tots els missatges apareguin en el nou idioma.
- TokenFilter: Comprova que el formulari no s'estigui reenviant seguint l'estratègia validació de mostra sincronitzada: Es crea un valor, calculat a partir del temps actual, després de processar l'acció i es guarda tant a la sessió de l'usuari com a un camp ocult als formularis de la pàgina. Al enviar un formulari es comprova que el valor enviat i el guardat en sessió coincideixin. Si no coincideixen es mostrarà l'error a la pàgina d'error per defecte de l'aplicació.

ModelAndView

És la classe que s'utilitza com a resposta d'una acció per incorporar el model i indicar la vista i el tipus de vista que s'utilitzarà per presentar-lo:

- Model: mapa amb el nom, que servirà per identificar l'objecte, i el valor o objecte en si.
- Nom de la vista: identificador que indica quina és la vista a mostrar en funció del tipus.
- Tipus de vista: és la classe Java que implementa *ResultDispatcher* i s'utilitzarà per redirigir a la vista indicada en el format adequat. Aquesta classe pot ser *TilesResultDispatcher*, *JspResultDispatcher*, *JsonResultDispatcher*, o bé una classe implementada per l'usuari.

ActionMessages

S'utilitza per afegir fàcilment missatges d'error o informació durant l'execució de l'acció i poder-los mostrar a la vista utilitzant l'etiqueta `h:messages`.

RequestContext

És la classe que implementa el patró objecte de context. El crea el controlador frontal al rebre una petició i, a partir dels objectes de petició i resposta, emmagatzema un conjunt de paràmetres que s'utilitzaran al llarg de la petició.

Aquest objecte serà el que utilitzin la resta de classes del framework i els filtres durant el processament d'una acció.

ResultDispatcher

És una interfície que proporciona els mètodes a implementar que s'utilitzaran per presentar un resultat a una vista. La classe *ViewManager* obté la implementació que s'indica al resultat d'executar una acció, podent esser implementada pel propi programador o utilitzar una de les que inclou Jewel:

- JspResultDispatcher: El resultat és una pàgina JSP, pel que es redirigeix la petició a un arxiu JSP indicat al *ModelAndView* retornat per l'acció. S'incorpora el model com atributs de la petició per a que siguin accessibles des de la vista.
- TilesResultDispatcher: El resultat és una plantilla que utilitza la llibreria Apache Tiles i està definida a l'arxiu *tiles-config.xml* de l'aplicació. Aquesta és la implementació que permet utilitzar la vista composta. El model també s'afegeix com a atributs de la petició.
- JsonResultDispatcher: El resultat és una resposta de tipus JSON, normalment utilitzada per una petició Ajax. Es converteix el model a objectes JSON mitjançant la llibreria GSon[24], de manera que una funció Javascript pugui recollir els resultats i tractar-los.

Controlador

El controlador és la classe que ha de programar el desenvolupador i conté les accions. No forma part del nucli del framework, però és una de les parts essencials i la que donarà la funcionalitat pròpia de les aplicacions que utilitzin Jewel.

11.3.2. Llibreria d'etiquetes

Jewel proporciona una llibreria d'etiquetes que s'utilitzaran per facilitar alguns aspectes de la creació de les vistes i, sobretot, per facilitar la utilització del framework. Aquesta llibreria està formada per un TLD[25], on es defineixen totes les etiquetes, i el conjunt de classes del paquet *org.jewel.taglib* que implementen les etiquetes i es mostren a la Figura 16.

La classe principal és *org.jewel.taglib.AbstractHtmlTag*, que estén la classe *javax.servlet.jsp.tagext.TagSupport* i afegeix atributs comuns a totes les etiquetes:

Atributs d'AbstractHtmlTag	Descripció de l'atribut
id	Atribut 'id' de les etiquetes HTML.
title	Atribut 'title' de les etiquetes HTML.
style	Atribut 'style' de les etiquetes HTML.

[24] Gson: Llibreria de Google utilitzada per convertir objectes Java a objectes JSON.
[25] Tag Library Definition (TLD): Definició d'una llibreria d'etiquetes mitjançant un arxiu XML que segueix l'esquema definit a http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd

styleClass	Atribut 'class' de les etiquetes HTML.

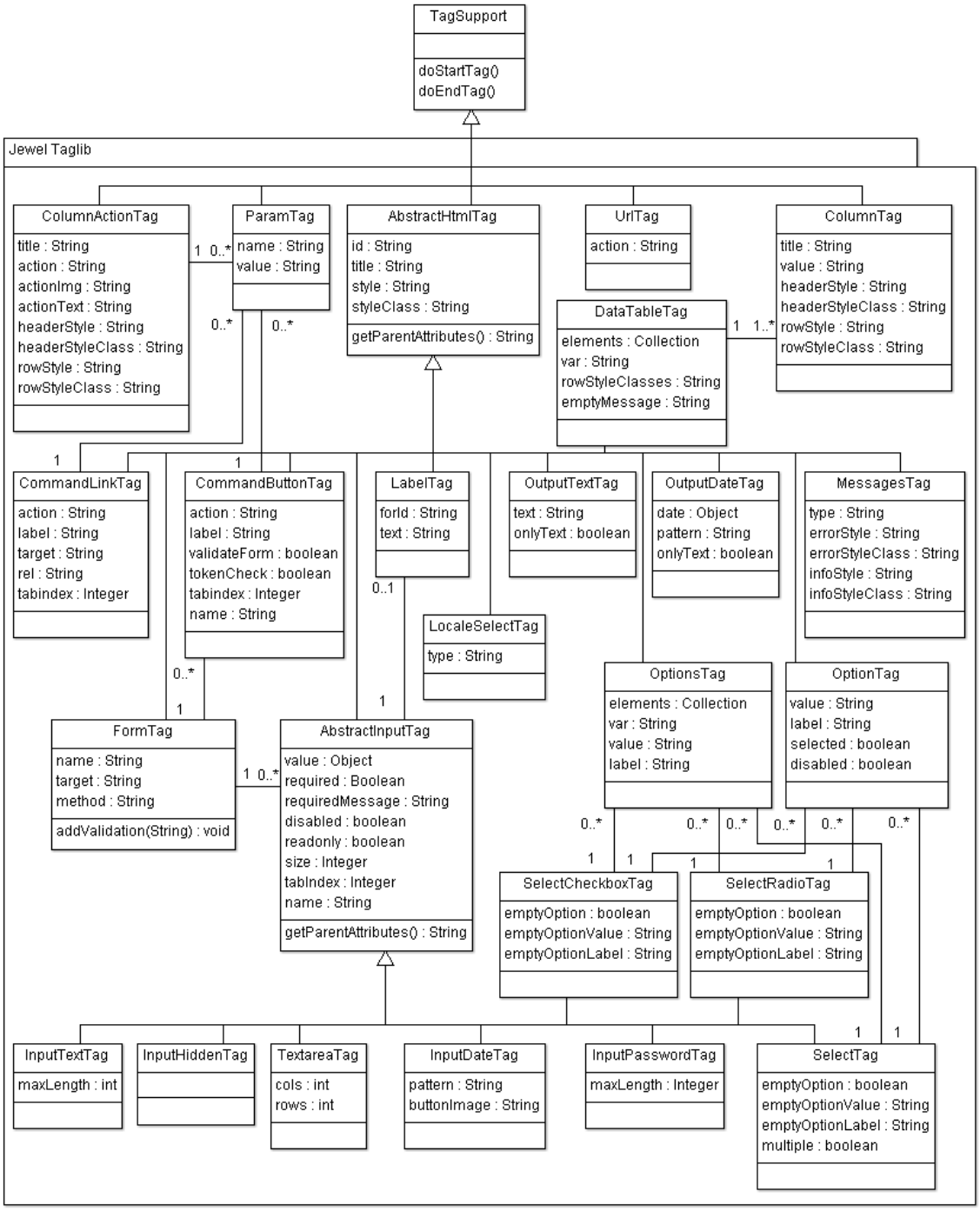


Figura 16. Diagrama de classes: Llibreria d'etiquetes

Classe	**FormTag**	**Classe pare**	AbstractHtmlTag
Ús	`<jwl:form name="frm_cerca" ...>...</jwl:form>`		
Descripció	Etiqueta que representa un formulari. És necessària per incloure les etiquetes que estenen de *AbstractInputTag* i els *CommandButtonTag*.		
Atribut	**Descripció de l'atribut**		
name	Atribut 'name' de l'etiqueta <form> generada que identifica el formulari.		
method	Atribut 'method' de l'etiqueta <form> generada que indica si el formulari seguirà el mètode GET o POST. En cas de no indicar-ho, el formulari seguirà el mètode POST per defecte.		
target	Atribut 'target' de l'etiqueta <form> generada.		

Classe	**MessagesTag**	**Classe pare**	AbstractHtmlTag
Ús	`<jwl:messages type="ERROR|INFO|ALL" ... />`		
Descripció	Etiqueta per mostrar tots els missatges d'error i/o informació que l'usuari afegeixi al context durant l'acció.		
Atribut	**Descripció de l'atribut**		
type	Serveix per definir quins tipus de missatges es vol mostrar: error, informació o tots.		
errorStyle	Atribut 'style' de les etiquetes generades per mostrar missatges d'error.		
errorStyleClass	Atribut 'class' de les etiquetes generades per mostrar missatges d'error.		
infoStyle	Atribut 'style' de les etiquetes generades per mostrar missatges d'informació.		
infoStyleClass	Atribut 'class' de les etiquetes generades per mostrar missatges d'informació.		

Classe	**DataTableTag**	**Classe pare**	AbstractHtmlTag

Ús	`<jwl:dataTable elements="#{persones} var="persona" ...>`
	`...`
	`</jwl:dataTable>`		

Descripció	S'utilitza per mostrar una taula amb les dades dels elements d'un objecte *java.util.Collection*. Està molt relacionat amb les etiquetes *ColumnTag* i *ColumnActionTag*, que serviran per presentar les cel·les de la taula amb les dades de cada element i les accions possibles sobre aquests.		
Atribut	**Descripció de l'atribut**		
elements	Elements continguts en un objecte *java.util.Collection*.		
var	Declaració de l'objecte amb el qual s'iterarà sobre els elements i que es farà servir a les columnes.		
rowStyleClasses	Llistat d'atributs 'class' separats per comes que serviran per definir l'estil de les files de la taula.		
border	Atribut 'border' de l'etiqueta <table> generada.		

cellpadding	Atribut 'cellpadding' de l'etiqueta <table> generada.
cellspacing	Atribut 'cellspacing' de l'etiqueta <table> generada.
width	Atribut 'width' de l'etiqueta <table> generada.
emptyMessage	Missatge que es mostrarà a la taula si no hi ha elements.

Classe	**ColumnTag**	**Classe pare**	TagSupport
Ús	`<jwl:column value="#{persona.nom}" ... />`		
Descripció	Etiqueta que renderitzarà les cel·les d'una columna a una taula de dades amb les dades dels elements de l'etiqueta superior *DataTableTag*.		
Atribut	**Descripció de l'atribut**		
value	Valor de la columna. Vendrà donat per una expressió que agafarà les dades dels elements del *DataTableTag*.		
title	Títol de la columna que es mostrarà a la capçalera de la taula.		
headerStyle	Atribut 'style' de l'etiqueta <th> de la capçalera de la columna.		
headerStyleClass	Atribut 'class' de l'etiqueta <th> de la capçalera de la columna.		
rowStyle	Atribut 'style' de l'etiqueta <td> de les cel·les de la columna.		
rowStyleClass	Atribut 'class' de l'etiqueta <td> de les cel·les de la columna.		

Classe	**ColumnActionTag**	**Classe pare**	TagSupport

Ús	`<jwl:columnAction action="#{personesController.editar}" actionText="Editar persona" ...>`
	`...`
	`</jwl:columnAction>`		

Descripció	Etiqueta que renderitzarà les cel·les d'una columna a una taula de dades amb un botó o un enllaç a una acció a efectuar sobre l'element que representa aquella fila.		
Atribut	**Descripció de l'atribut**		
action	Expressió que representa l'acció d'un controlador a executar.		
actionText	Text a mostrar a l'enllaç o a la imatge del botó.		
actionImg	Imatge que representa el botó on s'haurà de clicar per executar l'acció.		
title	Títol de la columna que es mostrarà a la capçalera de la taula.		
headerStyle	Atribut 'style' de l'etiqueta <th> de la capçalera de la columna.		
headerStyleClass	Atribut 'class' de l'etiqueta <th> de la capçalera de la columna.		
rowStyle	Atribut 'style' de l'etiqueta <td> de les cel·les de la columna.		
rowStyleClass	Atribut 'class' de l'etiqueta <td> de les cel·les de la columna.		

Classe	**LocaleSelectTag**	**Classe pare**	AbstractHtmlTag
Ús	`<jwl:localeSelect type="SELECT|LINKS" ... />`		

Descripció	Mostra l'opció de canviar d'idioma entre els disponibles a l'aplicació, indicats a l'arxiu de configuració, mitjançant un desplegable o enllaços.
Atribut	**Descripció de l'atribut**
type	Indicarà quin tipus de selecció es mostrarà: desplegable o enllaços.

Classe	**CommandButtonTag**	**Classe pare**	AbstractHtmlTag

Ús	`<jwl:commandButton action="#{personesBean.crear}" ...>`
	`...`
	`</jwl:commandButton>`		

Descripció	Renderitzarà un botó per enviar el formulari que el conté per executar una acció d'un controlador. Al clicar, es validarà el formulari (si hi ha elements amb validacions) i s'executarà, començant així una petició Jewel. Si l'acció conté paràmetres GET, a més dels paràmetres del formulari on està inclòs, es poden definir mitjançant etiquetes ParamTag a l'interior.		
Atribut	**Descripció de l'atribut**		
action	Expressió que representa l'acció d'un controlador a executar.		
label	Text o valor que es mostra al botó.		
validateForm	Atribut per indicar si s'ha de validar el formulari o no.		
tokenCheck	Atribut per indicar si s'ha de comprovar la mostra de reenviament de formularis (sempre i quan el filtre *TokenFilter* estigui actiu).		
name	Atribut 'name' de l'etiqueta <input> generada.		
tabindex	Atribut 'tabindex' de l'etiqueta <input> generada.		

Classe	**CommandLinkTag**	**Classe pare**	AbstractHtmlTag

Ús	`<jwl:commandLink action="#{personesBean.crear}" ...>`
	`...`
	`</jwl:commandLink>`		

Descripció	Serveix per executar una acció de Jewel mitjançant un enllaç. Aquesta etiqueta no executa cap formulari, pel que els paràmetres només poden ser de tipus GET i es poden definir mitjançant etiquetes ParamTag a l'interior.		
Atribut	**Descripció de l'atribut**		
action	Expressió que representa l'acció d'un controlador a executar.		
label	Text o valor que es mostra a l'enllaç.		
target	Atribut 'target' de l'etiqueta <a> generada.		
rel	Atribut 'rel' de l'etiqueta <a> generada.		
tabindex	Atribut 'tabindex' de l'etiqueta <a> generada.		

Classe	**UrlTag**	**Classe pare**	TagSupport
Ús	`<jwl:url action="#{personesBean.getOpcions}">` `...`		

	`</jwl:url>`
Descripció	Serveix per generar una URL cap a una acció. Si l'acció conté paràmetres GET, es poden definir mitjançant etiquetes ParamTag a l'interior.
Atribut	**Descripció de l'atribut**
action	Expressió que representa l'acció d'un controlador a executar. En cas de no especificar-se, la URL enllaçarà a la pàgina de benvinguda de l'aplicació.
ajax	Atribut per indicar que la URL cridarà a una acció Ajax. Si no es marca com a *true* a una petició Ajax, es generarà una nova mostra per validar el reenviament de formularis, pel que si s'envia un formulari després de la petició Ajax sense recarregar la pàgina donarà error per reenviament.

Classe	**ParamTag**	**Classe pare**	TagSupport
Ús	`<jwl:param name="grup" value="#{persona.grup} />`		
Descripció	Etiqueta utilitzada dins un *CommandLinkTag*, un *CommandButtonTag*, un *ColumnActionTag* o un *UrlTag* per indicar els paràmetres de URL d'una acció, és a dir, aquells que formaran part de la URL i s'utilitzaran com arguments del mètode que representa l'acció. Aquests paràmetres, com es veurà al capítol 12.3, s'han de declarar al mètode de l'acció que es vol executar.		
Atribut	**Descripció de l'atribut**		
name	Nom del paràmetre que s'utilitzarà per posar el valor del paràmetre al lloc adequat de la URL formada.		
value	Expressió que representa el valor que es posarà a la URL.		

Classe	**LabelTag**	**Classe pare**	AbstractHtmlTag
Ús	`<jwl:label forId="nom" text="#{msg['nom']} ... />`		
Descripció	Etiqueta utilitzada per dibuixar un element HTML de tipus <label> associat a un camp d'entrada de dades d'un formulari.		
Atribut	**Descripció de l'atribut**		
forId	Text identificador del camp relacionat amb l'etiqueta. Es correspon amb l'atribut 'for' de l'etiqueta <label> generada.		
text	Expressió que representa el valor del text contingut a l'etiqueta <label> generada.		

Classe	**OutputTextTag**	**Classe pare**	AbstractHtmlTag
Ús	`<jwl:outputText text="#{msg['titol_pagina']}" ... />`		
Descripció	Etiqueta utilitzada per escriure el valor d'una expressió Jewel, podent esser un missatge internacionalitzat o una propietat d'un objecte del context.		
Atribut	**Descripció de l'atribut**		
text	Expressió que representa el valor del text contingut a l'etiqueta		

	generada.
onlyText	Indica que el text indicat a l'atribut *text* no s'ha de posar dins una etiqueta SPAN. Si és així, s'ignorarà la resta d'atributs HTML ja que no hi haurà etiqueta HTML.

Classe	**OutputDateTag**	**Classe pare**	AbstractHtmlTag
Ús	`<jwl:outputDate date="#{projecte.data }" pattern="dd/MM/yyyy" ... />`		
Descripció	Etiqueta utilitzada per escriure una data en un format concret, podent esser una expressió Jewel o un objecte java.util.Date.		
Atribut	**Descripció de l'atribut**		
date	Expressió que representa la data a escriure.		
pattern	Patró o format en que s'escriurà la data.		
onlyText	Indica que la data no s'ha de posar dins una etiqueta SPAN. Si és així, s'ignorarà la resta d'atributs ja que no hi haurà etiqueta HTML.		

En quan als camps d'entrada de dades pels formularis, comparteixen una sèrie d'atributs que hereten de la classe *AbstractInputTag*, que a la vegada hereta de la classe *AbstractHtmlTag*:

Atributs d'AbstractInputTag	**Descripció de l'atribut**
value	Expressió Jewel que representa el valor del camp.
name	Atribut 'name' de l'etiqueta HTML generada. S'utilitzarà a l'hora de mapar el camp amb els paràmetres de l'acció.
required	Indica si el camp és obligatori al formulari que el conté, de manera que quan s'enviï es validi que té valor.
requiredMessage	Expressió amb el text d'informació que es mostrarà si el camp no té valor a l'hora d'enviar el formulari.
disabled	Serveix per indicar si el camp ha d'estar inhabilitat. Es correspon amb l'atribut 'disabled' de l'etiqueta HTML generada.
readonly	Serveix per indicar si el camp és de només lectura. Es correspon amb l'atribut 'readonly' de l'etiqueta HTML generada.
size	Atribut 'size' de l'etiqueta HTML generada.
tabindex	Atribut 'tabindex' de l'etiqueta HTML generada.

Classe	**InputTextTag**	**Classe pare**	AbstractInputTag
Ús	`<jwl:inputText name="persona.nom" value="#{persona.nom} ... />`		
Descripció	Representa un camp d'entrada de text. Es correspondrà amb un paràmetre de tipus *String*, *Integer*, *Float*, *Long* o *Double* en funció del paràmetre a l'acció.		

Atribut	Descripció de l'atribut
maxlength	Atribut 'maxlength' de l'etiqueta <input> generada.

Classe	InputHiddenTag	Classe pare	AbstractInputTag
Ús	`<jwl:inputHidden name="persona.id" value="#{persona.id} ... />`		
Descripció	Representa un camp d'entrada de text de tipus ocult. Es correspondrà amb un paràmetre de tipus *String*, *Integer*, *Float*, *Long* o *Double* en funció del paràmetre a l'acció.		
Atribut	**Descripció de l'atribut**		

Classe	InputPasswordTag	Classe pare	AbstractInputTag
Ús	`<jwl:inputPassword name="persona.clau" value="#{persona.clau} ... />`		
Descripció	Representa un camp d'entrada de text de tipus clau. Es correspondrà amb un paràmetre de tipus *String*, *Integer*, *Float*, *Long* o *Double* en funció del paràmetre a l'acció.		
Atribut	**Descripció de l'atribut**		
maxlength	Atribut 'maxlength' de l'etiqueta <input> generada.		

Classe	TextareaTag	Classe pare	AbstractInputTag
Ús	`<jwl:inputText name="persona.comentari" value="#{persona. comentari } ... />`		
Descripció	Representa un camp d'entrada de text de tipus *textarea*. Es correspondrà amb un paràmetre de tipus *String*.		
Atribut	**Descripció de l'atribut**		
cols	Atribut 'cols' de l'etiqueta <textarea> generada.		
rows	Atribut 'rows' de l'etiqueta <textarea> generada.		

Classe	InputDateTag	Classe pare	AbstractInputTag
Ús	`<jwl:inputDate name="persona.dataNaixament" pattern="dd/mm/yy" value="#{persona.dataNaixament} ... />`		
Descripció	Representa un camp d'entrada de text de tipus *input* acompanyat d'un botó que servirà per mostrar un calendari on seleccionar la data. Es correspondrà amb un paràmetre de tipus *String* o *Date* en funció del paràmetre a l'acció. El calendari es generarà amb l'extensió *datepicker* del nucli de jQuery. En cas de no tenir jQuery a la vista es mostrarà un missatge d'alerta.		
Atribut	**Descripció de l'atribut**		
pattern	Patró en que es mostra la data, per defecte 'dd/mm/yy'. Ha de ser un		

	format acceptat per jQuery.
buttonImage	URL de la imatge que s'utilitzarà com a botó per mostrar el calendari. Si no té valor, el calendari es mostrarà al entrar al camp *input* de la data.

Classe	**SelectTag**	**Classe pare**	AbstractInputTag

Ús	`<jwl:select name="persona.idEmpresa" value="#{persona.empresa.id}" ...>`
	`...`
	`</jwl:select>`		

Descripció	Representa un camp d'opcions desplegable o multi opció, on les opcions vendran donades per les etiquetes *OptionTag* o *OptionsTag*. Es correspondrà amb un paràmetre de tipus *String*, *Integer*, *Float*, *Long* o *Double* en funció del paràmetre a l'acció.		
Atribut	**Descripció de l'atribut**		
emptyOption	Indica si s'ha de posar una opció sense valor al seleccionable.		
emptyOptionValue	Expressió que indica el valor que té l'opció buida. Per defecte és un valor buit.		
emptyOptionLabel	Expressió que indica el nom de l'opció buida. Per defecte és la cadena de text 'Sense valor'.		
multiple	Indica si es tracta d'un seleccionable de múltiple selecció.		

Classe	**SelectRadioTag**	**Classe pare**	AbstractInputTag

Ús	`<jwl:selectRadio name="persona.idEmpresa" value="#{persona.empresa.id}" ...>`
	`...`
	`</jwl:selectRadio>`		

Descripció	Representa un conjunt de camps seleccionables de tipus *radio* que vendran donats per les etiquetes *OptionTag* o *OptionsTag*. El seu valor es correspondrà amb un paràmetre de tipus *String*, *Integer*, *Float*, *Long* o *Double* en funció del paràmetre a l'acció.		
Atribut	**Descripció de l'atribut**		
emptyOption	Indica si s'ha de posar una opció de tipus *radio* sense valor.		
emptyOptionValue	Expressió que indica el valor que té l'opció buida. Per defecte és un valor buit.		
emptyOptionLabel	Expressió que indica el nom de l'opció buida. Per defecte és la cadena de text 'Sense valor'.		

Classe	**SelectCheckboxTag**	**Classe pare**	AbstractInputTag

Ús	`<jwl:selectCheckbox name="persona.idHabilitats" value="#{persona.habilitats }" ...>`
	`...`
	`</jwl: selectCheckbox>`		

Descripció	Representa un conjunt de camps seleccionables de tipus *checkbox* que vendran donats per les etiquetes *OptionTag* o *OptionsTag*. El seu valor es correspondrà amb un paràmetre de tipus *String*, *Integer*, *Float*, *Long* o		

	Double en funció del paràmetre a l'acció.
Atribut	**Descripció de l'atribut**
emptyOption	Indica si s'ha de posar una opció de tipus *checkbox* sense valor.
emptyOptionValue	Expressió que indica el valor que té l'opció buida. Per defecte és un valor buit.
emptyOptionLabel	Expressió que indica el nom de l'opció buida. Per defecte és la cadena de text 'Sense valor'.

Classe	**OptionsTag**	**Classe pare**	AbstractHtmlTag
Ús	`<jwl:options elements="#{empreses}" var="empresa" label="#{empresa.nom}" value="#{empresa.id}" ... />`		
Descripció	Representa un conjunt d'opcions per una etiqueta *SelectTag*, *SelectRadioTag* o *SelectCheckboxTag*. En funció de l'etiqueta pare es representarà amb camps `<option>`, camps `<input type="radio">` o camps `<input type="checkbox">`.		
Atribut	**Descripció de l'atribut**		
elements	Elements continguts en un objecte *java.util.Collection*.		
var	Declaració de l'objecte amb el qual s'iterarà sobre els elements i que es podrà fer servir als atributs *label* i *value*.		
label	Expressió que representa el nom o etiqueta visible de l'opció. Es podrà utilitzar l'atribut *var* per accedir a una propietat de l'element que representa.		
value	Expressió que representa el valor de l'opció. Es podrà utilitzar l'atribut *var* per accedir a una propietat de l'element que representa.		

Classe	**OptionTag**	**Classe pare**	AbstractHtmlTag
Ús	`<jwl:option label="#{empresa.nom}" value="#{empresa.id}" ... />`		
Descripció	Representa una opció per una etiqueta *SelectTag*, *SelectRadioTag* o *SelectCheckboxTag*. En funció de l'etiqueta pare es representarà amb camps `<option>`, camps `<input type="radio">` o camps `<input type="checkbox">`.		
Atribut	**Descripció de l'atribut**		
label	Expressió que representa el nom o etiqueta visible de l'opció.		
value	Expressió que representa el valor de l'opció.		
selected	Indica si és l'opció seleccionada per defecte. Es correspon amb l'atribut 'checked' si l'etiqueta pare és un *SelectRadioTag* o *SelectCheckboxTag*, o amb l'atribut 'selected' si l'etiqueta pare és un *SelectTag*.		
disabled	Indica si l'opció es troba inhabilitada. Es correspon amb l'atribut 'disabled' de l'etiqueta HTML generada.		

Com s'ha pogut apreciar, no hi han atributs per representar els esdeveniments Javascript. Això és degut a que es promou la separació entre el codi HTML i el codi Javascript, de manera que per incorporar esdeveniments als elements d'una pàgina es faci mitjançant jQuery tal i com s'explica a la seva API[26]. D'aquesta manera s'aconsegueix evitar que el codi Javascript es mescli amb el codi HTML.

11.3.3. Anotacions

Les anotacions són una peça fonamental del framework ja que permetran indicar quines classes actuaran de controladors, quins dels seus mètodes representaran les accions, quines classes actuaran de filtres, quins dels seus mètodes es cridaran al processar el filtre i aspectes de seguretat a les accions.

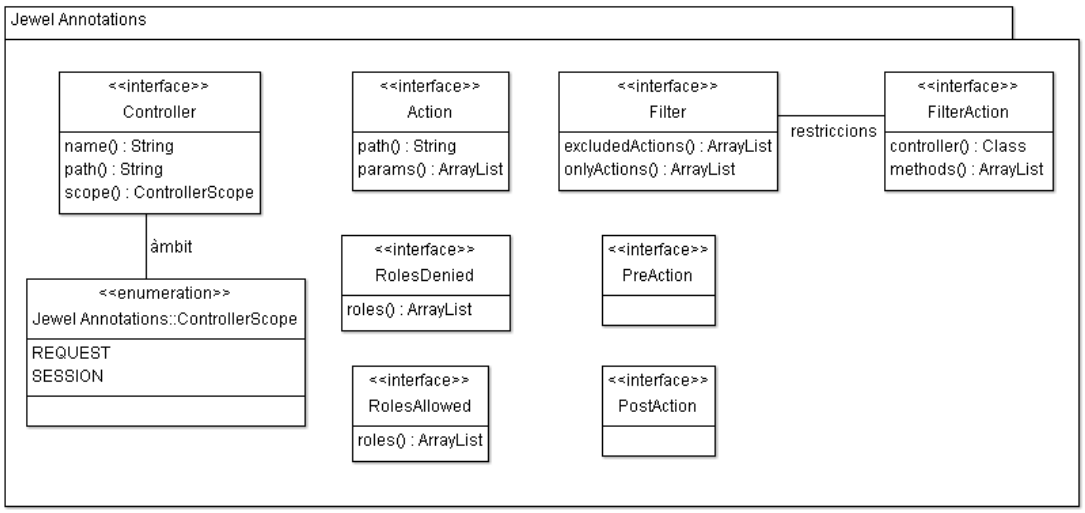


Figura 17. Diagrama de classes: Anotacions del framework

@Controller

S'utilitza per indicar que una classe és un controlador d'accions. Cal definir els següents paràmetres:

- *name*: Nom identificador del controlador que s'utilitzarà a les vistes per referenciar-lo.
- *path*: Patró URL amb el que es determinarà que una acció pertany a aquest controlador.
- *scope*: Àmbit del controlador, és a dir, si només romandrà viu durant la petició o si conté variables que han de guardar-se en sessió.

ControllerScope

Enumeració dels àmbits possibles d'un controlador:

- *Request*: Àmbit de petició, és a dir, el controlador es crearà al fer-se una petició però no serà accessible amb les mateixes dades a la següent petició.
- *Session*: Àmbit de sessió de l'usuari, és a dir, el controlador romandrà viu a la sessió de l'usuari i les variables seran accessibles, sempre que siguin públiques o tinguin un mètode d'accés, a les vistes mentre duri la sessió.

[26] http://api.jquery.com/category/events/

@Action

S'utilitza per indicar que un mètode d'un controlador representa una acció. Cal indicar els següents paràmetres:

- *path*: Patró URL que, junt amb el patró del controlador al que pertany el mètode, determinaran el patró URL que identifica l'acció. Pot contenir declaració de paràmetres, tal i com es veurà al capítol 12.3.1, que s'hauran de declarar com arguments del mètode.
- *params*: Llistat de noms que identificaran els paràmetres que s'enviaran a l'acció per tal d'associar-los als arguments del mètode.

@RolesAllowed

Anotació utilitzada a un mètode d'acció per definir els únics rols que el poden executar. Cal definir el paràmetre *roles* amb el llistat de rols permesos.

@RolesDenied

Anotació a nivell de mètode per indicar quins rols no tenen permís per executar l'acció. S'ha d'indicar el llistat de rols sense permís al paràmetre *roles*.

@Filter

Anotació que identifica una classe com a filtre Jewel. Es poden definir els següents paràmetres:

- *excludedActions*: S'utilitza per indicar a quines accions no s'ha d'executar el filtre mitjançant anotacions @FilterAction.
- *onlyActions*: Indica quines són les úniques accions a les que s'aplicarà el filtre mitjançant anotacions @FilterAction.

@FilterAction

Anotació per ser inclosa als atributs *excludedAction* i *onlyActions* que indicarà el controlador i els mètodes que estan limitats per l'atribut. És a dir, quines accions permeten o no permeten l'execució del filtre. S'han de definir els paràmetres següents:

- *controller*: Classe que representa el controlador.
- *methods*: Nom dels mètodes que representen les accions. En cas de no indicar mètodes s'inclouran a la restricció tots els mètodes del controlador indicat a l'atribut *controller*.

@PreAction

Anotació per designar que un mètode d'un filtre s'executarà abans d'invocar l'acció.

@PostAction

Anotació a nivell de mètode per indicar quin és el mètode d'un filtre a executar després d'invocar una acció. Es pot combinar amb @PreAction per indicar que un mètode d'un filtre s'executarà abans i després de l'acció.

11.4. Diagrames de seqüència

En aquest apartat es descriuen, mitjançant diagrames de seqüència, els principals processos que durà a terme el framework per implementar la funcionalitat, quines classes hi intervenen i com es relacionen.

11.4.1. Procés d'inicialització del framework

El primer que fa el framework és inicialitzar-se per configurar el controlador d'aplicacions i tots els gestors de Jewel. Això es fa quan s'inicialitza el *JewelServlet* amb el mètode `init()` que hereta de la classe *javax.servlet.http.HttpServlet* i s'invoca quan es desplega l'aplicació, si així s'ha indicat a l'arxiu de configuració web.xml.

Al inicialitzar el framework es segueix el diagrama descrit a la Figura 18 compost per els següents passos:

1. El servidor d'aplicacions invoca la inicialització de *JewelServlet* al desplegar l'aplicació.
2. *JewelServlet* obté una instància de *ApplicationController*. Com que és la primera vegada, es crearà la única instancia que romandrà viva a l'aplicació i contindrà tota la configuració del framework.
3. *JewelServlet* invoca la configuració de *ApplicationController* i aquest inicialitza la configuració de l'aplicació i els distints gestors:
   3.1. Inicialitza la configuració principal a partir de l'arxiu jewel-config.xml i l'arxiu web.xml.
   3.2. Inicialitza la configuració de la internacionalització a partir de l'arxiu jewel-config.xml, és a dir: quins idiomes estaran disponibles, quin és l'idioma per defecte, quina és la ruta dels arxius de propietats amb els missatges internacionalitzats i el nom dels paràmetres que s'utilitzaran per accedir als missatges i per canviar d'idioma.
   3.3. Inicialitza el gestor de filtres, que s'encarrega de:
      a) Analitzar l'arxiu de configuració de Jewel en cerca de la declaració dels filtres creats pel programador i la configuració dels filtres que proporciona Jewel.
      b) Escanejar les classes de l'aplicació en cerca dels filtres creats pel programador.
      c) Preparar els llistats amb els mètodes de cada filtre a executar abans i després de les accions.
   3.4. Inicialitza el gestor d'accions que cerca totes les accions programades per l'usuari i prepara un mapa amb la relació: patró URL – acció.
   3.5. Inicialitza el gestor de vista que s'encarregarà d'instanciar les classes *ResultDispatcher* per presentar les vistes.

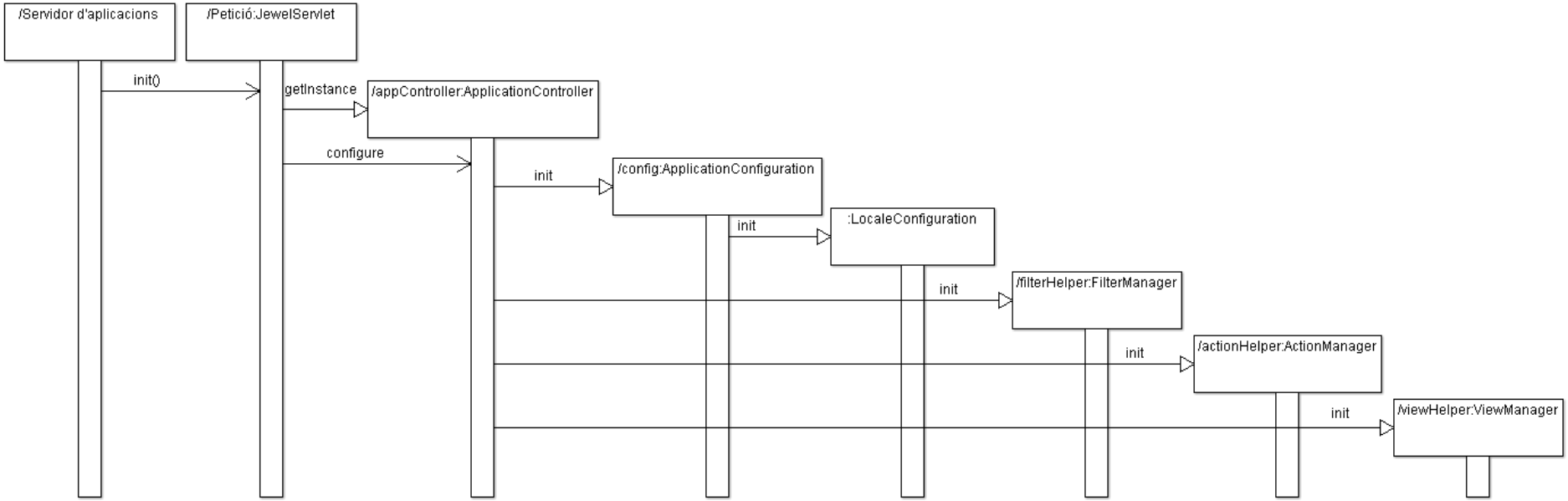


Figura 18. Diagrama de seqüència: Inicialització de Jewel

11.4.2. Procés de petició Jewel

Quan es rep qualsevol petició mapada al Servlet *JewelServlet*, aquest la processa mitjançant el mètode `service(request, response)` que s'encarrega de crear l'objecte de context i cridar el controlador d'aplicació per executar l'acció i presentar el resultat.

Es segueix el diagrama descrit a la Figura 19, que consta dels següents passos:

1. L'usuari llença una petició a Jewel a través de l'aplicació J2EE.
2. *JewelServlet* rep la petició i instancia el controlador d'aplicació *ApplicationController*.
3. *JewelServlet* crea l'objecte de context a partir dels objectes *HttpRequest* i *HttpResponse* que rep.
4. *JewelServlet* crida a *ApplicationController* per executar una acció passant-li l'objecte de context. El controlador d'aplicació comença el procés d'execució d'una acció:
   4.1. Crida al gestor d'accions, *ActionManager*, per trobar l'acció sol·licitada a la petició a partir de la URL.
   4.2. Utilitza el gestor de filtres, *FilterManager*, per executar els filtres pre-acció, sempre i quan al filtre no s'exclogui l'acció.
   4.3. Instancia la classe controladora que conté l'acció i crida el mètode que representa l'acció amb els paràmetres de l'objecte de context obtinguts a la petició. Aquest mètode retorna un objecte *ModelAndView* que conté el model que s'utilitzarà a la vista, el nom lògic de la vista i el tipus de vista a presentar.
   4.4. Utilitza el gestor de filtres, *FilterManager*, per executar els filtres post-acció si no estan exclosos per l'acció executada.
5. *JewelServlet* rep l'objecte *ModelAndView* i, si no és buit, torna a cridar el controlador d'aplicació perquè s'encarregui de presentar la vista. Si el resultat de l'acció és buit, es tornarà a presentar la vista anterior. El controlador d'aplicació comença el procés de presentar la vista:
   5.1. Utilitza el gestor de vistes, *ViewManager*, per trobar la implementació de *RequestDispatcher* que s'encarregarà de presentar la vista en funció del tipus indicat al *ModelAndView*.
   5.2. Instancia la classe que implementa *RequestDispatcher* i executa el mètode *dispatch*. Aquest mètode s'encarrega d'adequar el model al tipus de vista a presentar i redirigir a la vista, que pot ser un JSP, una plantilla Tiles, una resposta JSON, o una altra definida pel programador.

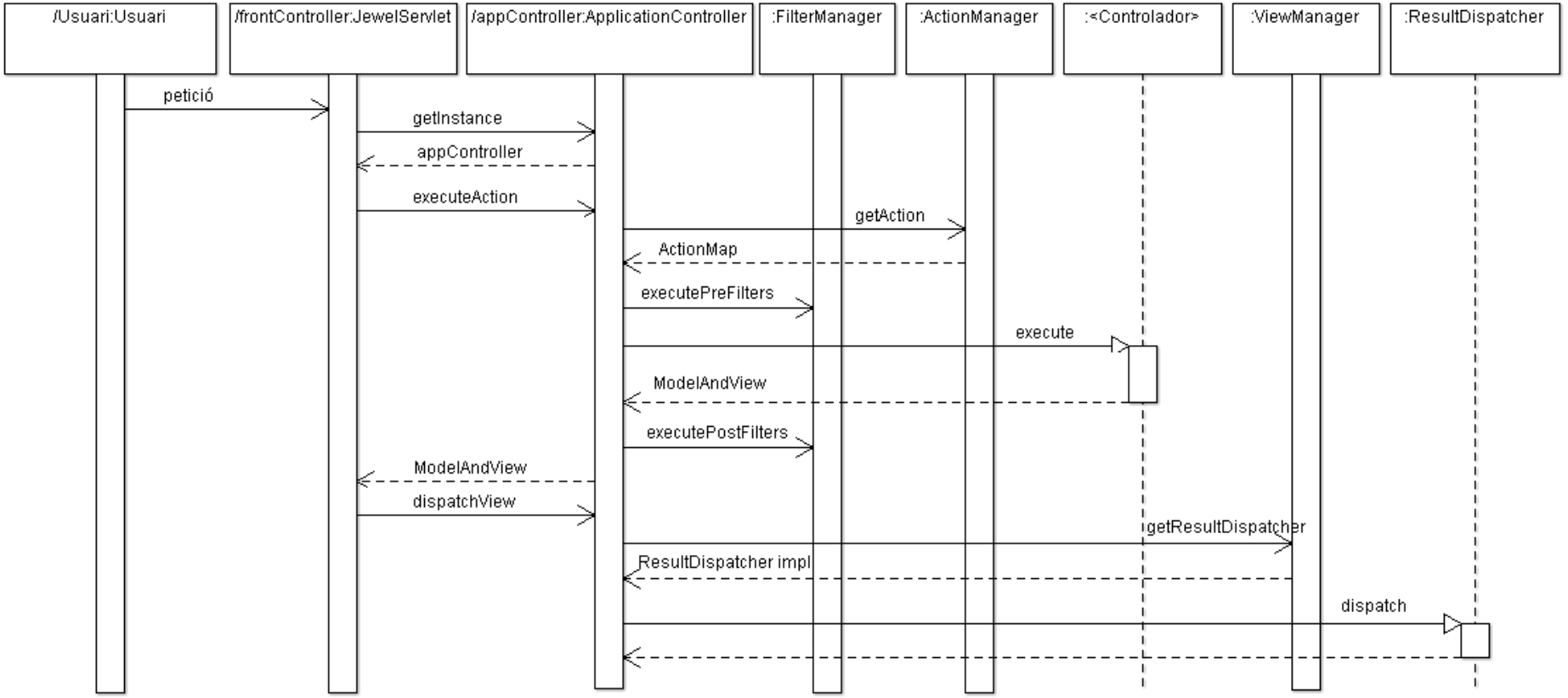


Figura 19. Diagrama de seqüència: Procés de petició Jewel

11.4.3. Procés d'execució dels filtres

A l'hora de processar una petició s'han d'executar els filtres abans i després de l'acció, tal i com s'ha vist al punt anterior. L'ordre d'execució dels filtres ve determinat per la declaració d'aquests a l'arxiu de configuració i és molt important, ja que un filtre pot dependre dels resultats d'un altre o influir-hi.

Així doncs, el procés d'execució dels filtres segueix el diagrama de seqüència indicat a la Figura 20 i es descriu amb els següents passos:

1. El controlador d'aplicació crida al gestor de filtres, *FilterManager*, perquè executi els filtres pre-acció abans d'executar l'acció. El gestor de filtres itera damunt el llistat de filtres i els invoca un a un segons l'ordre següent:
    a. Primer executa els filtres de Jewel: *TokenFilter*, per comprovar si s'està reenviant el formulari i en casi afirmatiu no seguir processant la petició i retornar l'error. Posteriorment s'executa el *LocaleFilter*.
    b. Després s'executen els filtres definits pel programador, tenint dues opcions per determinar l'ordre:
        i. Si el programador declara els filtres a l'arxiu de configuració *jewel-config.xml* s'executaran en l'ordre d'aparició en aquest.
        ii. Si no ho declara, s'executaran en l'ordre que s'hagin trobat les classes al inicialitzar-se el framework.
2. En haver-se executat els filtres, el controlador d'aplicació invocarà l'acció.
3. El controlador d'aplicació crida al gestor de filtres perquè executi els filtres post-acció després d'haver executat l'acció. El gestor de filtres, *FilterManager*, itera damunt el llistat de filtres post-acció i els invoca un a un segons l'ordre invers a l'execució dels filtres pre-acció:
    a. Primer s'executen els filtres definits pel programador en ordre invers al que s'han executat abans de l'acció.
    b. Després s'executen els filtres de Jewel: *LocaleFilter* i *TokenFilter*.

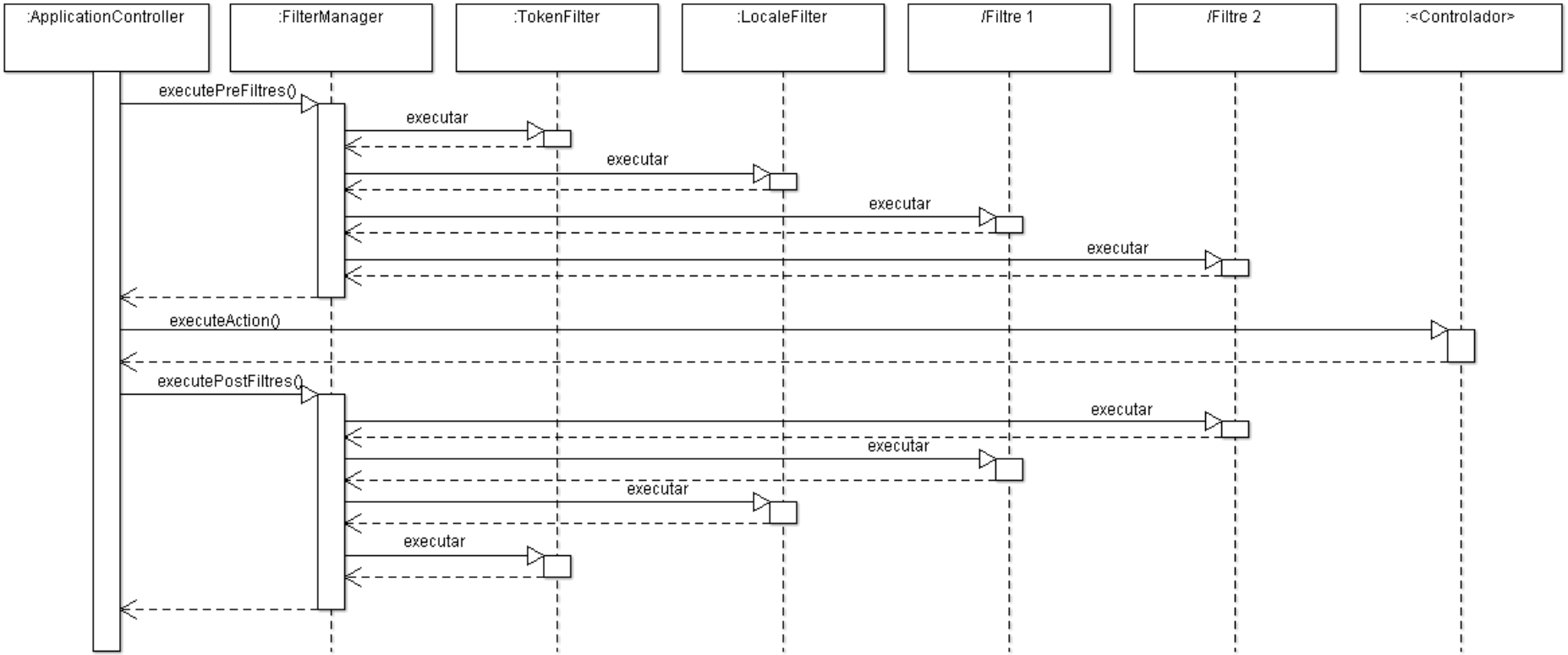


Figura 20. Diagrama de seqüència: Execució dels filtres

12. Utilització de Jewel

En aquest darrer apartat del disseny de Jewel es profunditzarà sobre com ha de desenvolupar una aplicació el programador: requeriments, dependències, configuració del framework i implementació dels components d'aquest. Es pretén enumerar els passos que ha de seguir el programador formant una guia de desenvolupament amb Jewel.

12.1. Requeriments

El primer que necessita Jewel, com a framework per aplicacions J2EE que és, és un servidor d'aplicacions J2EE amb JEE6 i Java 1.6 o superior. Alguns dels servidors d'aplicacions que es testejaran durant el desenvolupament de Jewel i seran suportats són els següents, ja que són dels més utilitzats actualment:

- Apache Tomcat 7
- Jboss AS 6.1 Final
- Jboss AS 7 Final
- Glassfish 3.0.1

A més, Jewel depèn de certes llibreries per implementar la seva funcionalitat. Aquestes llibreries, que ha d'aportat l'aplicació que utilitzi el framework, són les que s'indiquen al següent arbre de dependències:

- Apache Commons-Digester 2.1. Utilitzat per llegir l'arxiu XML de configuració.
    - Apache Commons-BeanUtils 1.8.3
- Apache Commons-BeanUtils 1.8.3. S'utilitza en conjunt amb reflection per accedir als atributs de les classes definides pel programador i associar-hi els paràmetres de petició en temps d'execució.
    - Apache Commons-Logging 1.1.1
    - Apache Commons-Collections 3.2.1
- SLF4j 1.6.4. Utilitzat per mostrar els missatges de depuració, d'error i d'informació independentment del sistema de depuració o *logging* utilitzat per l'usuari. Per tant, Jewel no n'imposa cap i es pot utilitzar Log4j, commons-logging, etc.
- Gson 2.0. Utilitzat per convertir els objectes Java del model a objectes JSON pels resultats de les accions que siguin d'aquest tipus. En cas de no utilitzar peticions JSON no importa incloure aquesta llibreria.
- Apache Tiles 2.2.2. Utilitzat per utilitzar plantilles de vistes. En cas de no utilitzar vistes Tiles no importa incloure aquesta llibreria.
    - Apache Commons-Digester 2.1
    - Simple Logging Facade for Java (SLF4J) 1.x

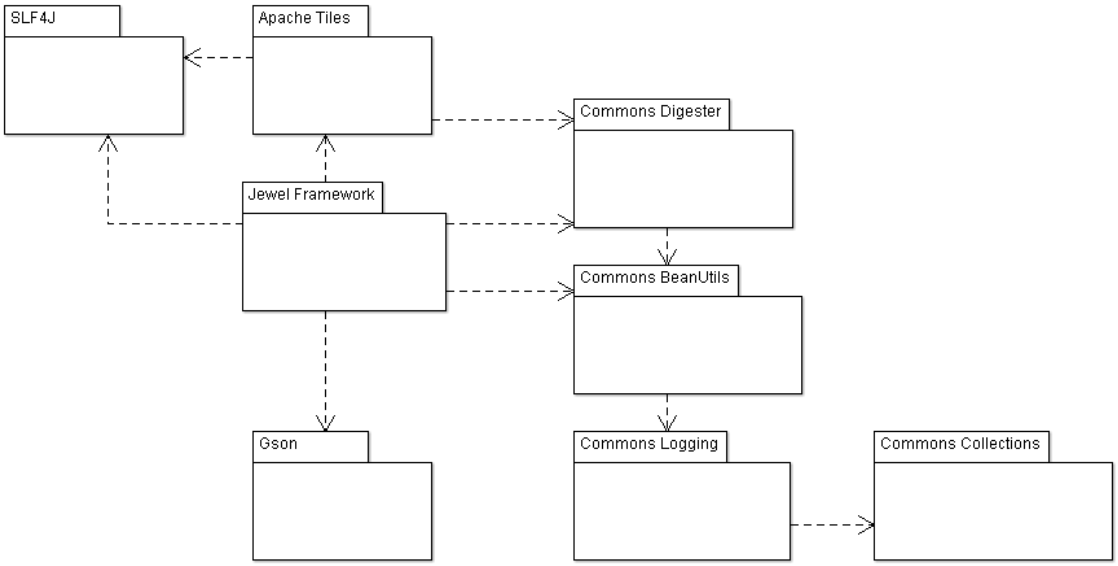


Figura 21. Diagrama de dependències de Jewel

12.2. Configuració de Jewel

El primer que cal fer per utilitzar Jewel a una aplicació J2EE és afegir el servlet principal, *org.jewel.core.JewelServlet*, a l'arxiu web.xml i mapar-lo a un patró URL. A més, caldrà indicar on es troba la definició de la llibreria d'etiquetes que proporciona Jewel dins l'etiqueta `<jsp-config>`.

Al Codi d'exemple 11 s'observa que totes les peticions que es facin a la URL amb patró `*.jwl` les rebrà i processarà JewelServlet. El framework accepta qualsevol patró al mapa de JewelServlet, pel que es podria haver utilitzat un altre patró com `/jewel/*`.

A més, es configura per a que s'inicialitzi al desplegar l'aplicació mitjançant l'etiqueta `<load-on-startup>` amb valor 1. Això permet que el framework quedi configurat abans de processar-hi qualsevol petició, pel que si hi ha algun error de configuració es detectarà abans que cap usuari accedeixi a l'aplicació i aquesta ja no es desplegarà.

El darrer paràmetre important a l'hora de declarar el Servlet és la ruta a l'arxiu de configuració propi de Jewel mitjançant el paràmetre d'inici *org.jewel.CONFIG_FILE*. Si no s'indica aquest paràmetre, Jewel cercarà l'arxiu a la ruta per defecte /WEB-INF/jewel-config.xml i en cas de no existir donarà un error al desplegar l'aplicació.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
...
  <servlet>
     <description>Jewel Servlet</description>
     <servlet-name>Jewel Servlet</servlet-name>
     <servlet-class>org.jewel.core.JewelServlet</servlet-class>
     <init-param>
        <description>Arxiu de configuració del framework
             </description>
        <param-name>org.jewel.CONFIG_FILE</param-name>
        <param-value>/WEB-INF/jewel-config.xml</param-value>
     </init-param>
     <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
     <servlet-name>Jewel Servlet</servlet-name>
     <url-pattern>*.jwl</url-pattern>
  </servlet-mapping>
  <jsp-config>
     <taglib>
        <taglib-uri>http://jewel.org/taglib</taglib-uri>
        <taglib-location>/WEB-INF/jewel.tld</taglib-location>
     </taglib>
  </jsp-config>
</web-app>
```

Codi d'exemple 11. Arxiu de configuració web.xml

L'arxiu de configuració de Jewel ha de seguir un esquema XSD, que es proporcionarà amb el framework. Aquest arxiu serà similar al del Codi d'exemple 12, on es configuren els següents elements:

- Filtres de Jewel
  En aquest apartat s'indica si els filtres que proporciona Jewel estaran actius o inactius.
- Filtres del programador
  Aquesta etiqueta presenta dues opcions:
  a) Que Jewel auto escanegi les classes de l'aplicació en cerca dels filtres: Etiqueta `<autoscan>` amb valor *true*.
  b) Indicar els filtres desenvolupats pel programador i l'ordre d'aquests: Etiqueta `<autoscan>` a *false* i etiquetes *<class>* amb les classes que implementen filtres.

  A l'exemple, s'indiquen els filtres *LogFilter* i *DatabaseLogFilter*, que al processar una petició s'executaran en aquest ordre abans d'executar l'acció i en ordre invers després d'haver-la executat.

- Configuració de la internacionalització
  S'indicarà quins idiomes té disponibles l'aplicació, quin és l'idioma per defecte i quina és la ruta base de l'arxiu de propietats que conté els missatges internacionalitzats. A l'exemple, el framework cercarà els missatges internacionalitzats als arxius "/WEB-

INF/messages_ca.properties", "/WEB-INF/messages_es.properties" i "/WEB-INF/messages_en.properties".

A més, s'indica quin és el nom del paràmetre utilitzat per canviar l'idioma de l'aplicació i el nom de l'atribut que s'utilitzarà a les vistes per accedir als missatges internacionalitzats.

- Pàgina d'error

  Serveix per indicar quina és la pàgina a mostrar quan es produeixi un error no controlat pel programador. La pàgina d'error pot ser un arxiu JSP o una plantilla Tiles, podent-ho declarar a l'etiqueta `<type>`, que pot contenir els valors "JSP" o "TILES". L'etiqueta `<location>` indica el nom o la ruta de la vista.

- Mode depuració

  Si està activat, es mostrarà informació de depuració al log. En entorns de producció es recomana desactivar-ho. A més, si està activat i es produeix un error es mostrarà la traça de l'excepció en una pàgina personalitzada de Jewel, ignorant la pàgina d'error de l'usuari.

```
<?xml version="1.0" encoding="UTF-8"?>
<jewel-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="jewel-config.xsd">
  <debug-mode>false</debug-mode>

  <jewel-filters>
    <token-filter>true</token-filter>
    <locale-filter>true</locale-filter>
  </jewel-filters>

  <custom-filters>
    <autoscan>false</autoscan>
    <class>org.jewel.example.filters.LogFilter</class>
    <class>org.jewel.example.filters.DatabseLogFilter</class>
  </custom-filters>

  <locale-config>
    <messages-file>/messages</messages-file>
    <default-language>ca</default-language>
    <language>es</language>
    <language>en</language>
    <change-locale-parameter>lang</change-locale-parameter>
    <resource-bundle-variable>msgs</resource-bundle-variable>
  </locale-config>

  <error-page>
    <location>error-page</location>
    <type>TILES</type>
  </error-page>
</jewel-config>
```

Codi d'exemple 12. Arxiu de configuració jewel-config.xml

A més, si l'aplicació utilitza la llibreria Apache Tiles per presentar vistes compostes s'haurà d'afegir la configuració d'aquesta a l'arxiu web.xml a més de crear un arxiu on es declarin les vistes a utilitzar. [27]

12.3. Implementació dels components de Jewel

En aquest apartat es veurà el que potser sigui el més important pel desenvolupador: com es configuren i s'implementen els components del framework. Es descriurà com s'implementa un controlador i les seves accions, per tal que siguin accessibles; com es crea un filtre, amb els mètodes a executar abans i després d'una acció; i com es programen els diferents tipus de vista on presentar els resultats d'executar una acció.

12.3.1. Controladors i accions

Els controladors són les classes Java on s'agruparan les accions que tenen alguna cosa en comú. Es tracta de classes simples declarades amb l'anotació @Controller que contenen una sèrie de mètodes que representen les accions, anotats com @Action . L'exemple del Codi d'exemple 13 servirà per explicar el seu ús.

La declaració del controlador amb l'anotació @Controller ens proporciona la següent informació:

- El nom del controlador amb el qual ens hi referenciarem a la vista és "personesController".
- El patró URL que determinarà que una acció pertany a aquest controlador és "/persones/*"
- El controlador té àmbit de petició, pel que no pot declarar atributs ja que només romandrien vius durant una petició, essent inaccessibles a la següent petició.

El controlador conté cinc accions, cadascuna amb característiques diferents per tal de cobrir tots els aspectes disponibles:

Acció simple sense paràmetres i vista composta

- El nom de l'acció és "principal" i es relaciona amb el mètode *principal()*.
- És una acció sense paràmetres accessible amb la URL "/persones/principal".
- L'acció retorna una vista de tipus Apache Tiles que es correspon amb el nom lògic "index".

Acció amb paràmetres de formulari i vista JSP

- El nom de l'acció és "llistat" i es relaciona al mètode *llistat()*.
- És una acció amb un paràmetre que representa un conjunt de noms i s'accedeix mitjançant la URL "/persones/llistat".
- El paràmetre declarat conté un conjunt d'entrades de text amb el nom "noms". Cal tenir en compte que el nom que se li dona a l'argument no importa, el nom que s'ha d'utilitzar a l'atribut *name* de les etiquetes dels camps dels formularis és el que es declara a l'atribut *params*.

[27] La configuració d'Apache Tiles queda fora de l'àmbit del projecte. Es pot consultar a la pàgina oficial del framework: http://tiles.apache.org/2.2/framework/index.html

- Aquesta acció retorna un model amb un llistat d'objectes de tipus Persona que seran accessibles des de la vista a través de l'atribut "persones".
- Es redirigirà a un JSP que es troba a la ruta "/persones/llistat_persones_nom.jsp" de l'aplicació.
- A més, en cas de no retornar un conjunt de persones, s'adjunta un missatge que es podrà mostrar a la vista allà on s'utilitzi l'etiqueta `<jwl:messages>`.

Acció amb paràmetres de URL i paràmetres de formulari mapats a un JavaBean

- El nom de l'acció és "llistatGrup" i es relaciona al mètode *llistatGrup()*.
- És una acció amb dos paràmetres, un de tipus URL i l'altre de formulari, i s'accedeix amb la URL "/persones/llistat/#{grup}".
- El primer paràmetre es troba a la mateixa URL. A la vista s'haurà indicat mitjançant l'etiqueta `<jwl:param name="grup" value="1"/>` dins d'un botó d'acció `<jwl:commandButton>`, un enllaç `<jwl:commandLink>`, una etiqueta `<jwl:url>` o una columna d'acció d'un `<jwl:dataTable>`.
- El segon paràmetre és un JavaBean creat per l'usuari al que s'accedirà amb el nom "filtre". Les seves propietats agafaran els valors dels *input* que tinguin a l'atribut *name* el següent patró: "filtre.<nom_propietat>".

Acció protegida amb paràmetres de formulari i vista composta

- El nom de l'acció és "afegirPersona" i es relaciona al mètode *afegirPersona()*.
- És una acció amb tres paràmetres de formulari i s'hi accedeix amb la URL "/persones/crear".
- No es retorna cap model.
- Es presentarà una vista composta utilitzant una plantilla Tiles anomenada 'fitxa_persona' que s'haurà declarat prèviament a l'arxiu de configuració del framework de Apache Tiles.
- A més, l'anotació @RolesAllowed serveix per indicar que aquesta acció només es podrà executar pels usuaris que tinguin el rol ADMIN.

Acció protegida a usuaris registrats sense resultat

- El nom de l'acció és "afegirVot" i es relaciona al mètode *afegirVot()*.
- Es mapa a la URL "/persones/votar/#{idUsuari}" on el paràmetre de URL *idUsuari* és un enter que identificarà l'usuari a votar i s'utilitzarà com argument del mètode.
- Aquesta acció està protegida contra els usuaris autenticats amb el rol ADMIN o USUARI, pel que si un usuari amb aquests rols intenta executar aquesta acció se'l redirigirà a la pàgina d'error.
- Al no retornar cap resultat es redirigirà a la pàgina on es trobés l'usuari al moment de llançar la petició.

```
@Controller(name="personesController", path="/persones",
scope=ControllerScope.REQUEST)
public class PersonesController {

   @Action(path="/principal", params={})
   public void principal() {
      return new ModelAndView(null, "index", TilesResultDispatcher.class,
           null);
   }

   @Action(path="/llistat", params={"noms"})
   public ModelAndView llistat(String[] nomsPersones) {
      Persona[] persones = Logic.obtenirPersones(nomsPersones);
      Map<String, Object> model = new HashMap<String, Object>();
      ActionMessages messages = new ActionMessages();
      if (persones != null) {
         model.put("persones", persones);
      }
      else {
         messages.addErrorMessage("No hi ha persones amb aquests noms");
      }
      return new ModelAndView(model, "/persones/llistat_persones_nom.jsp",
           JspResultDispatcher.class, messages);
   }

   @Action(path="/llistat/#{grup}", params={"filtre"})
   public ModelAndView llistatGrup(Integer idGrup, FiltrePersona filtre) {
      Persona[] persones = Logic.obtenirPersonesGrup(idGrup, filtre);
      Map<String, Object> model = new HashMap<String, Object>();
      model.put("persones", persones);

      return new ModelAndView(model, "/persones/llistat_persones.jsp",
           JspResultDispatcher.class, null);
   }

   @Action(path="/crear", params={"nom", "cognom", "edat"})
   @RolesAllowed(roles={"ADMIN"})
   public ModelAndView afegirPersona(String nom, String cognom, Integer
           edat) {
      Persona persona = new Persona();
      persona.setNom(nom);
      persona.setCognom(cognom);
      persona.setEdat(edat);

      Logic.guardarPersona(persona);

      return new ModelAndView(null, "fitxa_persona",
           TilesResultDispatcher.class, null);
   }

   @Action(path="/votar/#{usuari}", params={})
   @RolesDenied(roles={"ADMIN", "USUARI"})
   public void afegirVot(Integer idUsuari) {
      Logic.sumaVot(idUsuari);
   }
}
```

Codi d'exemple 13. Exemple de controlador amb àmbit de petició.

Controlador de sessió

El segon exemple, que es pot veure al Codi d'exemple 14, és el d'un controlador amb àmbit de sessió que serveix per guardar el tema o estil en que es veurà la pàgina:

- Disposa de dues propietats accessibles: un llistat de temes disponibles, només en mode lectura mitjançant el seu getter, i el tema seleccionat, que es pot llegir i escriure.
- Té una acció anomenada *canviarTema*, que serveix per canviar el tema i a la que s'accedeix mitjançant la URL "/theme/#{theme}". Disposa d'un paràmetre de URL anomenat 'theme' que servirà per identificar el tema al que es canvia. No retorna cap *ModelAndView*, pel que es redirigirà a la pàgina anterior.

```
@Controller(path="/theme", name="themeController",
scope=ControllerScope.SESSION)
public class ThemeController {

   private String themeSelected = "orange";
   private static List<String> themesAvailable;
   static {
      themesAvailable = new ArrayList<String>();
      themesAvailable.add("orange");
      themesAvailable.add("blue");
   }

   @Action(path="/#{theme}", params={})
   public void canviarTema(String nouTema) {
      if (themesAvailable.contains(nouTema)) {
         this.themeSelected = nouTema;
      }
   }

   // Getters / Setters
   public String getThemeSelected() {
      return themeSelected;
   }
   public void setThemeSelected(String themeSelected) {
      this.themeSelected = themeSelected;
   }
   public List<String> getThemesAvailable() {
      return themesAvailable;
   }

}
```

Codi d'exemple 14. Exemple de controlador amb àmbit de sessió.

Tipus de paràmetres a les accions

Tal i com s'ha vist als exemples, Jewel permet que les accions tinguin dos tipus de paràmetres:

- Paràmetres de URL. Aquests paràmetres són els que es troben a la mateixa URL i només podran ser del tipus String o Integer.
- Paràmetres de formulari. Aquests paràmetres són els que el programador declara als formularis de la vista i es poden correspondre a JavaBean definits per l'usuari o un dels tipus de Java següents:
  - java.lang.String
  - java.lang.Integer

- java.lang.Long
- java.lang.Float
- java.lang.Double
- java.lang.Boolean
- java.util.Date
- java.lang.String[]
- java.lang.Integer[]
- java.lang.Long[]
- java.lang.Float[]
- java.lang.Double[]
- java.lang.Boolean[]
- java.util.Date[]

La conversió la realitza Jewel automàticament, pel que el programador només ha de declarar el tipus a l'argument i el nom del paràmetre a l'atribut *param* de l'anotació @Action. Cal tenir en compte que els paràmetres de tipus *Date* han de rebre un valor amb el temps en milisegons, cosa que es fa automàticament si s'utilitza l'etiqueta `<jwl:inputDate>`.

Per accedir a les propietats d'un JavaBean, el nom del paràmetre definit a la vista ha de contenir el nom de l'argument de l'acció i el nom de la propietat separats per un punt, tal i com es mostra a continuació:

```
<jwl:inputText name="filtre.nom" value=""></jwl:inputText>
```

En aquest codi es referenciaria a la propietat "nom" de l'argument "filtre" declarat a l'acció.

12.3.2. Filtres

Un filtre és una classe que té un o dos mètodes on es declara una funcionalitat que s'ha d'executar abans i/o després d'una acció. Es declara amb l'anotació @Filter i els mètodes han de tenir l'anotació @PreAction, si s'han d'executar abans, o l'anotació @PostAction, si s'executen després de l'acció. Un mateix mètode pot tenir les dues anotacions per indicar que s'executa abans i després de les accions.

El Codi d'exemple 15 mostra la manera d'implementar un filtre amb dos mètodes separats per cada funcionalitat. El primer mètode, *preAction*, marcat amb l'anotació @PreAction, s'executarà abans de l'acció. El segon mètode, *postAction*, marcat amb l'anotació @PostAction, s'executarà després de processar l'acció.

```
@Filter
public class LogFilter {

   @PreAction
   public void preAction(RequestContext ctx) {
      Logger.guardaLog(Actions.PRE_ACTION, ctx.getUrl(),
           ctx.getRequest().getUserPrincipal().getName(), new Date());
   }

   @PostAction
   public void postAction(RequestContext ctx) {
      Logger.guardaLog(Actions.POST_ACTION, ctx.getUrl(),
           ctx.getRequest().getUserPrincipal().getName(), new Date());
   }

}
```

Codi d'exemple 15. Implementació d'un filtre

Des dels mètodes d'un filtre es tindrà accés a l'objecte de context, pel que es podran utilitzar les propietats d'aquest i els paràmetres de petició.

Un filtre pot declarar-se perquè s'executi només en certes accions utilitzant l'atribut *onlyActions* per indicar els controladors i les seves accions on només s'executarà aquest filtre. Per exemple, al Codi d'exemple 16 el filtre només s'executarà a totes les accions del controlador PersonesController.

```
@Filter(onlyActions={
      @FilterAction(controller=PersonesController.class)
})
```

Codi d'exemple 16. Filtre amb l'atribut *onlyActions*

Al Codi d'exemple 17 s'utilitza l'atribut *excludedActions* per indicar que el filtre s'executarà sempre, excepte a les accions *principal* i *llistat* del controlador PersonesController.

```
@Filter(excludedActions={
      @FilterAction(controller=PersonesController.class,
            methods={"principal", "llistat"})
})
```

Codi d'exemple 17. Filtre amb l'atribut *excludedActions*

12.3.3. Presentació de vistes personalitzats

Jewel, tal i com s'ha explicat anteriorment, proporciona tres maneres de presentar els resultats: JSP, Apache Tiles i JSON. Però, a més, també permet que el desenvolupador pugui crear nous tipus de presentació de vistes implementant la interfície *org.jewel.core.ResultDispatcher*. A continuació es descriuen els passos que ha de seguir el programador per crear-ne una de nova.

1- Implementar la interfície ResultDispatcher

Al implementar *ResultDispatcher* s'ha de d'implementar els mètodes *dispatch* i *dispatchError*, que són els que utilitzarà el framework a l'hora de presentar el resultat de l'acció. La missió del primer mètode és adequar el model resultant de l'acció al tipus de resultat i redirigir a la vista. El segon mètode serveix per presentar l'excepció que s'hagi pogut produir durant l'execució de l'acció.

En el Codi d'exemple 18 es presenta un resultat en JSON utilitzant la llibreria Gson per convertir el model a JSON. També es converteix el tipus de contingut de la resposta a "application/json" i es redirigeix o, en aquest cas, s'escriu a la vista.

En cas que es produeixi un error en el mètode *dispatch* s'ha de llançar una excepció de tipus *ResultDispatchException* que serà tractada pel framework i redirigirà a la pàgina d'error de l'aplicació.

2- Desenvolupar la vista

En aquest pas és on es programa la vista, és a dir, allà on es redirigirà des del mètode *dispatch* i on es presentarà el resultat de l'acció. Per exemple, en el cas de la implementació per JSP s'ha de desenvolupar l'arxiu JSP, i en el cas de la implementació per

una plantilla Apache Tiles s'han de desenvolupar els arxius JSP necessaris. A l'apartat 12.3.4 es descriu aquest procés.

```
public class JsonResultDispatcher implements ResultDispatcher {

   @Override
   public void dispatch(ModelAndView mav, ServletRequest req,
        ServletResponse resp) throws ResultDispatchException {
      resp.setContentType("application/json");
      PrintWriter out;
      try {
         out = resp.getWriter();
         Gson gson = new Gson();
         // Convertim el model al model de sortida adequat al tipus de
         //presentació
         String model = gson.toJson(mav.getModel());
         // Presentam el resultat a la vista
         out.print(model);
         out.flush();
      }
      catch (IOException e) {
         throw new ResultDispatchException("Error dispatxant el resultat
              JSON", e);
      }
   }

   @Override
   public void dispatchError(Exception error, String errorPage,
           ServletRequest req, ServletResponse resp)
           throws ResultDispatchException {
      resp.setContentType("application/json");
      PrintWriter out;
      try {
         out = resp.getWriter();
         Gson gson = new Gson();
         // Convertim l'error a JSON
         String model = gson.toJson(error);
         out.print(model);
         out.flush();
      }
      catch (IOException e) {
         throw new ResultDispatchException(
              "Error dispatxant el resultat JSON", e);
      }
   }
}
```

Codi d'exemple 18. Implementació JSON de ResultDispatcher

3- Indicar al resultat de l'acció que s'utilitzarà aquesta implementació

Al programar una acció d'un controlador s'ha de retornar un *ModelAndView* on s'indica el model i la vista on es presentarà el resultat. A més, s'ha d'indicar el tipus de resultat, és a dir, la implementació de *ResultDispatcher* que s'utilitzarà. Al Codi d'exemple 13 s'ha vist una acció que retorna el següent resultat:

```
return new ModelAndView(model, "/persones/llistat_persones.jsp",
      JspResultDispatcher.class, messages);
```

Això significa que es retorna un model on el tipus de vista és JSP, implementat per la classe *JspResultDispatcher*, i la vista es troba a l'arxiu `"/persones/llistat_persones.jsp"`.

12.3.4. Vistes

A l'hora de presentar un resultat en una vista, Jewel implementa tres tipus possibles de resultats: JSP, Tiles i JSON. En els següents apartats es mostraran exemples per cada un dels tres. Tant si s'utilitza un JSP simple com plantilles Tiles, es recomana utilitzar les etiquetes de Jewel per tal de facilitar l'ús del framework.

JSP

Jewel permet l'ús de JavaServer Pages per mostrar els resultats de les accions. El programador pot utilitzar aquest tipus de vista. Al següent exemple s'utilitzarà una pàgina JSP simple amb un formulari que es comunicarà amb l'acció *llistatGrup* del Codi d'exemple 13.

El primer que cal fer per utilitzar les etiquetes de Jewel és referenciar la llibreria al principi del document mitjançant la següent línia:

```
<%@ taglib prefix="jwl" uri="http://jewel.org/taglib" %>
```

Després s'ha d'escriure el codi de la pàgina, on en el cas del Codi d'exemple 19, els camps d'entrada es relacionaran amb l'argument "filtre" quan s'executi l'acció. L'acció que s'executarà en pitjar el botó "Cercar" és *llistatGrup* del controlador *personesController*. La mateixa etiqueta s'encarregarà de generar la URL adequada a l'acció amb el paràmetre *grup*.

En haver-se executat l'acció, es mostrarà el llistat de persones a la taula generada per l'etiqueta `<jwl:dataTable>`, que recorrerà el llistat de persones amb l'element d'iteració 'persona', utilitzat per definir els valors de les columnes.

A més, s'observa com s'utilitza la variable 'msgs' per accedir als missatges internacionalitzats, que es mostraran en l'idioma actual de la sessió o l'idioma per defecte de l'aplicació.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="jwl" uri="http://jewel.org/taglib" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html;charset=ISO-8859-1">
   <title>Cercar persones</title>
</head>
<body>
<jwl:form name="f_crear_persona">
   <div>
      <jwl:inputText name="filtre.nom"></jwl:inputText>
      <jwl:inputText name="filtre.cognom"></jwl:inputText>
      <jwl:inputText name="filtre.edatIni"></jwl:inputText>
      <jwl:inputText name="filtre.edatFin"></jwl:inputText>
   </div>
   <div>
      <jwl:commandButton url="#{personesController.llistatGrup }"
           value="Cercar">
        <jwl:param name="grup" value="1" />
      </jwl:commandButton>
   </div>
   <div>
      <jwl:messages type="ALL" />
      <jwl:dataTable elements="#{persones }" var="persona">
        <jwl:column value="#{persona.nom }" title="#{msgs['nom'] }" />
        <jwl:column value="#{persona.cognom }" title="#{msgs['cognom'] }"
/>
        <jwl:column value="#{persona.edat }" title="#{msgs['edat'] }" />
      </jwl:dataTable>
   </div>
</jwl:form>
</body>
</html>
```

Codi d'exemple 19. Pàgina JSP a Jewel

Tiles

Les plantilles de Apache Tiles estan formades per arxius JSP i un arxiu de configuració XML on es defineixen les plantilles base i les vistes.

Per configurar les vistes de Apache Tiles s'haurà de seguir els següents passos:

1- Configurar l'arxiu web.xml per incloure els servlets i/o filtres del framework.
2- Crear un arxiu tiles.xml on es defineixen les plantilles i les vistes tal i com es mostra al Codi d'exemple 20. En l'exemple es crea una plantilla "base" que s'utilitza a la vista "fixa_persona", utilitzada al mètode *afegirPersona* del Codi d'exemple 13.
   A les vistes, normalment, només canviarà la part corresponent al cos de la base, és a dir, l'atribut "body".
3- Crear els arxius JSP de les plantilles i les vistes, que seguiran les mateixes regles que els arxius JSP simples descrites a l'apartat anterior.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
"http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
<tiles-definitions>
	<definition name="base" template="/template/base.jsp">
		<put-attribute name="header" value="/template/header.jsp" />
		<put-attribute name="body" value="/template/principal.jsp" />
		<put-attribute name="footer" value="/template/footer.jsp" />
	</definition>

	<definition name="fitxa_persona" extends="base">
		<put-attribute name="body" value="/fitxa.jsp" />
	</definition>
</tiles-definitions>
```

Codi d'exemple 20. Exemple d'arxiu de configuració de Apache Tiles

JSON

El tipus de resposta JSON s'utilitzarà normalment a les peticions Ajax que es realitzin mitjançant Javascript, pel que no es presentarà una vista en si, si no que es generarà una resposta on el tipus de contingut de la capçalera serà "application/json".

Per exemple, si l'acció *llistatGrup* del Codi d'exemple 13 retornés un tipus de resultat JSON en comptes de JSP, el resultat JSON seria similar al Codi d'exemple 21 i es podria tractar fàcilment des de Javascript amb frameworks com JQuery [28].

```
{"persones":[
   {"nom":"Juanma","cognom":"Lopez","edat":25},
   {"nom":"Juan Manuel","cognom":"Lopez","edat":25}]
}
```

Codi d'exemple 21. Resposta de petició JSON

[28] La generació d'una petició Ajax i el seu tractament queda fora de l'àmbit del disseny del projecte, però es pot trobar més informació a la pàgina oficial de JQuery: http://api.jquery.com/jQuery.ajax/

13. Disseny de l'extensió per Eclipse

L'objectiu de l'extensió per l'IDE Eclipse era agilitzar la configuració del framework i l'aprenentatge d'aquest. Després d'haver fet el disseny del framework, en contra del que s'havia planificat, es veu innecessari una extensió, ja que la configuració del framework és mínima i molt senzilla.

Només és necessari un únic arxiu XML, tal i com hem vist al capítol 12.2, on només s'ha de configurar 5 etiquetes i que disposa d'un XSD clar proporcionat amb el framework. A més, per la creació de components només es necessita saber utilitzar dues anotacions bàsiques per crear les accions, @Controller i @Action, i l'anotació @Filter per crear els filtres.

Per tant, es decideix que, en comptes de desenvolupar una extensió que no aportaria gaires avantatges, es profunditzarà més en la implementació de Jewel i en el desenvolupament de l'aplicació d'exemple, que faran més servei a l'usuari a l'hora de comprendre l'ús del framework.

14. Disseny de l'aplicació d'exemple

Per tal de demostrar el bon funcionament de Jewel Framework i la seva utilitat es dissenyarà i desenvoluparà una aplicació que utilitzi tot el seu potencial. L'aplicació serà senzilla però farà ús de totes les utilitats que proporciona el framework.

L'aplicació simularà el funcionament d'un gestor de recursos per projectes d'una empresa i tindrà els següents mòduls o apartats:

- Gestió de departaments. Es podran donar d'alta i editar departaments de l'empresa.
- Gestió d'empleats. Es podran donar d'alta i editar empleats d'una empresa, assignar-los a un o altre departament i canviar el seu perfil (cap de projecte, programador, analista, etc.).
- Gestió de projectes. Es podran crear i editar projectes, assignant-hi els empleats de l'empresa.

14.1. Requisits funcionals de l'aplicació

Codi	Descripció del requeriment
RFU_01	Un administrador podrà donar d'alta un departament de l'empresa.
RFU_02	Un administrador podrà editar les dades d'un departament de l'empresa.
RFU_03	Un usuari podrà consultar les dades d'un departament de l'empresa.
RFU_04	Un administrador podrà donar d'alta un empleat a un departament de l'empresa, indicant també el seu perfil i les seves habilitats.
RFU_05	Un administrador podrà editar les dades d'un empleat, excepte el seu codi d'empleat, que no es podrà actualitzar.
RFU_06	Un usuari podrà consultar les dades d'un empleat
RFU_07	Un administrador podrà donar d'alta un projecte.
RFU_08	Un administrador podrà editar les dades d'un projecte, excepte el codi que no es podrà actualitzar.
RFU_09	Un administrador podrà assignar empleats a un projecte, sempre i quan no estiguin donats de baixa.
RFU_10	Un usuari podrà consultar les dades d'un projecte i els empleats que hi estan assignats.
RFU_11	Un usuari podrà canviar l'idioma de l'aplicació i elegir entre Català (per defecte), castellà o anglès.
RFU_12	Un usuari podrà canviar l'estil visual de l'aplicació.

14.2. Base de dades

L'aplicació tindrà una petita base de dades formada per 7 taules. S'utilitza MySQL 5.1 amb el motor InnoDB[29] per realitzar les proves, tot i que al fer ús d'Hibernate[30], el canvi a un altre sistema gestor de base de dades no afectaria a l'aplicació. El model entitat relació és el següent:

[29] Tecnologia d'emmagatzemat estàndard de MySQL que suporta transaccions ACID (Atomicitat, Consistència, Aïllament i Durabilitat) i integritat referencial.
[30] Framework per la capa de persistència que facilita el mapeig d'atributs entre una base de dades relacional i el model d'objectes de l'aplicació.

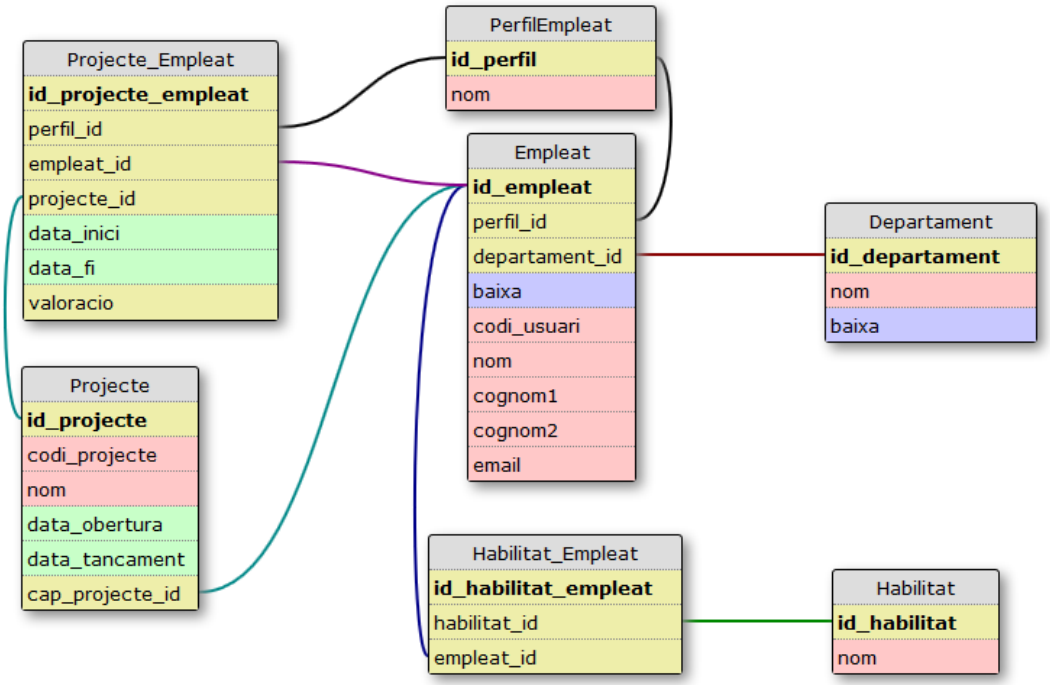


Figura 22. Model Entitat-Relació de l'aplicació d'exemple

A continuació s'explica amb més detall per a què serveix cada taula, els seus camps i la descripció d'aquests. S'utilitzen les següents abreviatures:

- PK - Primary Key: Clau primària de la taula.
- FK - Foreign Key: Cau estrangera cap a una altra taula.

Empleat	Informació dels empleats de l'empresa	
Camp	**Tipus**	**Descripció**
id_empleat	INTEGER (PK)	Identificador de l'empleat. Clau primària
perfil_id	INTEGER (FK)	Perfil professional de l'empleat. Clau estrangera cap a la taula PerfilEmpleat
departament_id	INTEGER (FK)	Departament al que pertany l'empleat. Clau estrangera cap a la taula Departament
baixa	BIT	Indica si l'empleat està donat de baixa o no
codi_usuari	VARCHAR(12)	Codi d'identificació de l'usuari dins l'empresa
nom	VARCHAR(30)	Nom de l'empleat
cognom1	VARCHAR(30)	Primer cognom de l'empleat
cognom2	VARCHAR(30)	Segon cognom de l'empleat
email	VARCHAR(30)	Correu electrònic de contacte

PerfilEmpleat	Perfils d'empleat disponibles a l'empresa. P.ex: Cap de projecte, arquitecte, programador, ...	
Camp	**Tipus**	**Descripció**
id_perfil	INTEGER (PK)	Identificador del perfil. Clau primària
nom	VARCHAR(20)	Nom descriptiu del perfil d'empleat.

Departament	Departament o secció de l'empresa. P.ex: Dept. informàtic, Dept. econòmic, ...	
Camp	**Tipus**	**Descripció**
id_departament	INTEGER (PK)	Identificador del departament. Clau primària
nom	VARCHAR(30)	Nom identificatiu del departament
baixa	BIT	Indica si l'empleat està donat de baixa o no

Habilitat	Habilitat d'un empleat. P.ex: J2EE, Anàlisi Mètrica, HTML5, ...	
Camp	**Tipus**	**Descripció**
id_habilitat	INTEGER (PK)	Identificador de l'habilitat. Clau primària
nom	VARCHAR(20)	Nom descriptiu de l'habilitat

Habilitat_Empleat	Relació entre els empleats i les habilitats	
Camp	**Tipus**	**Descripció**
id_habilitat_empleat	INTEGER (PK)	Identificador de la relació. Clau primària
habilitat_id	INTEGER (FK)	Identificador de l'habilitat que es relaciona
empleat_id	INTEGER (FK)	Identificador de l'empleat que es relaciona

Projecte	Projectes de l'empresa	
Camp	**Tipus**	**Descripció**
id_projecte	INTEGER (PK)	Identificador del projecte. Clau primària
codi_projecte	VARCHAR(10)	Codi d'identificació del projecte utilitzat a la documentació de l'empresa
nom	VARCHAR(30)	Nom descriptiu del projecte
data_obertura	DATE	Data en que s'inicia el projecte
data_tancament	DATE	Data en que finalitza el projecte
cap_projecte_id	INTEGER (FK)	Identificador de l'empleat que és el cap de projecte

Projecte_Empleat	Relació entre empleat i projecte per designar els empleats que han estat assignats a un projecte	
Camp	**Tipus**	**Descripció**
id_projecte_empleat	INTEGER (PK)	Identificador de la relació. Clau primària
empleat_id	INTEGER (FK)	Identificador de l'empleat assignat al projecte
projecte_id	INTEGER (FK)	Identificador del projecte que es relaciona
perfil_id	INTEGER (FK)	Identificador del perfil que tenia l'empleat al moment d'assignar-lo al projecte
data_inici	DATE	Data de començament de l'empleat al projecte
data_fi	DATE	Data de finalització de la relació entre l'empleat i el projecte
valoracio	FLOAT	Valoració que li donen a l'empleat respecte la feina feta al projecte

14.3. Característiques de Jewel utilitzades

El gestor de recursos i projectes desenvolupat utilitzarà les funcionalitats de Jewel tal i com s'explica als següents apartats.

14.3.1. Accions i controladors

L'aplicació farà ús de 9 controladors que agruparan una sèrie d'accions relacionades. La majoria d'aquestes accions es corresponen amb els requisits funcionals del punt 14.1.

- **AdminDepartamentsController**: Controlador amb àmbit de petició que conté les accions d'administració dels departaments:
  - Nou departament. Prepara les dades necessàries per presentar la pantalla de creació d'un nou departament.
  - Crear departament. Recull les dades del departament introduïdes i el crea. L'única dada que ha d'introduir l'administrador és el nom.
  - Editar departament. Prepara les dades per consultar un departament i, si es canvien, les guarda. L'única dada que pot modificar l'administrador és el nom.
- **AdminEmpleatsController**: Controlador amb àmbit de petició per administrar els empleats de l'empresa. Conté els següents accions:
  - Nou empleat. Prepara les dades necessàries per crear un nou empleat, tals com el llistat de departaments, el llistat de perfils d'empleat i el llistat d'habilitats disponibles per seleccionar.
  - Crear empleat. Crea un nou empleat a partir de les dades introduïdes per l'administrador, que són: codi d'usuari, nom, primer cognom, segon cognom, departament, perfil i habilitats. D'aquestes dades, l'única opcional és el segon cognom.
  - Editar empleat. Prepara les dades necessàries per canviar les dades d'un empleat i, si es canvien, les actualitza. Pot actualitzar totes les dades excepte el codi d'usuari.
  - Comprova codi d'usuari. Acció Ajax per consultar si un codi d'usuari ja s'utilitza o està disponible per un nou empleat. Retorna una resultat JSON que es tractarà amb Javascript.
- **AdminProjectesController**: Controlador d'àmbit de petició per gestionar els projectes de l'empresa que disposa de les següents accions:
  - Nou projecte. Prepara les dades necessàries per crear un nou projecte: llistat d'empleats que poden esser seleccionats com a cap de projecte.
  - Crear projecte. Guarda les dades introduïdes del nou projecte: codi, nom, cap de projecte, data d'obertura i data de tancament. D'aquestes, el camp de data de tancament es pot deixar buit.
  - Editar projecte. Prepara les dades del projecte i les recull per actualitzar-les. El codi del projecte no es pot actualitzar.
  - Afegir empleat. Afegeix un empleat al projecte entre un rang de dates.
  - Editar empleat. Edita l'assignació de l'empleat al projecte, podent canviar la data final i la valoració.
- **DepartamentsController**: Controlador d'àmbit de petició per consultar les dades dels departaments de l'empresa amb les següents accions:
  - Cercar departaments. Cerca un llistat de departaments a partir d'un filtre que es correspon amb un formulari per introduir el nom del departament que es cerca.

    - Consultar departament. Consulta les dades d'un departament a partir del seu identificador i les mostra.
- **EmpleatsController**: Controlador amb l'àmbit de petició que conté les accions necessàries per consultar les dades dels empleats de l'empresa:
    - Cercar empleats. Cerca i mostra dins una taula els empleats que coincideixen amb els següents criteris del formulari de cerca: codi d'usuari, nom, primer cognom, segon cognom, departament i/o perfil d'empleat.
    - Consultar empleat. Consulta i mostra les dades d'un empleat a partir del seu identificador.
    - Fitxa d'empleat. Consulta i mostra les dades d'un empleat a una finestra llesta per imprimir a partir del seu codi d'usuari.
- **PrincipalController**: Controlador de petició amb una única acció:
    - Índex. Mostra la pàgina d'inici utilitzant una plantilla de Apache Tiles.
- **ProjectesController**: Controlador de petició amb les següents accions:
    - Cercar projecte. Mostra un llistat de projectes a partir d'un filtre amb els següents criteris de cerca: codi del projecte, nom, cap de projecte, departament, data d'obertura i data de tancament.
    - Consultar projecte. Presenta les dades d'un projecte a partir del seu identificador.
- **ThemeController**. Controlador dins l'àmbit de sessió on es guarda el tema visual en què l'usuari vol veure l'aplicació. Disposa d'una única acció per canviar el tema, que té per paràmetre el nom identificador del nou tema.

14.3.2. URL amigables

Jewel és un framework que promou les direccions URL amigables, és a dir, que aportin informació sobre el que es visualitzarà i no només un conjunt de codis indesxifrables. A l'aplicació s'utilitzarà el següent mapeig entre les direccions i les accions que representen:

Direcció URL	Acció
/admin/depts/nou	Nou departament
/admin/depts/crear	Crear departament
/admin/depts/editar/#{id}	Editar departament
/admin/empleats/nou	Nou empleat
/admin/empleats/crear	Crear empleat
/admin/empleats/editar/#{id}	Editar empleat
/admin/empleats/json/comprova-codi	Comprova codi d'usuari
/admin/projectes/nou	Nou projecte
/admin/projectes/crear	Crear projecte
/admin/projectes/editar/#{id}	Editar projecte
/departaments/cercar	Cercar departaments
/departaments/veure/#{id}	Consultar un departament
/empleats/cercar	Cercar empleats
/empleats/consultar/#{id}	Consultar un empleat
/empleats/fitxa/#{codi_usuari}	Fitxa d'un empleat
/principal/index	Índex
/projectes/cercar	Cercar projectes
/projectes/consultar/#{id}	Consultar un projecte
/theme/#{codi_tema}	Canviar tema visual

14.3.3. Filtres

L'aplicació necessita registrar a un fitxer de depuració o *log* la informació de quan s'executen les accions d'administració. Per tant, es crearà un filtre anomenat *LogFilter* que només s'executi a les accions dels controladors d'administració. Per tant, presentarà el següent codi:

```
@Filter(onlyActions={
     @FilterAction(controller=AdminDepartamentsController.class),
     @FilterAction(controller=AdminEmpleatsController.class),
     @FilterAction(controller=AdminProjectesController.class)}
)
public class LogFilter {
   Logger _log = Logger.getLogger(LogFilter.class);

   @PreAction
   public void preAction(RequestContext ctx) {
      _log.debug("\nLogFilter at " +
           DateUtil.formatDate(new Date()) + " Acció: " + ctx.getUrl());
   }
}
```

Codi d'exemple 22. Filtre LogFilter de l'aplicació d'exemple

14.3.4. Configuració amb anotacions i XML

La configuració de Jewel a l'aplicació es farà mitjançant un únic arxiu XML anomenat jewel-config.xml, tal i com es mostra al Codi d'exemple 12, on configurarem el principal del framework.

En quan als filtres, els controladors i les seves accions es configuraran mitjançant les anotacions que aporta Jewel. D'aquesta manera queda un codi net i fàcil de mantenir.

14.3.5. Internacionalització

A l'arxiu de configuració s'indica que s'utilitzarà el filtre *LocaleFilter* que proporciona Jewel, pel que es podrà utilitzar missatges localitzats i es canviarà d'idioma amb un menú desplegable generat per l'etiqueta `<jwl:localeSelect>`. A més, es defineix com a idioma per defecte el català (codi "ca") i com idiomes addicionals el castellà i l'anglès.

Per tal d'utilitzar missatges localitzats s'indica que l'arxiu de recursos es troba a l'arrel del directori de classes i té el nom base "messages". Així, es crearan tres arxius de propietats amb els missatges:

- *messages_ca.properties*. Missatges en català.
- *messages_es.properties*. Missatges en castellà.
- *messages_en.properties*. Missatges en anglès.

14.3.6. Vistes: ús de Apache Tiles, JSP i JSON.

L'aplicació de gestió utilitzarà els tres tipus de resultats possibles que aporta Jewel:

- **JSP**: S'utilitzarà aquest resultat per presentar la fitxa d'empleat a l'acció del mateix nom.
- **JSON**: Aquest tipus de resultat serà d'utilitat a l'hora de comprovar la disponibilitat d'un codi d'usuari. Es cridarà a l'acció mitjançant Ajax, amb l'ajuda de jQuery, i el resultat JSON es tractarà per informar a l'usuari si el codi està utilitzat o no.

- **Apache Tiles**: S'utilitzarà la funcionalitat de vista composta que aporta aquesta llibreria per tota la resta d'accions de l'aplicació. La vista composta es compondrà de 4 parts: la base a l'arxiu "/template/base.jsp", la capçalera a l'arxiu "/template/header.jsp", el peu a l'arxiu "/template/footer.jsp" i el cos de la vista, que és la part variable de la vista i variarà en funció de l'acció executada.
  La definició de les vistes compostes es realitzarà a l'arxiu de configuració de Apache Tiles: tiles.xml.

14.3.7. Ús de la llibreria d'etiquetes

Aquesta aplicació utilitzarà la llibreria d'etiquetes que proporciona Jewel fent ús de totes les etiquetes per demostrar que funcionen correctament i per beneficiar-se del que aporten: facilitat de mapat entre camps del formulari i paràmetres de l'acció, facilitat per mapar els formularis o enllaços a les accions de l'aplicació i reducció del codi necessari a les vistes per presentar els resultats d'una acció i per presentar els llistats d'elements.

Per poder utilitzar la llibreria d'etiquetes de Jewel s'ha d'incorporar l'arxiu TLD a la carpeta WEB-INF de l'aplicació i importar-lo als JSP que la utilitzin amb el següent scriptlet:

```
<%@ taglib prefix="jwl" uri="http://jewel.org/taglib" %>
```

14.3.8. Validació de formularis

Jewel facilita la validació al costat del client de camps obligatoris a formularis, funcionalitat que s'utilitzarà als formularis on s'introdueixin dades (vistes on crear i editar projectes, departaments i empleats) per validar que els camps tenen valor. Per tant, tots els camps que a l'apartat 14.3.1 s'han definit com obligatoris tindran l'atribut *required=true* a l'etiqueta que els representa, a més d'un missatge localitzat a l'atribut *requiredMessage*, que es mostrarà si el camp és buit a l'hora d'enviar el formulari.

14.3.9. Protecció davant reenviament de formularis

S'utilitzarà protecció davant reenviament de formularis a tots aquells on es modifiquin dades, és a dir, als de creació o edició de dades de la part d'administració. D'altre banda, els formularis de cerca tindran inhabilitada aquesta comprovació indicant-ho al botó d'acció `<jwl:commandButton>` mitjançant l'atribut *tokenCheck* a fals.

L'única acció que no comprovarà el reenviament de formularis és la comprovació del codi d'usuari, ja que al tractar-se d'una petició Ajax de consulta no cal comprovar el reenviament de formulari ni generar una nova mostra.

14.3.10. Seguretat

Com ja s'ha explicat, Jewel proporciona dues anotacions per limitar l'execució de les accions als rols indicats. Això es farà servir per limitar l'execució de les accions dels controladors d'administració al rol d'administrador utilitzant l'anotació @RolesAllowed.

A més, el fet d'utilitzar URL amigables també facilita l'ús de les restriccions de seguretat J2EE que es poden configurar a l'arxiu web.xml, pel que també es podria definir una restricció de seguretat sobre el patró URL "/admin/*" tal i com es mostra al Codi d'exemple 23.

En comptes de fer-ho així es decideix no posar aquesta restricció, per demostrar la funcionalitat dels permisos sobre les accions i mostrar el missatge d'error controlat des de la pròpia aplicació en comptes del missatge d'error del servidor.

Encara que, per protegir tota l'aplicació aplicarem una restricció de seguretat a tot el domini per tal que només puguin accedir usuaris amb els rols 'jewel_admin' o 'jewel_user'.

```
...
  <security-constraint>
   <web-resource-collection>
      <web-resource-name>Accions d'administració</web-resource-name>
      <description></description>
      <url-pattern>/admin/*</url-pattern>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
   </web-resource-collection>
   <auth-constraint>
      <description>Rols permesos</description>
      <role-name>jewel_admin</role-name>
   </auth-constraint>
  </security-constraint>
...
```

Codi d'exemple 23. Configuració de la seguretat a l'aplicació d'exemple gràcies a la URL amigable

14.4. Implementació del projecte

Per començar, s'han seguit les passes indicades al capítol 12 per configurar Jewel i afegir les dependències necessàries. Al utilitzar una base de dades, s'ha afegit també la llibreria que conté el connector de Java per MySQL i les llibreries de Hibernate.

Per l'accés a BD s'ha seguit el patró DAO genèric (PATRICIO, ANTHONY; EBERSOLE, STEVE. "Generic Data Access Objects"), on cada classe DAO estén una classe genèrica i serveix per accedir a una taula a través de l'entitat persistent JPA que la representa. Per sobre dels DAO hi han les classes de servei que implementen la lògica de l'aplicació i contenen els mètodes que cridaran les accions dels controladors.

D'aquesta manera es separen els rols de cada capa i es millora molt la mantenibilitat de l'aplicació. Tal i com es pot veure, cada paquet té la seva funcionalitat:

Paquet Java	Funció
org.jewel.example.action	Controladors Jewel amb les accions
org.jewel.example.filters	Filtres Jewel
org.jewel.example.javabean	Classes d'ajuda
org.jewel.example.persistence	Entitats JPA
org.jewel.example.services	Classes de la capa de negoci
org.jewel.example.services.dao	Classes de la capa d'accés a BD, a través d'Hibernate
org.jewel.example.util	Classes d'utilitats

15. Conclusions finals

El desenvolupament d'un framework no és una feina fàcil, ja que du un treball d'investigació elevat i un temps d'implementació llarg. Però crec que he après molt i ha valgut la pena dedicar tot el temps que hi he hagut de dedicar.

He assolit els objectius que em vaig marcar al principi:

- He après com funcionen i com s'utilitzen els frameworks analitzats, pel que ja tinc una bona base per si els he d'utilitzar al món professional.
- He millorat coneixements de J2EE i après a utilitzar correctament els patrons J2EE i perquè és important el seu ús.
- He profunditzat sobre el patró MVC i perquè és convenient utilitzar-lo.
- He crescut com a analista, al haver de pensar tot el que volia pel framework; com a arquitecte de programari, per la definició i disseny de Jewel; i com a programador, per la implementació del projecte.
- He comprès la importància d'una bona documentació i a utilitzar eines com Javadoc per proporcionar una API completa.
- He après a utilitzar llibreries Java que no havia emprat mai i a cercar i aprendre a utilitzar allò que em feia falta per poder continuar amb el projecte.

En quant al producte, puc dir que estic orgullós del framework resultant, ja que considero que és plenament funcional, tot i que sempre es pot millorar. He pogut comprovar que facilita el desenvolupament d'aplicacions amb l'aplicació d'exemple, que m'ha dut menys temps del que havia planificat al principi, pel que l'he pogut utilitzar per millorar més el framework.

En definitiva, ha estat un projecte ambiciós i motivador des d'un principi que m'ha fet créixer com enginyer informàtic i m'ha fet gaudir d'aquest final de carrera.

Ampliacions

Tot i que em sembla que el framework ha quedat bastant complet hi ha diverses línies d'ampliació futures que es podrien implementar per millorar la qualitat del producte:

- Enviament d'arxius a les accions. S'hauria d'implementar una etiqueta <jwl:inputFile> que generés un camp d'entrada de tipus *file*. El formulari passaria a ser de tipus "multipart" i l'acció recolliria un argument de tipus *java.io.File*.
- Expressions amb arrays. Les etiquetes, a més d'acceptar objectes Collection, acceptarien objectes Array per iterar sobre ells o per accedir als seus elements.
- Extensió de la llibreria d'etiquetes. Es podrien proporcionar moltes més etiquetes, a l'estil de JSF.
- Ampliar la integració amb Ajax. Tot i que es permeten peticions Ajax, Jewel podria proporcionar etiquetes que facilitessin les operacions amb Ajax, tal i com fan altres frameworks.

- Integració amb Maven. Es podria generar un *archetype*[31] de Maven per què fos més fàcil incorporar Jewel a un projecte tenint un esquelet de projecte per començar a desenvolupar ràpidament. A més, amb Maven no s'haurien de preocupar de les dependències d'aquest, ja que les descarregaria automàticament.
- Regles de navegació. Es podria afegir una configuració per restringir la navegació entre pàgines, de manera que es pogués definir de quina a quina pàgina es pot anar.

[31] Achetype de Maven: Extensió que permet a un usuari crear un projecte a partir d'una plantilla anomenada *archetype* i que, a partir de l'arxiu POM.xml generat, permet descarregar les dependències automàticament.

Bibliografia

ALUR, DEEPAK; CRUPI, JOHN; MALKS, DAN (2003, 10 de juny). *Core J2EE Patterns: Best Practices and Design Strategies* (2a. ed.). Estats Units: Prentice Hall.

APACHE FOUNDATION. "Apache Struts 2 Documentation". [Data de consulta: octubre de 2011] <http://struts.apache.org/2.x/docs/guides.html>

APACHE FOUNDATION. "Apache Commons Documentation". [Data de consulta: desembre de 2011]
<http://commons.apache.org/>

APACHE FOUNDATION. "Apache Tiles 2.2 Documentation". [Data de consulta: desembre de 2011] <http://tiles.apache.org/>

APACHE FOUNDATION. "Apache Tomcat 7.0 Documentation". [Data de consulta: desembre de 2011]
<http://tomcat.apache.org/tomcat-7.0-doc/jndi-datasource-examples-howto.html>

BELAEVSKI, NICK (2010, 25 de febrer). "Mock Objects for Test Driven JSF Development". [Data de consulta: novembre de 2011]
<http://community.jboss.org/wiki/MockObjectsForTestDrivenJSFDevelopmentorgjbosstest-jsfjsf-mockProject>

BERNARD, EMMANUEL i altres autors. "JSR 303 Bean Validation Specification". [Data de consulta: desembre de 2011]
<http://jcp.org/en/jsr/detail?id=303>

BERNARD, EMMANUEL; EBERSOLE, STEVE; KING, GAVIN. "Hibernate Entity Manager. Setup and configuration". [Data de consulta: gener de 2012]
<http://docs.jboss.org/hibernate/entitymanager/3.6/reference/en/html/configuration.html>

Diversos autors. "Reference Documentation". Spring Framework. [Data de consulta: octubre de 2011]
<http://static.springsource.org/spring/docs/current/spring-framework-reference/html/>

Diversos autors. "JavaServer Faces API (2.0)". [Data de consulta: octubre de 2011]
<http://download.oracle.com/docs/cd/E17802_01/j2ee/javaee/javaserverfaces/2.0/docs/api/index.html>

DONALD, KEITH (2010, 25 de gener). "Ajax simplifications in Spring 3.0". [Data de consulta: novembre de 2011]
<http://blog.springsource.com/2010/01/25/ajax-simplifications-in-spring-3-0/>

DOUGLAS, JAMES (2010). "Barebones Spring MVC". [Data de consulta: octubre de 2011] <http://www.etnassoft.com/biblioteca/barebones-spring-mvc/>

FOWLER, MARTIN (2005, 26 de juny). "Inversion of Control". [Data de consulta: octubre de 2011] <http://martinfowler.com/bliki/InversionOfControl.html>

GEARY, DAVID; HORSTMANN, CAY (2010). *Core JavaServer Faces* (3a. ed.). Estats Units: Prentice Hall

GEARY, DAVID. "Simply Singleton". [Data de consulta: desembre de 2011]
<http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-designpatterns.html>

HENNEBRUEDER, SEBASTIAN (2009, novembre). "JSF 2 – Evaluation test". [Data de consulta: novembre de 2011]
<http://www.lalilуna.de/articles/posts/jsf-2-evaluation-test.html>

HIGHTOWER, RICHARD (2008, 29 de gener). "Getting started with JavaServer Faces 1.2". [Data de consulta: octubre de 2011]
<http://www.ibm.com/developerworks/java/tutorials/j-jsf2/index.html>

KING, GAVIN; BAUER, CHRISTIAN; ANDERSEN, MAX; BERNARD, EMMANUEL; EBERSOLE, STEVE;. "Hibernate 3.3 Reference Documentation. Tutorial". [Data de consulta: gener de 2012]
<http://docs.jboss.org/hibernate/core/3.3/reference/en/html/tutorial.html>

ORACLE SUN. "Java SE Security Documentation". [Data de consulta: desembre de 2011]
<http://www.oracle.com/technetwork/java/javase/jaas/index.html>

PATRICIO, ANTHONY; EBERSOLE, STEVE. "Generic Data Access Objects". [Data de consulta: gener de 2012]
<http://community.jboss.org/wiki/GenericDataAccessObjects>

PÉREZ GARCÍA, ALEJANDRO. "Tutorial: JSF 2 ya está aquí". [Data de consulta: octubre de 2011]
<http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=jsf2Return>

RAIBLE, MATT (2010). "Comparing JVM Web Frameworks". [Data de consulta: novembre de 2011]
<http://www.parleys.com/#st=5&id=2118&sl=1>

ROSE INDIA TECHNOLOGIES (2008, 18 de desembre). "Detailed introduction to Struts 2 Architecture". [Data de consulta: octubre de 2011]
http://www.roseindia.net/struts/training/day1/struts2architecture.shtml

SINGH, INDERJEET; LEITCH, JOEL; WILSON, JESSE. "Gson User Guide". [Data de consulta: desembre de 2011]
<https://sites.google.com/site/gson/gson-user-guide>

VIRAL PATEL (2009, 22 de desembre). "Introduction to Struts 2 Framework". [Data de consulta: octubre de 2011]
<http://viralpatel.net/blogs/2009/12/introduction-to-struts-2-framework.html>

Annex I: Instal·lació de l'aplicació d'exemple

1. Descarregar el servidor d'aplicacions Tomcat 7.0.x de la pàgina oficial: http://tomcat.apache.org/download-70.cgi
2. Per instal·lar-lo, si s'ha descarregat l'arxiu ZIP, només cal descomprimir-lo a una carpeta.
3. Afegir els usuaris i rols de l'aplicació:
    a. Obrim l'arxiu <directori_tomcat>/conf/tomcat-users.xml.
    b. Afegim els següents usuaris i rols just abans de l'etiqueta </tomcat-users>:

```
<role rolename="jewel_admin"/>
<role rolename="jewel_user"/>
<user username="jlopez" password="jlopez" roles="jewel_admin"/>
<user username="mclosa" password="mclosa" roles="jewel_user"/>
```

4. Crear la base de dades:
    a. Cal tenir un MySQL 5 instal·lat. Si no es té es pot descarregar de la pàgina oficial: http://dev.mysql.com/downloads/mysql/5.0.html
    b. Executar els arxius '1_tables.sql' i '2_inserts.sql' en aquest ordre per crear l'esquema i les taules amb les dades inicials.
5. Afegir la configuració a la base de dades pel context de l'aplicació:
    a. Editar l'arxiu <directori_tomcat>/conf/server.xml.
    b. Afegir el següent fragment de codi just abans de l'etiqueta </Host>:

```
<Context docBase="Jewel-Project" path="/jewel-project"
         reloadable="true"
         source="org.eclipse.jst.jee.server:Jewel-Project">

    <Resource name="jdbc/JewelExampleDS" auth="Container"
           type="javax.sql.DataSource"
           maxActive="100" maxIdle="30" maxWait="10000"
           username="root" password=""
           driverClassName="com.mysql.jdbc.Driver"
           url="jdbc:mysql://localhost:3306/jewel_example"/>
</Context>
```

    Aquesta configuració és per una base de dades MySQL en local al port per defecte: 3306. Es referencia a un esquema de nom 'jewel_example', tal i com ve preparat a l'script de base de dades.
6. Configuració de Log4j, el sistema de *logging* elegit per implementar SLF4j:
    a. Si es vol configurar el log perquè surtin la informació del framework es pot fer creant un arxiu "log4j.properties" a la carpeta <directori_tomcat>/lib que segueixi el següent model:

```
log4j.rootLogger=INFO, A1
log4j.appender.A1.Threshold=INFO
log4j.category.org.apache.commons.digester=INFO
# A1 is set to be a ConsoleAppender which outputs to System.out.
log4j.appender.A1=org.apache.log4j.ConsoleAppender
# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
# The conversion pattern uses format specifiers. You might want to
# change the pattern an watch the output format change.
log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %37c %3x -
%m%n
```

Per més informació sobre els arxius de configuració de Log4j es pot consultar la pàgina oficial:
http://logging.apache.org/log4j/1.2/

7. Copiar l'arxiu <jewel-project.war> que conté l'aplicació a la carpeta <directori_tomcat>/webapps.
   L'aplicació ja porta totes les dependències necessàries, incluint el connector per MySQL, les llibreries d'Hibernate 4 i Log4j, pel que no cal afegir res al servidor.
8. Iniciar el servidor Tomcat 7.0:
   a. Des de Windows: executar l'arxiu <directori_tomcat>/bin/startup.bat
   b. Des de Linux: executar l'arxiu <directori_tomcat>/bin/startup.sh
9. Accedir a l'aplicació amb un navegador a la següent URL:
   http://localhost:8080/jewel-project/
   Posar l'usuari "mclosa-mclosa" per accedir amb el rol d'usuari o l'usuari "jlopez-jlopez" per accedir com administrador.

Annex II: Dependències del framework

Jewel Framework necessita un conjunt de llibreries per compilar. A continuació s'enumeren i es diu on descarregar-les.

- **Apache Commons Digester 2.1**
  http://commons.apache.org/digester/download_digester.cgi
- **Apache Commons BeanUtils 1.8.3**
  http://commons.apache.org/beanutils/download_beanutils.cgi
- **Google Gson 2.0**
  http://code.google.com/p/google-gson/downloads/list
- **Apache Tiles 2.2.2 (arxius API i Servlet)**
  http://tiles.apache.org/download.html
- **SLF4j 1.6.4**
  http://slf4j.org/download.html

A més, per utilitzar Jewel amb totes les seves funcionalitats s'han d'incloure al projecte les següents llibreries (ja incloses al WAR de l'aplicació d'exemple).

- **Apache Commons Collections 3.2.1**
  http://commons.apache.org/collections/download_collections.cgi
- **Apache Commons Logging 1.1.1**
  http://commons.apache.org/logging/download_logging.cgi
- **Apache Tiles 2.2.2 (arxius Core, JSP i Template)**
  http://tiles.apache.org/download.html
- **Connector per la implementació de SLF4j**
  http://slf4j.org/download.html
- **Implementació d'un Log compatible amb SLF4j**
  - **Log4j**
    http://logging.apache.org/log4j/1.2/download.html