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

Emilio Verdejo Grasa
Enginyeria en Informàtica

Consultor: Josep María Camps
18/06/2012

Llicència

Agraïments

A la meva dona Vanessa, per el recolzament mostrat en tot aquest temps en la meva decisió per acabar aquesta carrera i per el temps que li he pres amb els estudis.

A la meva mare, per la seva fortalesa demostrada en els moments difícils i saber donar-me la empenta necessària per seguir endavant.

A la meva filla Aina, que sense arribar als dos anyets ja li comença a agradar la tecnologia: desitjo que sigui capaç de valorar una bona formació acadèmica tal com la valorem els seus pares.

En recuerdo de mi padre,

Por su actitud, nobleza i saber ser
Un buen padre, un buen marido i un buen abuelo.

Resum del projecte Final de Carrera - J2EE

En assignatures tractades en aquesta carrera he estudiat com crear aplicacions web de client prim i m'han posat de manifest una sèrie de problemàtiques que es troba un programador quan vol crear una aplicació web.

Aquestes aplicacions tracten de separar en tres capes el seu desenvolupament:

- La capa web o de presentació es la que veu el client de l'aplicació al seu navegador i ha de tenir una interfície gràfica agradable i adaptada a les seves necessitats.

- La capa intermèdia o de negoci es pròpia de cada projecte i porta intrínseca la lògica pròpia del projecte.

- Finalment la capa de dades enllaça de forma transparent la persistència de les dades amb la base de dades.

La modularitat d'aquestes tres capes permet que al canviar qualsevol capa el projecte continuar funcionant facilitant d'aquesta manera el desacoblament del codi.

Aquest projecte de final de carrera que s'engloba dins de la àrea de J2ee (Java 2 Entrerprise Edition) pretén especialitzar-se en la capa de presentació desenvolupar un marc de treball (framework) que li faciliti al programador les classes necessàries per poder posar en marxa de forma àgil aquesta capa de les aplicacions de client prim.

Estudiarem quin son els problemes que es trobarà qualsevol programador al desenvolupar la capa de presentació d'una aplicació web, quins patrons d'estudi donen solució a aquestes problemàtiques. Veurem com resolt aquest problemes productes desenvolupats al mercat informàtic de java.

Després d'estudiar els productes que hi ha al mercat, desenvoluparé el meu framework per tal de donar-li al programador una eina de ràpida configuració i fàcil utilització en les seves funcions.

Finalment demostraré amb la creació de una aplicació web com ha d'utilitzar el programador el framework i com facilita el desenvolupament de la capa de presentació de aplicacions web.

Índex de continguts

Índex de continguts

Índex de figures

1.- Introducció del Projecte Final de Carrera - J2EE

1.1- Definició del projecte:

El projecte final de carrera escollit s'engloba dins de la temàtica J2EE que ja s'ha desenvolupat dins de la carrera en assignatures com enginyeria de programari de components i sistemes distribuïts

En aquesta assignatura s'han tractat les aplicacions que es desenvolupaven per Internet com aplicacions de client prim.

Per tal de facilitar el manteniment i canvis sobre una aplicació de client prim normalment aquestes aplicacions s'estructuren en tres capes de treball: la capa de presentació, la capa de negoci i la capa de persistència.

La capa de persistència te com a responsabilitat el manteniment i tractament de les dades de l'aplicació per les altres capes, ja sigui implementat aquesta persistència sobre una base de dades, fitxers xml o altres recursos.

La capa de negoci implementa les funcionalitats pròpies de cada projecte i es la que es responsabilitza de quines funcions podrà demanar la capa de presentació sobre la aplicació web.

Finalment la capa de presentació es la que estableix la presentació de l'aplicació al navegador, la navegació entre pàgines i permisos sobre aquestes accions.

Aquesta capa te implícita una sèrie de problemes com pot ser la centralització de les peticions que fa l'usuari sobre l'aplicació, el control de les accions que pot o no pot fer l'usuari, la presentació de les dades al navegador.

Son aquest problemes els que plantejarem en aquest projecte i com els podem resoldre. Moltes d'aquestes problemàtiques ja se l'han trobat molts programadors i les podem resoldre aplicant patrons de programació com pot ser el MVC (Model Vista Controlador) i els patrons J2EE per aquesta capa.

Estudiarem els patrons mes utilitzats i veurem com podem facilitar la resolució de problemes al nostre projecte.

A mes a mes estudiarem com al mercat de desenvolupament d'aplicacions J2EE estan sorgint diferents propostes que intenten solucionar la problemàtica que es dona a la capa de presentació.

Aquestes propostes, denominades frameworks de presentació , normalment estudien la problemàtica de la capa de presentació des de diferents visions i donen una solució java (normalment fitxers jars) que facilita la implementació als desenvolupadors d'aplicacions j2ee.

Estudiarem les propostes mes conegudes i veurem quins son els punts que tenen en comú per tal de crear el nostre framework de treball per la capa de presentació.

Aquest framework es desenvoluparà en java per tal de donar-li al desenvolupador d'aplicacions una eina de fàcil configuració i implementació per aquesta capa i que resolgui la seva problemàtica implícita i només tingui que preocupar-se de les altres dos capes.

1.2 - Objectius del projecte:

Vaig escollir aquesta area de projecte per aprofundir millor sobre el desenvolupament d'aplicacions j2ee i poder desenvolupar millor el meu treball actual com a webmaster. Considero que es una de les àrees de projectes que esta en expansió i que necessita de bons especialistes al mercat de la informàtica.

Com a futur enginyer l'objectiu d'aquest projecte es situar-nos en una posició mes elevada al desenvolupament d'aplicacions i veure quins problemes es troba els desenvolupadors al crear una aplicació de client prim.

Ara dono un pas més fent una abstracció de quins problemes tenia com a desenvolupador d'aplicacions j2ee i com els podem resoldre de forma genèrica per facilitar el treball a altres desenvolupadors.

Concretament ens fixarem en la capa de presentació estudiant quins problemes tenim com a desenvolupador en aquesta capa i com els marcs de treball (frameworks) faciliten la seva resolució.

L'objectiu d'aquest projecte no es crear una eina innovadora que doni solucions al mercat informàtic ja que s'ha de tenir present que dins del mon java ja hi ha moltíssim frameworks que donen solucions a aquestes problemàtiques, si no d'estudi de quins problemes genèrics ens trobarem en la capa de presentació, conèixer com l'han resolt altres arquitectes de programari i crear la nostra pròpia solució.

Primer actuarem com a investigadors de les eines mes utilitzades del mercat i veurem quines son les característiques comuns a totes elles: quines problemàtiques es donen en la capa de presentació i com podem les podem resoldre.

Quan tinguem clar con resoldre aquest problemes, crearem una arquitectura que faciliti al desenvolupador d'aplicacions la creació de la capa de presentació d'una aplicació j2ee donant-li al desenvolupador d'aplicacions j2ee una eina (framework) per tal de que no es tingui que preocupar dels problemes derivats de la capa de presentació i pugui fixar-se en les altres capes

Després ens tornarem a posar en la pell d'un desenvolupador d'aplicacions j2ee i demostrarem amb la creació d'una aplicació concreta que s'ha facilitat el treball al desenvolupador al treballar amb el nostre framework.

1.3 - Planificació del projecte:

1.3.1 - Fites i lliuraments

El projecte del framework de presentació esta compost per quatre lliuraments en les següents dates:

PAC 1 – 14/03/2012
Objectius: Planificació del projecte i dels lliuraments parcials fins el lliurament dels productes finals Lliurables: Pla de treball – Planificació del projecte

PAC 2 – 19/04/2012
Objectius: Estudi del mercat dels frameworks j2ee de la capa de presentació mes utilitzats Lliurables: Document d'estudi de anàlisis de patrons de la capa de presentació i frameworks j2ee mes utilitzats al mercat.

PAC 3 – 04/06/2012

Objectius:
Disseny del framework presentació i implementació per lliurament al desenvolupador

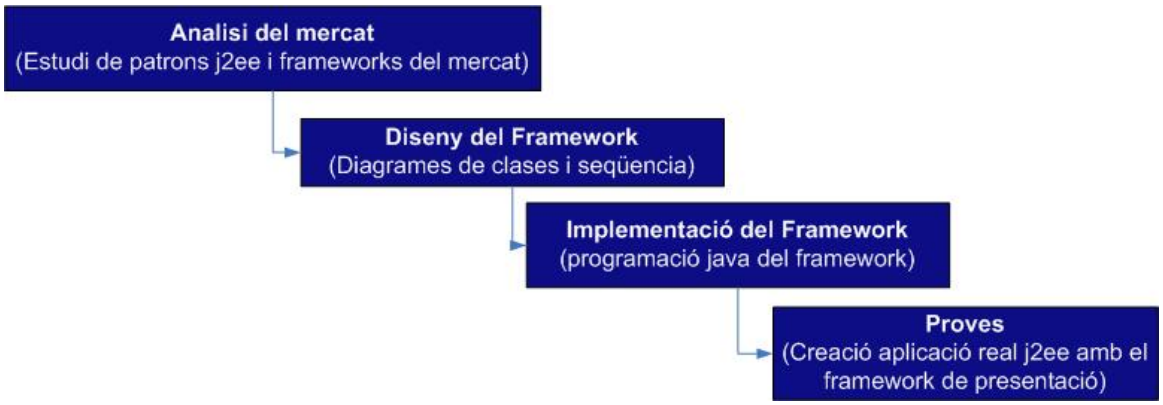
Lliurables:
-Document de disseny del framework
-Implementació java del framework

Lliurament final – 18/06/2012

Objectius:
Aplicació del framework en una aplicació real j2ee per tal de demostrar la facilitat d'us

Lliurables:
-Document de memòria del projecte
-Implementació d'una aplicació real amb el framework
-Presentació final

A fi d'arribar als objectius proposat en cada una de les fites he establert una planificació del projecte que ha de suposar avançar en les seves tasques de forma mes acurada. Aquesta planificació es lineal i seguint el següent model:



[figura 1: planificació lineal del projecte]

1.3.2 - Planificació de tasques i dates de lliurament

Al present document es detalla un diagrama de Gantt amb totes les tasques de la EDT i una planificació temporal de totes les fites i els lliuraments parcials que implicarà arribar el dia 18/06/2012 a la finalització del projecte PFC de “Disseny i implementació d’un marc de treball (framework) de presentació per aplicacions j2ee”.

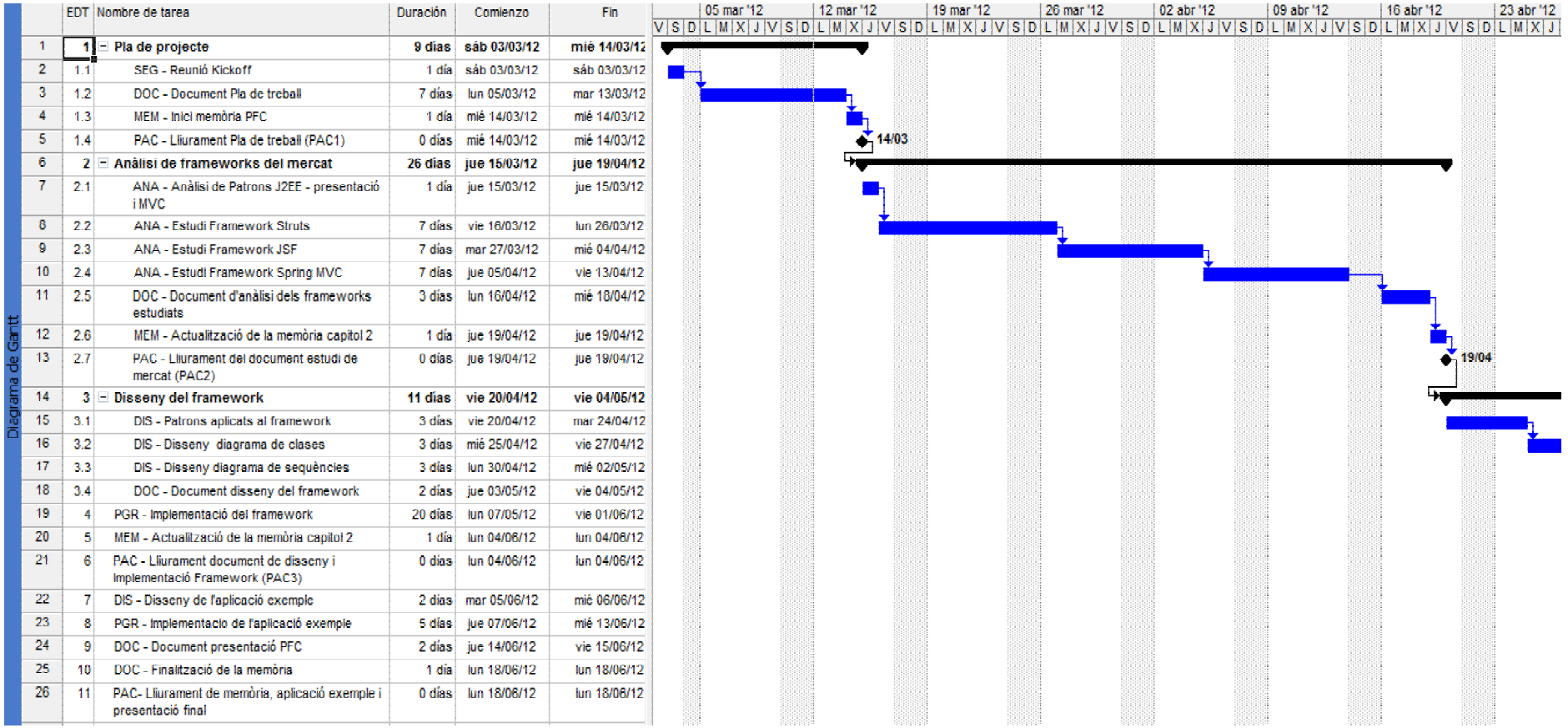
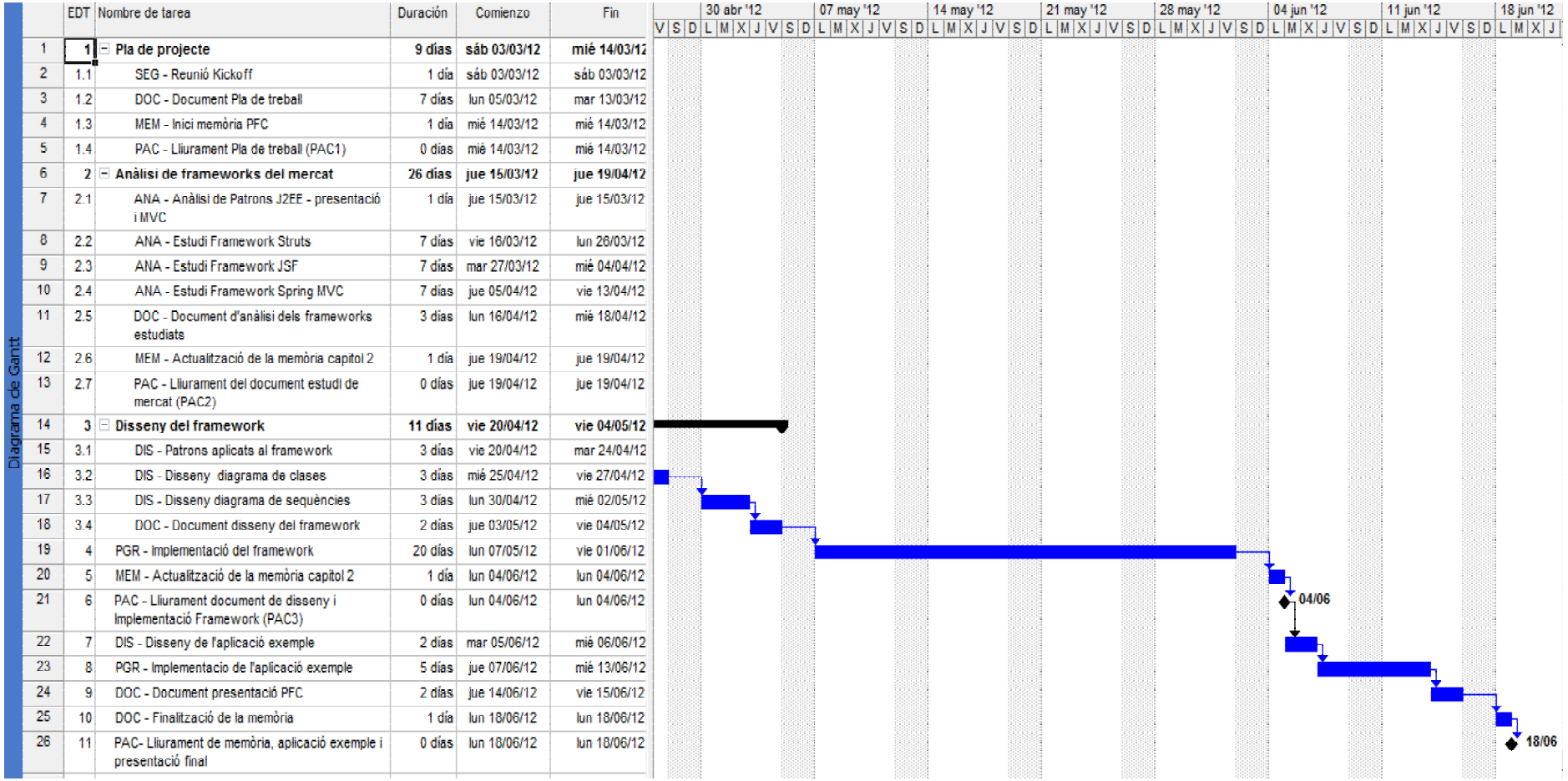


Diagrama de Gantt

	EDT	Nombre de tarea	Duración	Comienzo	Fin
1	1	**Pla de projecte**	**9 días**	**sáb 03/03/12**	**mié 14/03/12**
2	1.1	SEG - Reunió Kickoff	1 día	sáb 03/03/12	sáb 03/03/12
3	1.2	DOC - Document Pla de treball	7 días	lun 05/03/12	mar 13/03/12
4	1.3	MEM - Inici memòria PFC	1 día	mié 14/03/12	mié 14/03/12
5	1.4	PAC - Lliurament Pla de treball (PAC1)	0 días	mié 14/03/12	mié 14/03/12
6	2	**Anàlisi de frameworks del mercat**	**26 días**	**jue 15/03/12**	**jue 19/04/12**
7	2.1	ANA - Anàlisi de Patrons J2EE - presentació i MVC	1 día	jue 15/03/12	jue 15/03/12
8	2.2	ANA - Estudi Framework Struts	7 días	vie 16/03/12	lun 26/03/12
9	2.3	ANA - Estudi Framework JSF	7 días	mar 27/03/12	mié 04/04/12
10	2.4	ANA - Estudi Framework Spring MVC	7 días	jue 05/04/12	vie 13/04/12
11	2.5	DOC - Document d'anàlisi dels frameworks estudiats	3 días	lun 16/04/12	mié 18/04/12
12	2.6	MEM - Actualització de la memòria capitol 2	1 día	jue 19/04/12	jue 19/04/12
13	2.7	PAC - Lliurament del document estudi de mercat (PAC2)	0 días	jue 19/04/12	jue 19/04/12
14	3	**Disseny del framework**	**11 días**	**vie 20/04/12**	**vie 04/05/12**
15	3.1	DIS - Patrons aplicats al framework	3 días	vie 20/04/12	mar 24/04/12
16	3.2	DIS - Disseny diagrama de clases	3 días	mié 25/04/12	vie 27/04/12
17	3.3	DIS - Disseny diagrama de seqüències	3 días	lun 30/04/12	mié 02/05/12
18	3.4	DOC - Document disseny del framework	2 días	jue 03/05/12	vie 04/05/12
19	4	PGR - Implementació del framework	20 días	lun 07/05/12	vie 01/06/12
20	5	MEM - Actualització de la memòria capitol 2	1 día	lun 04/06/12	lun 04/06/12
21	6	PAC - Lliurament document de disseny i implementació Framework (PAC3)	0 días	lun 04/06/12	lun 04/06/12
22	7	DIS - Disseny de l'aplicació exemple	2 días	mar 05/06/12	mié 06/06/12
23	8	PGR - Implementacio de l'aplicació exemple	5 días	jue 07/06/12	mié 13/06/12
24	9	DOC - Document presentació PFC	2 días	jue 14/06/12	vie 15/06/12
25	10	DOC - Finalització de la memòria	1 día	lun 18/06/12	lun 18/06/12
26	11	PAC- Lliurament de memòria, aplicació exemple i presentació final	0 días	lun 18/06/12	lun 18/06/12



	EDT	Nombre de tarea	Duración	Comienzo	Fin
1	1	**Pla de projecte**	**9 días**	**sáb 03/03/12**	**mié 14/03/12**
2	1.1	SEG - Reunió Kickoff	1 día	sáb 03/03/12	sáb 03/03/12
3	1.2	DOC - Document Pla de treball	7 días	lun 05/03/12	mar 13/03/12
4	1.3	MEM - Inici memòria PFC	1 día	mié 14/03/12	mié 14/03/12
5	1.4	PAC - Lliurament Pla de treball (PAC1)	0 días	mié 14/03/12	mié 14/03/12
6	2	**Anàlisi de frameworks del mercat**	**26 días**	**jue 15/03/12**	**jue 19/04/12**
7	2.1	ANA - Anàlisi de Patrons J2EE - presentació i MVC	1 día	jue 15/03/12	jue 15/03/12
8	2.2	ANA - Estudi Framework Struts	7 días	vie 16/03/12	lun 26/03/12
9	2.3	ANA - Estudi Framework JSF	7 días	mar 27/03/12	mié 04/04/12
10	2.4	ANA - Estudi Framework Spring MVC	7 días	jue 05/04/12	vie 13/04/12
11	2.5	DOC - Document d'anàlisi dels frameworks estudiats	3 días	lun 16/04/12	mié 18/04/12
12	2.6	MEM - Actualització de la memòria capitol 2	1 día	jue 19/04/12	jue 19/04/12
13	2.7	PAC - Lliurament del document estudi de mercat (PAC2)	0 días	jue 19/04/12	jue 19/04/12
14	3	**Disseny del framework**	**11 días**	**vie 20/04/12**	**vie 04/05/12**
15	3.1	DIS - Patrons aplicats al framework	3 días	vie 20/04/12	mar 24/04/12
16	3.2	DIS - Disseny diagrama de clases	3 días	mié 25/04/12	vie 27/04/12
17	3.3	DIS - Disseny diagrama de sequències	3 días	lun 30/04/12	mié 02/05/12
18	3.4	DOC - Document disseny del framework	2 días	jue 03/05/12	vie 04/05/12
19	4	PGR - Implementació del framework	20 días	lun 07/05/12	vie 01/06/12
20	5	MEM - Actualització de la memòria capitol 2	1 día	lun 04/06/12	lun 04/06/12
21	6	PAC - Lliurament document de disseny i Implementació Framework (PAC3)	0 días	lun 04/06/12	lun 04/06/12
22	7	DIS - Disseny de l'aplicació exemple	2 días	mar 05/06/12	mié 06/06/12
23	8	PGR - Implementacio de l'aplicació exemple	5 días	jue 07/06/12	mié 13/06/12
24	9	DOC - Document presentació PFC	2 días	jue 14/06/12	vie 15/06/12
25	10	DOC - Finalització de la memòria	1 día	lun 18/06/12	lun 18/06/12
26	11	PAC- Lliurament de memòria, aplicació exemple i presentació final	0 días	lun 18/06/12	lun 18/06/12

[figura 2: planificació temporal del projecte]

1.3.3 - Diccionari EDT

Finalment s'informa de les tasques que s'han especificat al diagrama de Gantt, la seva durada i els resultats que s'obtindran després de la seva resolució.

1. Pla de projecte	
Descripció: Defineix el projecte, els seus objectius i planificació fins la seva finalització. Duració: 9 Dies	Resultats: Document pla de treball on es planifica tot el treball a fer.

1.1 SEG - Reunió kickoff	
Descripció: Introducció dels continguts de l'àrea de PFC genèriques i primera tutoria amb el consultor per aclariments de quina finalitat te el pfc de j2ee Duració: 1 Dia	Resultats: Determinar la finalitat i l'abast del projecte.

1.2 DOC – Document del pla de treball	
Descripció: Documentació - Documentar la planificació del projecte pfc-j2ee Duració: 7 Dies	Resultats: Documentar la finalitat i planificació del projecte.

1.3 MEM –Inici de memòria pfc	
Descripció: Memòria – Iniciar la memòria complimentant-la amb el document pla de treball Duració: 1 Dia	Resultats: Memòria amb el capítol 1 documentat.

2 ANA – Anàlisi dels frameworks del mercat	
Descripció: ANA – Anàlisis de patrons j2ee i dels frameworks de presentació j2ee mes utilitzats al mercat informàtic Duració: 26Dies	Resultats: Document d'anàlisi de mercat.

2.1 ANA –Anàlisi de Patrons J2EE - Presentació i MVC	
Descripció: ANA – Anàlisis de patrons j2ee: MVC, frontcontroller, Intercepting filter... Duració: 1Dia	Resultats: 1er punt del document d'anàlisi de mercat.

2.2 ANA - Estudi Framework Struts	
Descripció: ANA – Anàlisi de framework struts: arquitectura i seqüència de peticions – respostes del framework Duració: 7Dies	Resultats: 2º punt del document d'anàlisi de mercat.

2.3 ANA - Estudi Framework JSF	
Descripció: ANA – Anàlisi de framework JSF: arquitectura i seqüència de peticions – respostes del framework Duració: 7Dies	Resultats: 3º punt del document d'anàlisi de mercat.

2.4 ANA - Estudi Framework Spring MVC	
Descripció: ANA – Anàlisi de framework Spring MVC: arquitectura i seqüència de peticions – respostes. Duració: 7Dies	Resultats: 4º punt del document d'anàlisi de mercat.

2.5 DOC - Document d'anàlisi dels frameworks estudiats	
Descripció: DOC – Documentar els 3 punts anteriors posar en coneixement les característiques comuns a tots els frameworks Duració: 3Dies	Resultats: Finalització document d'anàlisi de mercat amb conclusions finals de característiques utilitzades per tots tres

2.6 MEM - Actualització de la memòria capítol 2	
Descripció: MEM – Actualitzar la memòria amb el document d'anàlisi de mercat. Duració: 1Dia	Resultats: Memòria actualitzada amb el document de anàlisi de mercat

3 Disseny del framework	
Descripció: Disseny del framework en funció de les característiques trobades al document d'anàlisi de mercat. Duració: 11Dies	Resultats: Document de disseny del framework propi.

3.1 DIS - Patrons aplicats al framework	
Descripció: DIS- Disseny de l'aplicació dels patrons estudiats al punt 2.1 al nostre framework. Duració: 3Dies	Resultats: Punt 1 del document de disseny del framework

3.2 DIS – Disseny diagrama de classes	
Descripció: DIS- Disseny del diagrama de classes del framework Duració: 3Dies	Resultats: Punt 2 del document de disseny del framework

3.3 DIS - Disseny diagrama de seqüències	
Descripció: DIS- Disseny del diagrama de seqüències de inicialització i petició - resposta del framework Duració: 3Dies	Resultats: Punt 3 del document de disseny del framework

3.4 DOC – Document disseny del framework	
Descripció: DOC- Documentació de tots els punts anteriors en un document lliurable al consultor Duració: 2Dies	Resultats: Document lliurable de disseny del framework propi

4 PGR - Implementació del Framework	
Descripció: PGR- Programació java del framework Duració: 20Dies	Resultats: jar empaquetat amb les classes del framework i codi font.

5 MEM - Actualització de la memòria capítol 2	
Descripció: MEM – Actualització de la memòria amb el disseny del framework Duració: 1Dia	Resultats: Capítol 2 de la memòria actualitzat.

7 DIS – Disseny de l'aplicació exemple	
Descripció: DIS – Disseny de l'aplicació d'exemple que utilitzarà el framework Duració: 2Dies	Resultats: Document de disseny de l'aplicació exemple

8 PGR – Implementació de l'aplicació exemple	
Descripció: PGR – Programació de l'aplicació exemple que utilitzarà el framework (proves del framework) Duració: 5Dies	Resultats: Fitxer war del web exemple i codi font

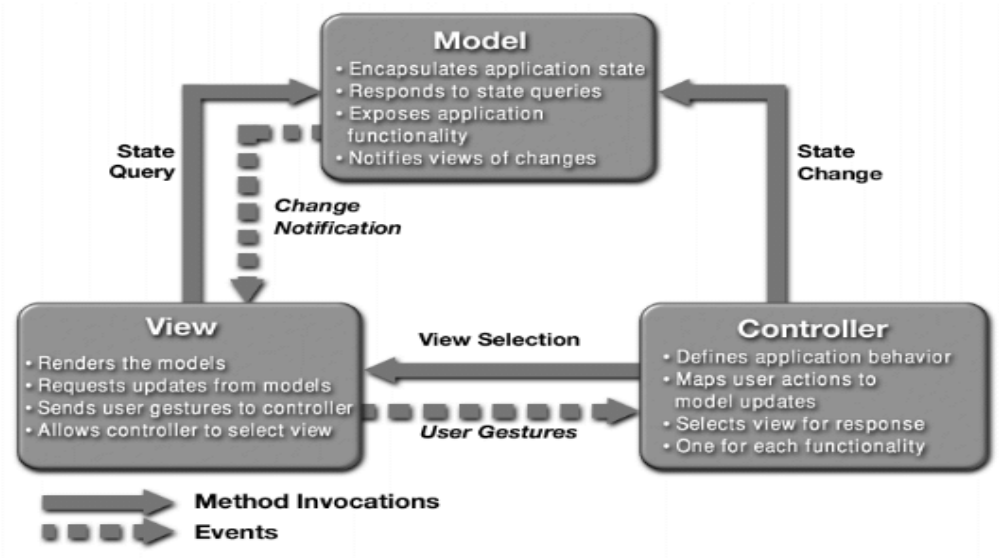
9 DOC - Document presentació PFC	
Descripció: DOC – Elaboració de la presentació de PFC Duració: 2Dies	Resultats: Lliurament de la presentació de PFC

10 DOC - Finalització de la Memòria	
Descripció: DOC – Finalitzar el document de la memòria Duració: 1Dia	Resultats: Lliurament de la memòria de PFC

2.- Anàlisis de patrons de la capa de presentació

2.1 - El patró MVC – patró de disseny:

El Patró MVC (Model Vista Controlador) es un dels mes utilitzats en aplicació j2ee. Aquest patró tracta de separar la lògica de negoci de l'aplicació (model) de la presentació de les dades (vista) que es presentarà al usuari com a resposta a les seves peticions. Aquestes peticions es centralitzen sobre la figura del controlador.



[figura 3: patró model vista controlador]

El MVC té tres components:

- Controlador: Centralitza les peticions de les vistes traduint-les a accions per donar-les al model i que aquest les pugui executar.
- Model:Representa les dades de la empresa i les regles de negoci que donen accés a les dades.
- Vista: Son la representació dels continguts donats per el model.

Estratègies d'implementació:

- Clients basat en web: utilitza pagines JSP per la vista, Servlets per el controlador y EJB para desenvolupar el model
- Controlador centralitzat: El patró frontcontroller, que estudiarem després desenvolupa aquesta estratègia
- Clients sense fils.

Conseqüències de la seva utilització:

- Reutilització del model: la utilització de diferents vistes amb el mateix model facilita el manteniment del model i la reutilització de codi.
- Fàcil suport a altres tipus de clients: recodificant el controlador i les vistes podem utilitzar el mateix model.
- Incrementa la complexitat del disseny.

2.2 - Patrons J2EE- capa de presentació:

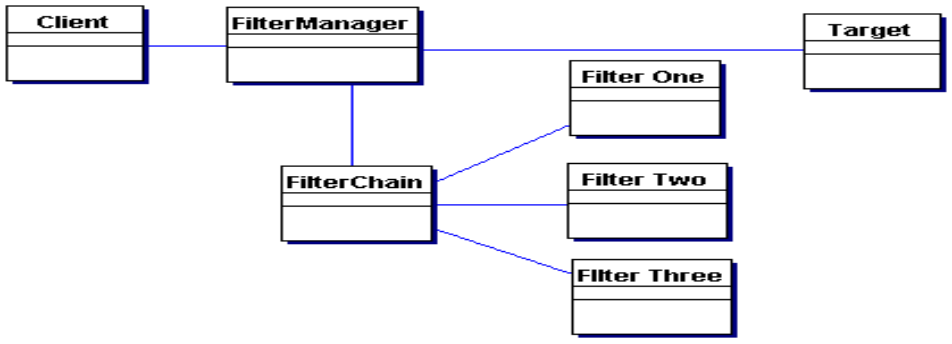
2.2.1- Intecepting Filter:

Aquest patró permet pre-processar i post-processar les peticions que fa un client web per tal de aplicar-li accions com podria ser:Validacions sobre IPs que poden accedir a l'aplicació, autentificacions de clients, violacions sobre el path de les peticions, controls sobre el navegador del client...

Aquestes validacions es podrien programar com una sèrie de "ifs/else" condicionals encadenats però implicaria el manteniment constant del codi.

Amb aquest controlador les peticions que fa el client sobre un target sempre passen per el filterManager que crea una col·lecció ordenada de filtres denominada FilterChain. Aquest filterChain es el l'encarregat de passar de forma ordenada per cada un dels filtres especificats afectant a la petició (Filter one, Filter Two, Filter Three).

De manera que si volem crear un nou pre-procesament de la petició només tenim de programar el filtre específic.



[figura 4: patró intercepting filter]

Estratègies d'implementació:

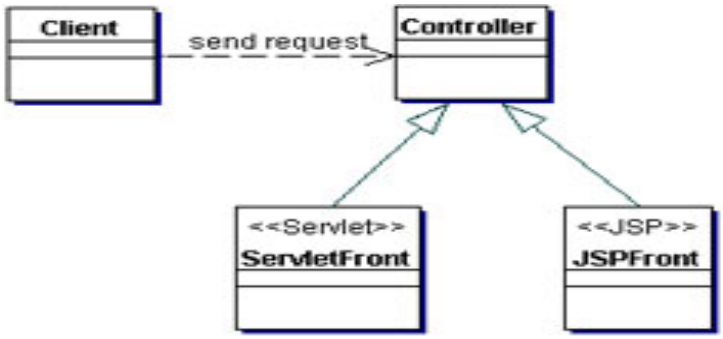
- Custom Filter: estratègia personalitzada per el desenvolupador i poc flexible ja que no proporciona embolcall ni a la resquest ni a la response.
- Standard Filter Strategy: son filtres construïts sobre interfícies i que s'afegeixen o s'eliminen de forma declarativa en el descriptor d'aplicacions de l'aplicació
- Base Filter Strategy: Es una superclasse comú a tots els filtres implementats per tal de tenir la funcionalitat bàsica.
- Template Filter: Plantilla filtre que permet implementar la plantilla de mètodes que tindrà el filtre.

Conseqüències de la seva utilització:

- Proporcionen un lloc de centralització per controlar el processament de les peticions de la mateixa manera que ho fa el controlador.
- Millora la reutilització ja que s'afegeixen i s'eliminen de forma transparent amb una configuració flexible sense tenir que recompilar el codi.

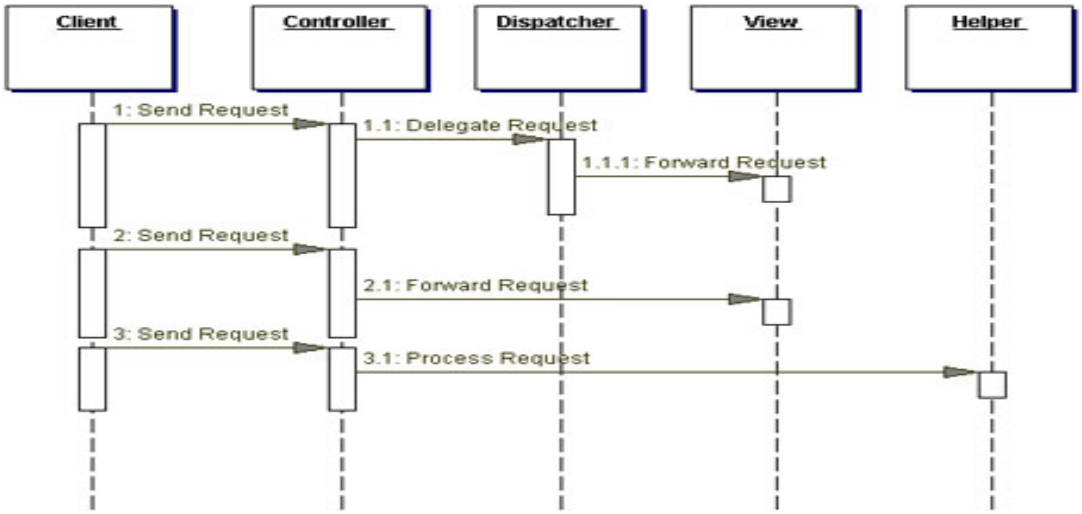
2.2.2 - Front Controller:

Centralitza les peticions de la capa de presentació per dona suport a la integració de serveis, la recuperació de continguts del model i el control de vistes i navegació permetent la reutilització de codi entre peticions



[figura 5: diagrama de classes del patró front controller]

Quan el controller rep la petició del client podria delegar a un helper per fer tasques d'autenticació. La petició es delega al dispatcher que es l'encarregat de buscar la vista i la navegació. Aquesta vista amb el seu model es representada per la view que es la que genera la resposta a la petició



[figura 6: diagrama de seqüències del patró front controller]

Estratègies d'implementació:

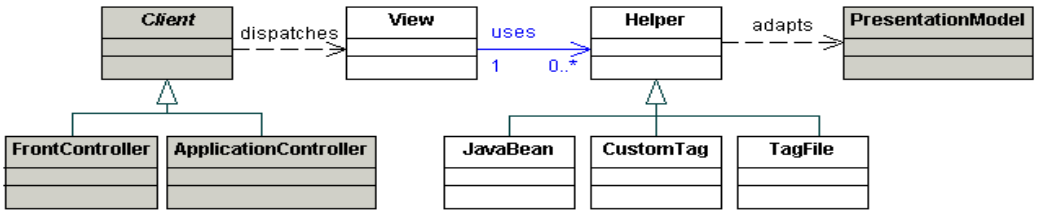
- Servlet Front: defineix el controlador com un Servlet
- JSP Front: defineix el controlador com una pagina JSP
- Comand and controler: basat en el patró de disseny "Comand" dona una interfície genèrica per els components "helper" delegant-li la responsabilitat i minimitzant l'acoblament.
- Physical Resource Mapping: les peticions es fan sobre noms físics de recursos
- Logical Resource Mapping: les peticions es fan sobre noms lògics de recursos en lloc de l'anterior.
- Multiplexed Resource Mapping : les peticions son sobre noms lògics però fa un mapeig per nom lògic a tot els seus recursos.
- Dispatcher in controller: s'utilitza si el dispatcher es mínim acoblant-lo amb el controlador
- Base Front: defineix una base controladora de la que han d'estendre la resta de controladors
- Filter Controller: implementat com un filtre (explicat anteriorment)

Conseqüències de la seva utilització:

- Centralitza el control de les peticions i millora la seguretat d'intents il·lícits de accessos a l'aplicació
- Millora reutilització de codi donant una implementació neta.

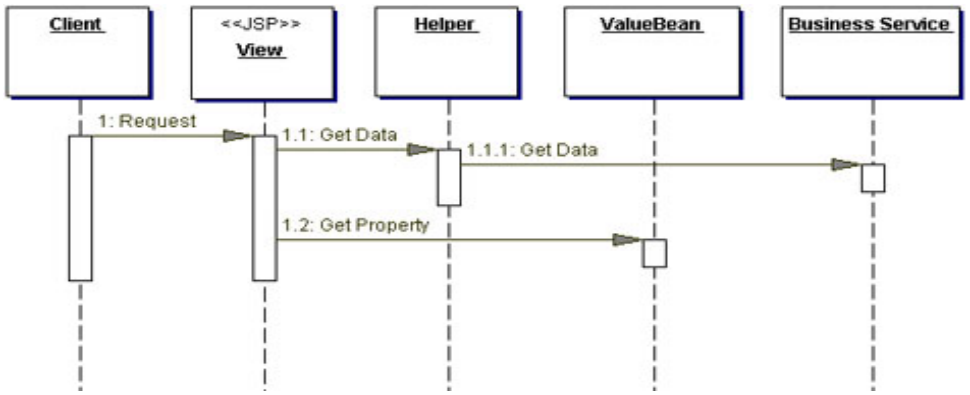
2.2.3 - View Helper:

El ViewHelper serveis per que les vistes del MVC tractin les dades del model delegant-les als helpers que li ajudaran a fer la presentació i d'aquesta manera desacoblar el disseny de la programació necessària per mostrar les dades del dona el model.
Les vistes tenen codi formatejat (javabeans o etiquetes personalitzades) per donar la responsabilitat del codi als helpers que son les que ajudaran a muntar la presentació.



[figura 7: diagrama de classes del patró view helper]

Es tracta d'un controlador fet a mida entre el client i la vista(presentació) que normalment es un Javabean i que pot demanar operacions sobre beans entitat (valueBean) o sobre objectes de negoci(Bussiness Service)



[figura 8: diagrama de seqüències del patró view helper]

Estratègies d'implementació:

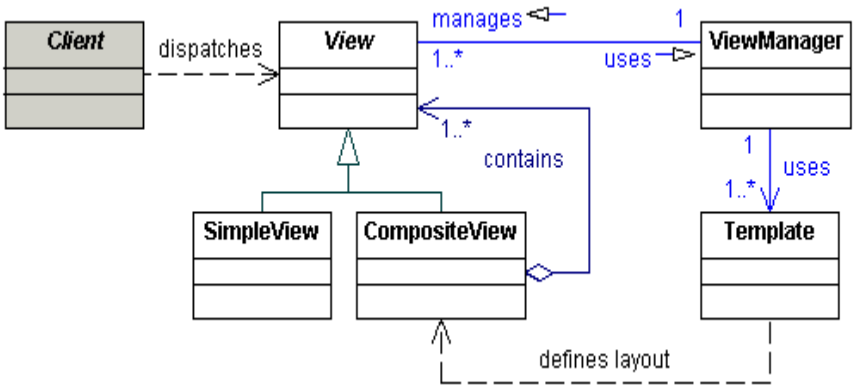
- JSP View: Defineix les vistes com a pàgines jsp i es perfecte per desacoblar el treball dels programadors dels dissenyadors
- Servlet View: Defineix les vistes com a servlets complicant el codi ja que embebeix les etiquetes de marca dins del codi java.
- Javabean Helper: El helper es un javabean facilitant la integració en entorns jsp i facilitat la comprensió del codi
- Custom Tag Helper: S'implementa com una etiqueta personalitzada amb l'inconvenient de la utilització de molts artefactes accessoris com son els descriptors de etiquetes, els fitxers de configuració i la pròpia etiqueta.
- Bussines Delegate as a Helper: S'implementa amb un delegat de negoci que implementa el codi a la capa especifica de negoci.
- Transformer Helper: S'implementa com un fitxer xslt que esta relacionat amb el fitxer xml de les dades del model.

Conseqüències de la seva utilització:

- Separació de roles de dissenyadors i programadors.
- Amb la no utilització de scriptlets al codi i la incorporació del helper es millora la separació entre model i vista i la seva encapsulació.

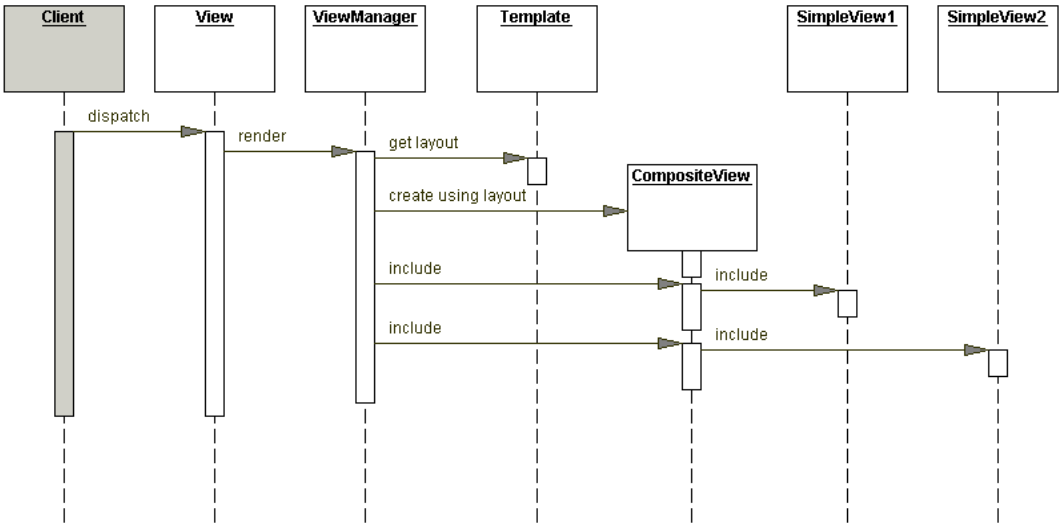
2.2.4 - Composite View:

Es útil per tal de desagregar la vista en subvistes reutilitzables, no duplicar codi i poder crear un disseny modular de la vista.
Permet la creacions de dissenys pedra-cartro amb el muntatge modular de cada subvista.



[figura 9: diagrama de classes del patró view]

El controlador de vistes maneja la inclusió de porcions de plantilla a la vista composta. Aquestes porcions o peces atòmiques també podrien ser una altra vista composta.



[figura 9: diagrama de seqüències del patró view]

Estratègies d'implementació:

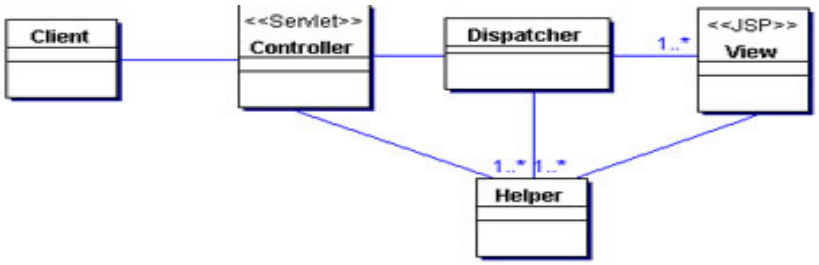
- Jsp Page View (ver patró view helper)
- Servlet View (ver partó view helper)
- Javabean View Management:el controlador de la vista es un javabean que s'encarrega de la distribució i la composició de la vista.
- Standard Tag View Management: el controlador de la vista son etiquetes JSP estàndards com <jsp:include>
- Custom Tag View Management: La lògica implementada a la etiqueta controla al composició i la distribució de la vista.
- Transformer View Management: S'implementa utilitzant un transformador xsl.

Conseqüències de la seva utilització:

- Millora la modularitat de la vista i reutilització
- Facilitat el manteniment ja que es mes fàcil fer canvis en porcions de la plantilla.
- Millora la flexibilitat facilitat la inclusió de les vistes en temps d'execució però te impacte al rendiment.

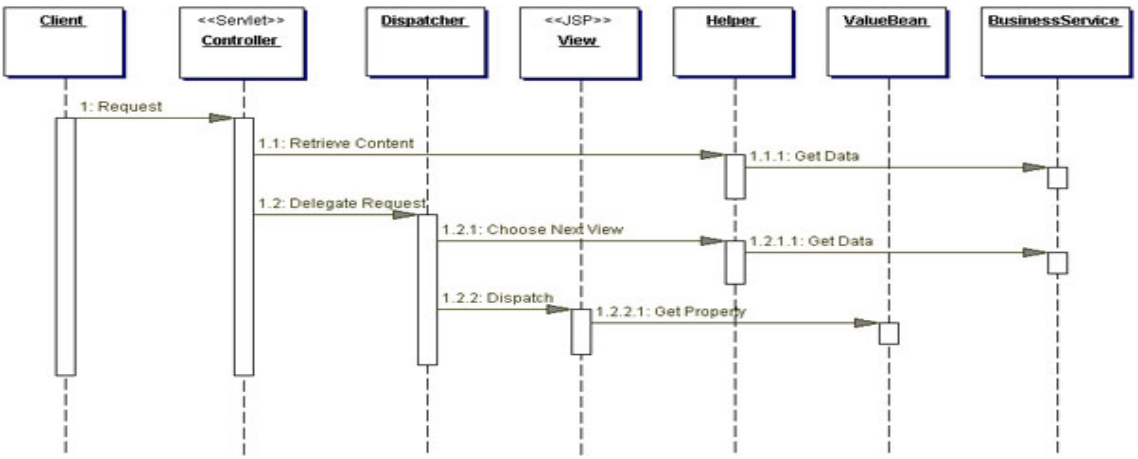
2.2.5 - Service To Worker:

Descriu la combinació del patró frontController i viewHelper amb un component dispatcher. El “Helper” es l’encarregat de la recuperació del contingut mentre que el “Dispatcher” es el responsable del control de la vista i de la navegació. Aquest dispatcher pot encapsular-se dins del controlador o en un component separat.



[figura 10: diagrama de classes del patró service to worker]

El punt de partida es la petició d’un client sobre el controller. Aquest delega sobre el dispatcher el control i navegació de la vista.
Quant mes responsabilitat tingui aquest dispatcher mes s’aproxima sobre a aquest patró i quant menys responsabilitat tingui mes s’aproxima al patró “Dispatcher View”. El “Helper” ajuda a la vista o al controlador a completar el processament de la petició.



[figura 11: diagrama de seqüències del patró service to worker]

Estratègies d'implementació:

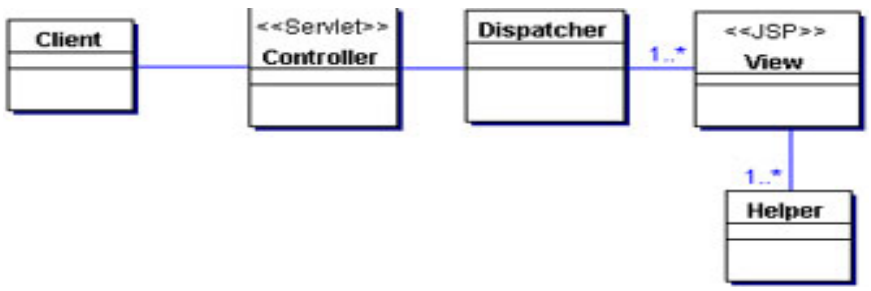
- Servlet Front o JSP Front, Dispatcher in Controller (del patró Front Controller)
- Servlet View, JavaBean Helper, Custom Tag Helper, Transformer Helper (del patró View Helper)

Conseqüències de la seva utilització:

- Centralitza el control de les peticions i millora la reutilització
- Millora la separació de rols dissenyadors / desenvolupadors
- Els "Helpers" milloren la separació entre la vista i el processament de la lògica de negoci

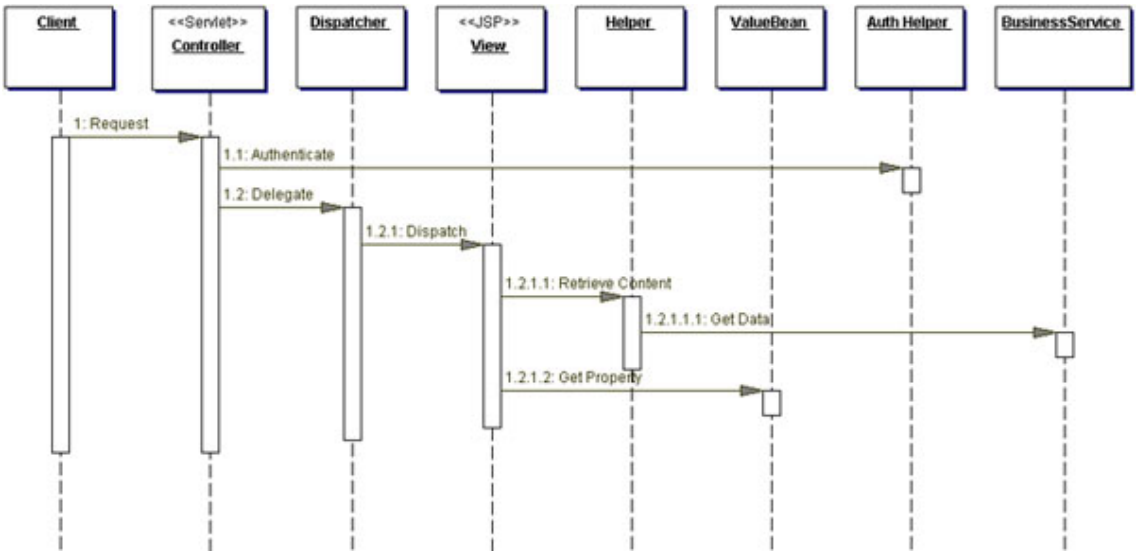
2.2.6 - Dispatcher View:

Amb la mateixa intenció que l'anterior patró, però amb un dispatcher que te un rol moderat en el control de la vista com podria ser la simple traducció de l'acció en la vista a mostrar sense tenir que accedir a objectes de negoci.



[figura 12: diagrama de classes del patró dispatcher view]

En aquest cas el dispatcher ja no fa crides als objectes de negoci, sinó que es la pròpia vista la que s'ajuda del "Helper".



[figura 13: diagrama de classes del patró dispatcher view]

Estratègies d'implementació:

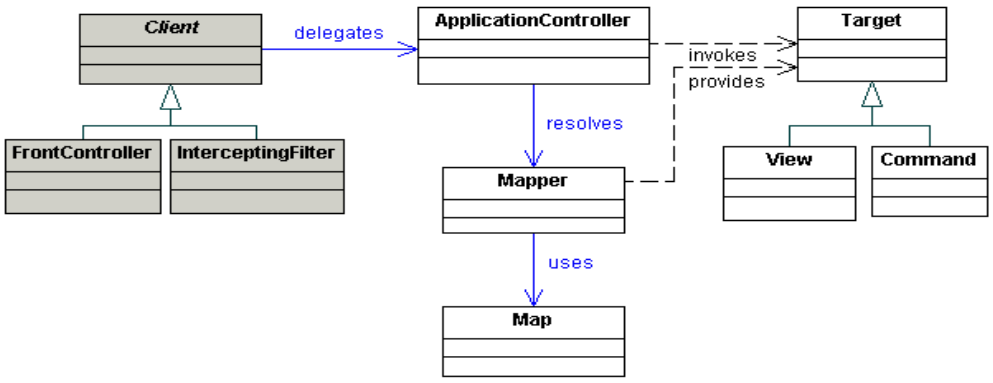
- Servlet Front o JSP Front, Dispatcher in Controller (del patró Front Controller)
- Servlet view, JavaBean Helper, Custom Tag Helper (del patró View Helper)

Conseqüències de la seva utilització:

- Centralitza el control de les peticions i millora la reutilització
- Millora la separació de rols dissenyadors / desenvolupadors
- Els "helpers" milloren la separació entre la vista i el processament de la lògica de negoci

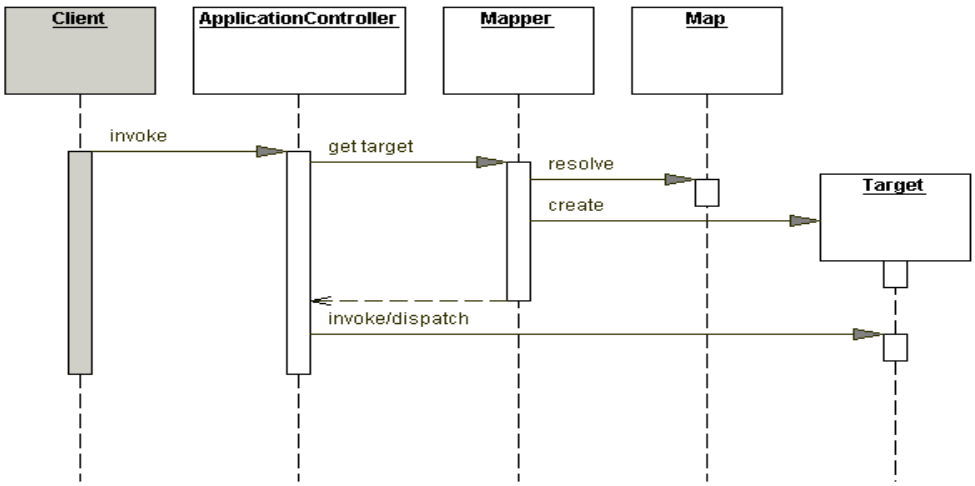
2.2.7 - Aplication Controller:

Un dels patrons nous que van aparèixer en la segona edició del “Core J2ee Patterns” ha estat el “Aplicacion Controller” on queda clar que el controller delega la responsabilitat sobre aquest nou component que s’encarrega ara de la gestió de les peticions i la resolució de la vista, deixant el front controller mes lleuger.



[figura 14: diagrama de classes del patró aplication controller]

La petició que rep el controller del client la transmet al control “aplicattion controller”. Aquest control que prèviament ha muntat un “Mapper” de les possibles accions-respostes interroga aquest “Mapper” per mapejar la petició i la resposta i poder trobar el “target” a crear. Amb aquest “target” (acció) el “AplicationController” invoca la acció del model i fa el “dispatch” de la vista.

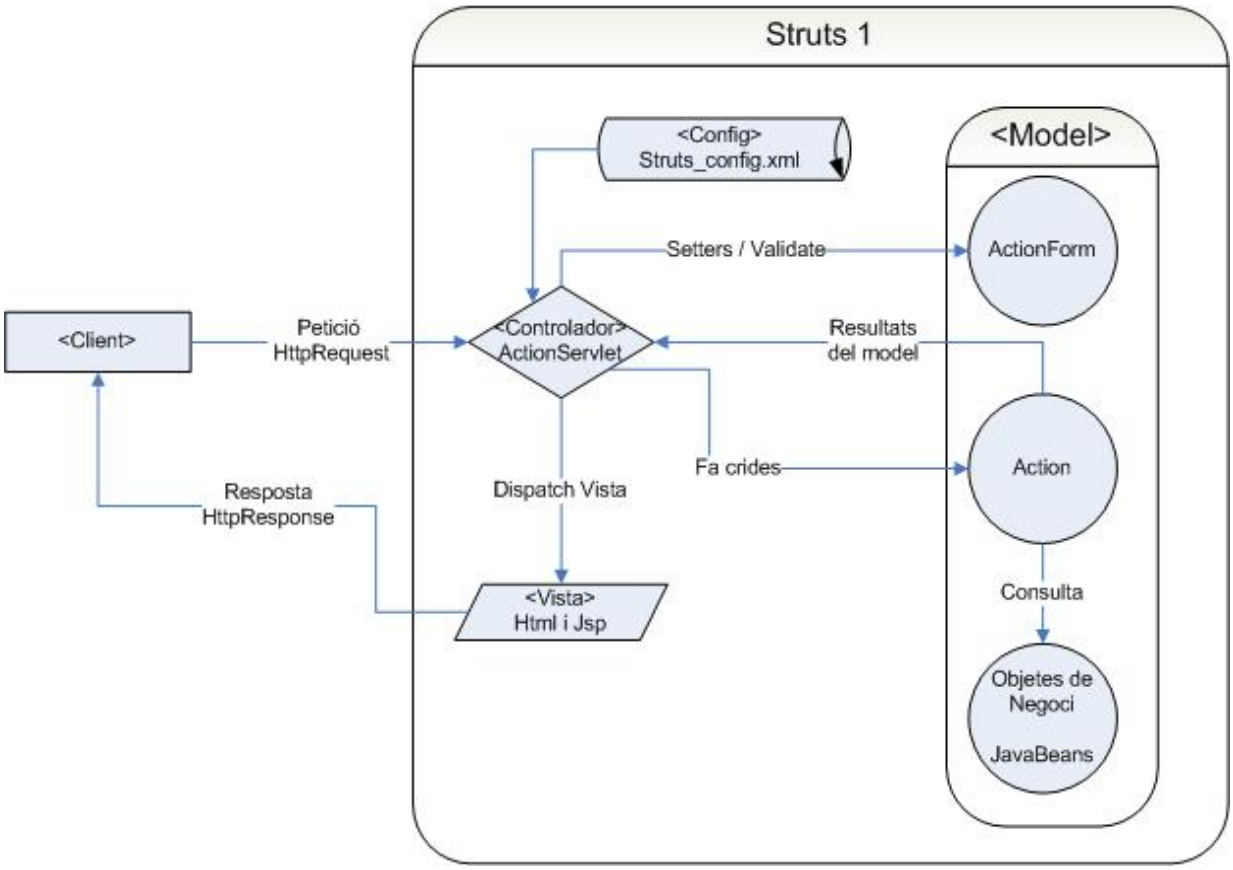


[figura 15: diagrama de seqüències del patró aplication controller]

3.- Estudi del framework de presentació Struts

3.1- Struts 1:

Struts es un framework desenvolupat per Apache Software Fundation basat en un MVC on tenim els següents components:



[figura 16: diagrama de petició - resposta de Struts 1]

- Controlador: Esta basat en un patró “froncontroler amb estratègia Servletfront” denominat ActionServlet que distribueix les peticions dels client web al dispatcher. Aquestes peticions arriben al ActionServlet amb la extensió *.do.

Per fer el dispatcher de la petició “ActionServlet” invoca les accions demanades per el client mitjançant les classes “Action” aplicant un patró “Command”.

Aquestes accions es defineixen prèviament al fitxer de configuració del framework denominat struts_config.xml. El framework al crear el Servlet llegeix aquest fitxer i fa un mapejat de les accions a través de classes “ActionMaping” guardant-les sobre un contenidor denominat ”ActionMapper”.

El programador que utilitzi aquest framework s’ha de preocupar d’implementar accions que heretin de la classe “Action” i configurar-les al fitxer struts_config.xml.

En la configuració del fitxer struts_config.xml es decideix per quin path respondrà aquesta acció i quina serà la vista a mostrar de l’acció (forward).

- Model: Les Accions programades per el programador que son subclasses de “Action” son las que a través d’un patró “Facade” faran les crides als objectes de negoci.

S’utilitzen també les classes ActionForm per ajudar al programador en la comunicació de la vista amb la model . Aquestes subclasses de la classe “ActionForm” s’han de configurar amb el paràmetre “name” dins de les accions del fitxer de configuració i son les que amb getters i setters dels camps de la vista donen un helper al programador.

- Vista: Son implementades amb pagines html i jsp.

Struts també dona una llibreria d’etiquetes per tal de que el programador no implementi scriptles i pugui separar clarament el disseny de la vista de la programació del retorn del model.

3.2 - Struts 2:

En la segona versió de Struts el projecte de Apache es fusiona amb webwork2 donant un framework mes flexible i mes modern en la seva configuració. Entre les millores trobades apuntem les següents:

- Utilització de anotacions java que comencen a funcionar a partir de la versió java5 per tal de crear les accions i per tant ja no s'han de configurar al fitxer de configuració. Es redueix la configuració d'accions.

- Major desacoblament de les accions ja que ara son pojos que no necessiten d'heretar sobre classes abstractes. La implementació d'aquest pojos es fan a través de les interfícies de anotacions i no a través de l'herència de les classes Action.

  S'utilitza tècniques de Java Reflection per tal de que el framework pugi instanciar i executar aquestes accions en temps d'execució.

- Aquestes accions també poden utilitzar javabeans per fer les seves validacions i per tan desapareixen també els ActionsForms de la versió anterior.

- Una configuració mes senzilla del framework amb la partició del fitxer en mes d'un per facilitar la tasca de configuració al programador.

- Utilització de plantilles, templates i temes facilitat la implementació del patró "composite view" amb la possible utilització de llibreries externes com freemaker o velocity a mes a mes de pagines jsp combinades amb struts-tiles (continuant amb la seva versió del composite view del struts1).

- Defineix comportaments per defecte facilitant la seva configuració:
  La configuració del framework a través del seu fitxer struts.xml pot ser per tres vies:

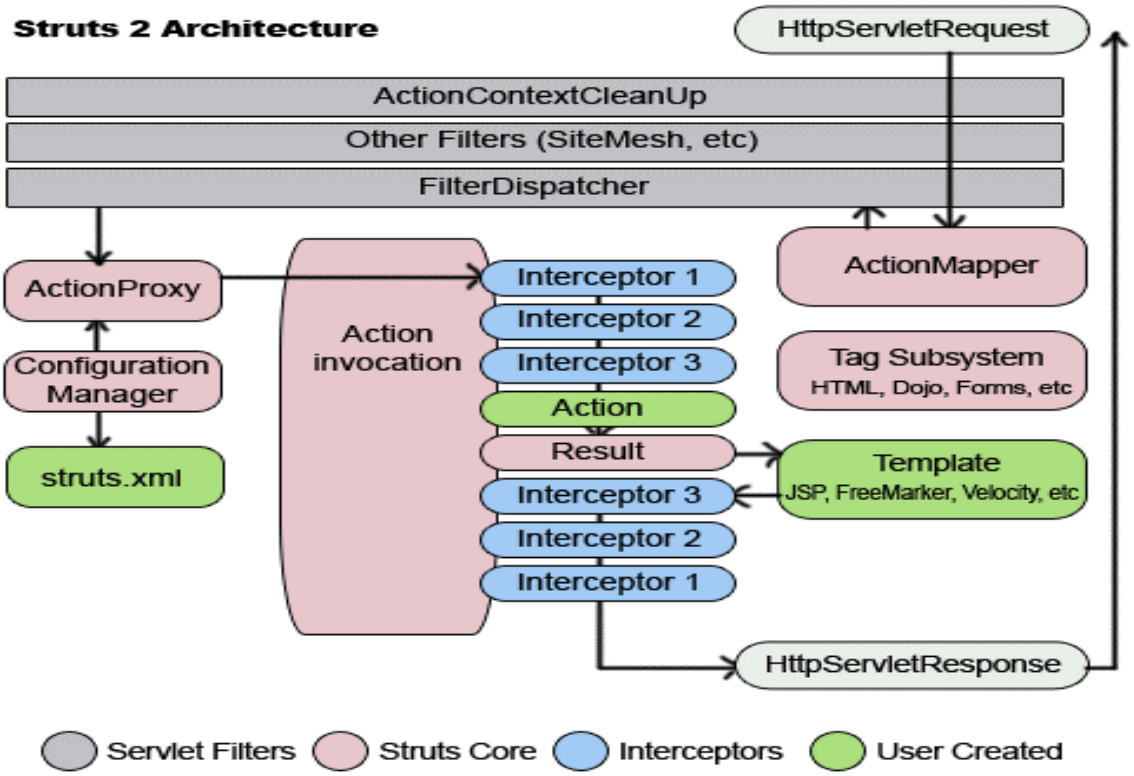
  1- configurant les accions i els forwards segons el model de struts1 mantenint la seva compatibilitat.

  2- Amb convencions de nom: deixant que struts defineixi per defecte la URI i nom de la vista que tindrà cada acció en funció del seu nom de classe.
  Per exemple: la acció de "llistarempleats" estarà definida amb la url /llistarempleats i amb la vista llistarempleats.jsp per defecte.

  3- Evitar la configuració de les accions al fitxer struts.xml per tal de les accions siguin implementades amb anotacions.

En struts2 les peticions arriben a un nou controlador (el filterDispatcher) implementat per un filtre estàndard que busca sobre el ActionMapper si s'ha d'executar una acció. Les peticions son configurades al fitxer de configuració amb la extensió *.action.

Struts2 també implementa el patró "filter intercepting" amb el ActionContextCleanUp i altres possibles filtres de l'aplicació per tal de facilitar la integració amb altres tecnologies mitjançant l'aplicació de filtres en el seu fitxer de configuració.



[figura 17: diagrama de petició - resposta de struts 2]

En cas positiu d'execució d'una acció el "filterDispatcher" delega al ActionProxy la invocació de l'acció a executar.

El ActionProxy que configura el framework amb els fitxers de configuració en la seva inicialització, crea la classe “Action invocation” i seguint el patró “Command” invoca l’acció utilitzant el mètode execute() que ha de tenir obligatòriament el pojo que implementa l’acció.

Aquest mètode no rep cap paràmetre i per tant si a l’acció se li ha d’enviar paràmetres s’utilitza la tècnica de injecció de dependencies amb Setters .

Tant prèviament com a posteriori a la execució de l’acció es poden configurar i utilitzar interceptors per validacions, per controlar el flux de la petició, per pujar fitxers, etc. Aquest interceptors que estan en una pila s’apliquen en un ordre determinat en la preexecució i a la inversa en la postexecució de la acció com queda reflectit al gràfic.

Finalment el resultat de l’acció (objecte Result), que potser modelat amb templates es retornat al client a través la httpServletResponse.

Aquest objecte Result es retornat amb la finalitat de permetre l’execució no nomes de jsps com a vista si no qualsevol pluging implementat (p. ex correu electrònic, freemaker, velocity...)

3.3 - Característiques diferenciables de Struts:

- La configuració de les accions del framework mitjançant anotacions java5 sense necessitat de configuració amb fitxers xml com es feia en la primera versió.

- L’objecte Result: Implementació de les vistes amb jsps amb la possibilitat de poder utilitzar suports per la generació de vistes com struts tiles, o productes externs com velocity o freemaker.

- L’ús de una llibreria pròpia taglib (“Struts 2 UI Tags”) que facilita la comunicació del model amb la vista per tal de reduir a cero la programació de scriptles en les vistes.

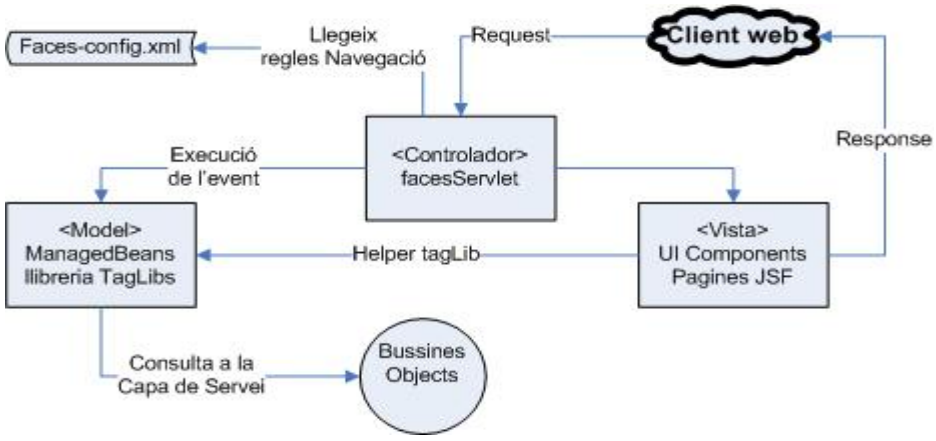
- La internacionalització i18n: struts proporciona la internacionalització dels missatges mitjançant un interceptor de les accions que es configura amb un arxiu amb la extensió “.properties” i que tradueix el missatges de forma escalable en funció de si el fitxer te el nom de la acció, interfície, classe, paquet o fitxer de configuració global.

- Validació de formularis definint les regles de validació amb fitxers configurables xml (<Nom de la classe de l’acció>-validation.xml) o mitjançant anotacions.

4.- Estudi del framework de presentació JSF

4.1- MVC aplicat al framework JSF:

Sun Microsystems va desenvolupar el framework de presentació Java Server Faces (JSF) seguint el patró MVC però diferenciant-se del altres en la seva orientació a esdeveniments - oients. En JSF trobem els següents components:



[figura 18: diagrama de petició - resposta de JSF]

- Controlador: Es tracta d'un Servlet FrontController (facesServlet) que esta configurat de la mateixa manera que en struts en el fitxer web.xml de l'aplicació i que centralitza les peticions que l'usuari fa. Aquest controlador llegeix les regles de navegació entre pàgines del fitxer de configuració faces-config.xml.

En aquestes regles es determina quina navegació tenim en una pàgina quant s'executa un esdeveniment o acció.

P. ex: al següent fragment del fitxer faces-config.xml es crea una regla de navegació que per la pàgina "introduirnom.jsp" al executar l'acció hola llançarà la pàgina hola.jsp i l'acció adeu llançarà la pàgina adeu.jsp.

```
<navigation-rule>
        <from-view-id>/pages/introduirnom.jsp</from-view-id>
        <navigation-case>
                <from-outcome>hola</from-outcome>
                <to-view-id>/pages/hola.jsp</to-view-id>
        </navigation-case>
        <navigation-case>
                <from-outcome>adeu</from-outcome>
                <to-view-id>/pages/adeu.jsp</to-view-id>
        </navigation-case>
</navigation-rule>
```

- Vista: En aquest cas la vista es una pàgina .jsf que implementa l'arbre "UI Components". Aquest arbre, que pot ser rendearitzat sobre una pàgina html o wml, conté una sèrie de components que faciliten la comunicació entre la vista i el model. Aquest components son del tipus:

    - UICommand: control que dispara *actions* quant s'activa.
    - UIForm: grup de controls que envien dades a l'aplicació.
    - UIGraphic: mostra una imatge.
    - UIInput: agafa les dades de entrada de l'usuari
    - UIOutput: mostra la sortida de dades de l'usuari.
    - UIPanel: mostra una taula.
    - UISelectItem: representa un ítem d'un conjunt de ítems
    - UISelectItems: representa un conjunt complet de ítems.
    - UISelectBoolean: per seleccionar valors booleans.
    - UISelectMany: permet seleccionar diversos ítems del grup.
    - UISelectOne: permet seleccionar un ítem del grup de ítems

L'accés a aquest components des de les pagines .jsf de la vista es fa mitjançant dos llibreries de etiquetes (taglibs) que faciliten la navegació per l'arbre i que s'han d'incloure al inici de la pàgina.

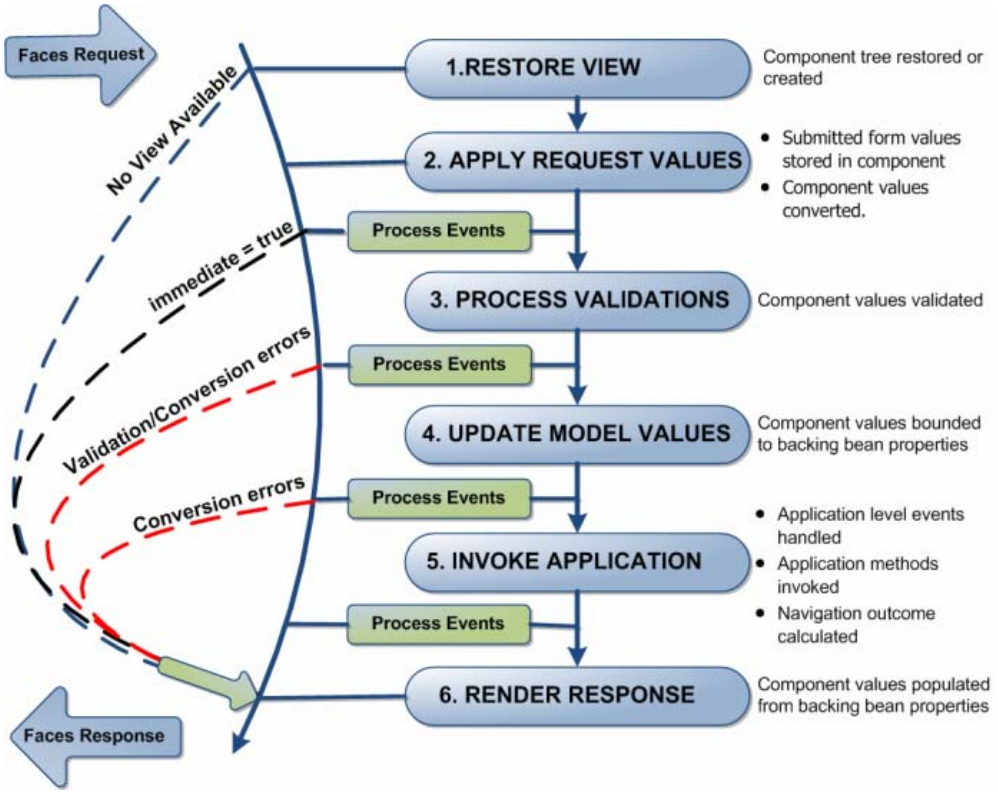
```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
< %@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
```

La llibreria jsf/core permet connectar els esdeveniments dels components de l'arbre amb el servidor mitjançant mapejadors esdeveniments amb els oients. La llibreria jsf/html permet crear els objectes de presentació de dades i missatges d'error en forma d'arbre navegable**.**

- Model: Els components especificats sobre la vista comuniquen amb el model mitjançant javabeans ("ManagedBeans") que han d'estar definits al fitxer de configuració del framework. Les seves propietats mitjançant getters i setters i els seus mètodes son els que enllacen sobre els components de la vista.

Els mètodes faran les crides necessàries sobre els objectes de negoci per tal de donar el model a la vista.

4.2 - Cicle de vida de una petició al framework:



[figura 19: diagrama de cicle de vida d'una petició JSF]

Quan arriba una petició al framework i s'ha d'executar una acció i pintar una pàgina jsf per donar la resposta al client s'ha de passar per les següents 6 fases:

1.- Reconstituir l'arbre de components:
Quan es fa una petició d'una pàgina jsf o es produeix un esdeveniment com clickar un enllaç o un botó, JSF construeix l'arbre de components, connecta aquest components amb el mapejador d'esdeveniments i validadors i guarda l'estat al FacesContext

2.- Aplicar valors als components:
Cada component de l'arbre agafa el seu valor dels paràmetres de la petició amb el mètode processDecode . Si es produeix un esdeveniment JSF comunica l'esdeveniment als oients interessats.

3.- Processar validacions:
En aquesta fase es fa el processament de les validacions dels components de l'arbre examinant els seus atributs especificats en les regles de validació i avaluant les regles amb els valors. Si s'incompleix una regla de validació s'afegeix un missatge al FacesContext i es renderitza la resposta amb els missatges d'error.
Si es produeixen errors de conversió en la petició - acció també es mostraran com a missatges d'error.

4.- Actualitzar els valors del model:
Validats els valors ja pot actualitzar el model. Si aquest valors no es poden convertir als atributs del model es passarà directament a la fase de renderitzat de la pagina resposta amb l'error produït.

5.- Invocar Aplicació:
Es la fase on JSF maneja qualsevol esdeveniment de l'aplicació ja sigui una redirecció a una altre pàgina, l'enviament d'un formulari o qualsevol altre resposta

6.- Renderitzar la resposta:
S'invoca la codificació dels components i es dibuixa els components de l'arbre guardat al FacesContext. Si s'han produït errors es dibuixa la pagina original amb els errors sobre les etiquetes output_errors.

4.3 - Característiques diferenciables de JSF:

- La característica mes remarcable es la diferencia amb altres frameworks per la seva orientació a esdeveniments: JSF instal·la en memòria un arbre de components i l'usuari produeix esdeveniments sobre els components que son tractats per donar una resposta en forma d'acció a l'usuari.

  Aquest esdeveniments manejats per oients quan son executats fan crides als objectes de negoci donant una resposta en forma d'una interfície (arbre de components) a la part de servidor. Aquesta resposta pot ser rendearitzada a una pàgina html, wml, xml....

- A partir de JSF2 s'ha facilitat la configuració del controlador a mitjançant anotacions java5 de manera que els controladors, les regles de navegació, el ManagedBeans i els validadors poden configurar-se mitjançant anotacions o mitjançant el fitxer faces-config.xml.

- Internacionalització i18n: es pot implementar la configuració d'idioma local a l'apartat <default-locale>es</default-locale> del fitxer de configuració.
  També es pot especificar un fitxer de missatges d'idioma local definit el fitxer properties a l'apartat <resource-bundle> del mateix fitxer de configuració.

- Suport directe d'Ajax amb la etiqueta <f:ajax>, validadors i conversors de dades amb etiquetes jsf-core que permeten fer la validació o la conversió de les dades d'un esdeveniment abans de la fase d'actualització del model.

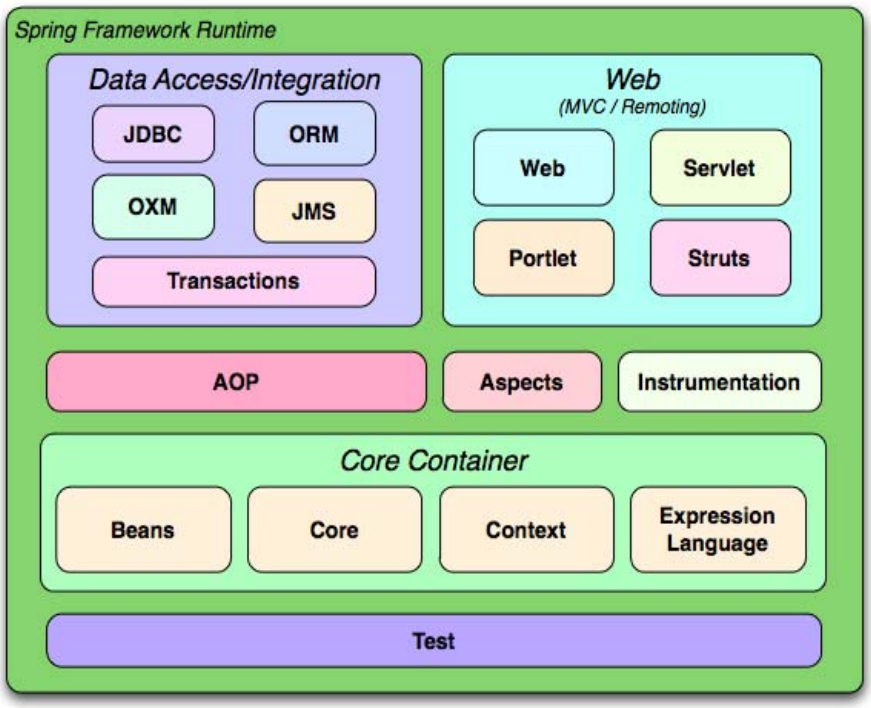
- La implementació de llibreries de suport com Icefaces, RichFaces que complementen el "UI components" amb nous elements que ajuden al programador en la presentació de la vista.

5.- Estudi del framework de presentació Spring MVC

5.1 - Mòduls del framework Spring:

El framework Spring comença a funcionar a partir del l'any 2002 quan Rot Johnson acompanya els casos pràctics en el seu llibre "Expert one-on-one J2EE Design and Development". A partir d'aquí la comunitat informàtica començar a donar-li noves aportacions sent ara mateix la versió 3 de l'actual framework.

S'ha de dir que Spring no es tracta només d'un framework de presentació web, si no d'un conjunt de mòduls que facilita el desenvolupament de programari i que seguidament donaré a conèixer:



[figura 20: Mòduls del framework Spring]

- Core Container: Es la part més important del framework on tenim un Factory de beans (BeansFactory), la injecció de dependències i la inversió de control.

  També trobem el mòdul Context que funciona de forma similar al registre JNDI facilitant l'accés a tots els objectes del framework, donant facilitats a la internacionalització i per la propagació d'esdeveniments.
  A mes a mes aquest mòdul també te un EL(Expression Language) especificat en jsp 2.1

- Data Acces/Integration: Aquest mòdul dona integració amb la capa de dades en una aplicació: te utilitats per poder fer connexions jdbc a la base de dades, l'ús de frameworks de mapeig d'objectes com iBatis, Hibernate, mapeig d'objectes - fitxers xml, el controls de transaccions i el mòdul per poder implementar un servei de missatgeria.

- Web: En aquest mòdul trobem el webServlet que es qui implementa Spring MVC i que estudiarem amb mes detall. A mes a mes també trobem connexions al framework Struts i un MVC implementat per entorns amb portlets.

- AOP / instrumentation: Entre d'altres coses aquest mòdul permet crear interceptors que permeten el desacoblament de codi.

- Test: Finalment aquest mòdul permet el testeig de aplicacions amb utilitats tipus junit.

5.2 - Mòdul WebServlet (Spring MVC):

Aquest mòdul esta compost per dos elements principals:

- Dispatcher Servlet: Es el controlador del MVC i esta implementat com un Servlet seguint el patró “frontcontroller” que rep de forma centralitzada les peticions del clients web. S’ha de configurar de la mateixa als altres dos frameworks dins del fitxer web.xml amb la següent configuració:

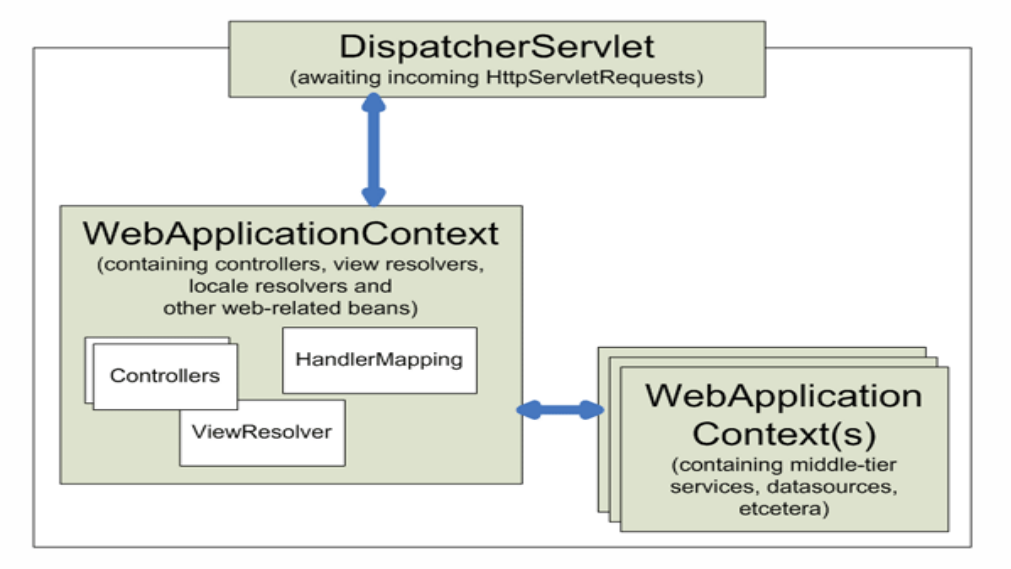
```
<web-app>
<servlet>
        <servlet-name>miservlet</servlet-name>
<servlet-class>
        org.springframework.web.servlet.DispatcherServlet
</servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
        <servlet-name>miservlet</servlet-name>
        <url-pattern>*.form</url-pattern>
</servlet-mapping>
</web-app>
```

  Quant el servlet s’inicialitza busca al directori web-inf de l’aplicació el fitxer de configuració amb el nom [servlet-name]-servlet.xml i crea els beans definits.

- WebApplicationContext: Es una extensió del ApplicationContext amb utilitats necessàries per aplicacions web que conté el següent tipus de beans

  - Controllers: controladors del MVC.
  - Handler mapping: mapejat de les peticions dels controladors i interceptors de pre-process i post-procés de les peticions.
  - View Resolvers: objectes que donen la vista al MVC
  - Locale Resolver: dona el locale del client per poder donar internacionalització.
  - Theme Resolver: donen temes per la web
  - Multipart file Resolver: per pujades de fitxers multipart en formularis html.
  - Handler exception resolver: mapeja excepcions a les vistes.



[figura 21: Mòduls del webservlet]

Per la implementació dels Controllers s'utilitzen objectes POJOS que no necessiten estendre de cap classe concreta i que contenen les accions a executar del MVC. Per definir aquest objectes i que el WebApplicacionContext pugui trobar-los s'utilitzen anotacions java per tal de definir l'objecte.

Amb l'anotació @Controller es defineix aquest tipus de objectes controladors. Es el DispatcherServlet el que fa un escaneig al inicialitzar-se buscant aquesta anotació en els objectes que l'usuari del framework defineix com a controladors i els registra al framework com a beans. En aquest registre també es mapeja el path al que respondrà l'acció amb la anotació @Requestmapping.

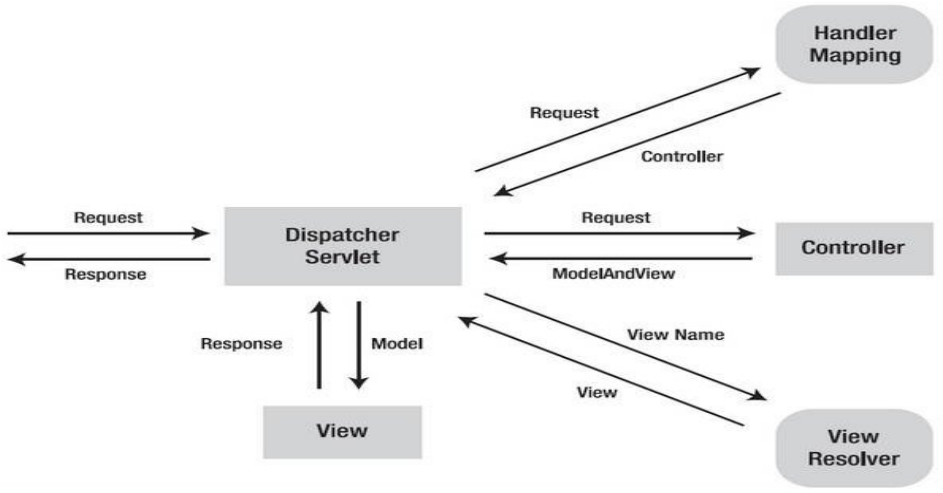
P.ex.: En el següent controlador HelloWordController es defineix l'acció HelloWorldController que s'executarà en el mapeig de la url /helloWold.
Aquest controlador retorna un objecte ModelAndView que conté de forma com compactada el model i el nom de la vista del MVC. El model conté un mapa amb els objectes retornats per la lògica de negoci.

```
@Controller
public class HelloWorldController {

        @RequestMapping("/helloWorld")
        public   ModelAndView helloWorld() {
                ModelAndView mav = new ModelAndView();
                mav.setViewName("helloWorld");
                mav.addObject("message", "Hello World!");
                return mav;
        }
}
```

5.3 - Seqüència d'una petició Spring MVC:

Com es veu al següent gràfic, una petició sobre el framework de Spring es centralitzada al dispatcherServlet. Aquest consulta quin es el controlador que respon al mapejat de la request, si te configurat interceptors sobre la acció es pre-processant i es post-processan aquest interceptors, desacoblant d'aquesta manera el codi.



[figura 22: Diagrama de seqüències d'una petició Spring MVC]

Aquest interceptors implementen la interfície HandleInterceptor i executen tres mètodes: un abans de que el handler de la petició s'executi, un altre després de la execució del handler i un altre al final de la petició. Si qualsevol dels mètodes retorna un false la acció es avortada.

Si el dispatcher ha trobat el controller i el mètode a executar aquest executa el seu codi mitjançant "java reflection" i retorna al dispatcherServlet un mapa amb els objectes retornats per la implementació del model i el nom lògic d'una vista per renderitzar de forma encapsulada en un objecte ModelAndView.

El següent pas es que el View Resolver pugui resoldre del nom lògic de la vista i pugui rendearitzar-la al tipus de vista. Aquest "viewResolver" poden implementar la resolució de la vista com una pàgina jsp, com un xml, com el mapeig a un recurs de una url, com un ajudant de plantilles tipus velocity o freemaker, o també podem implementar la nostra solució de resolució de la vista a partir de la classe abstracta.
Finalment aquesta vista es retornada al DispatcherServlet per que la presenti en forma de resposta al client web.

5.4 - Característiques diferenciables de Spring MVC:

Spring es un framework que supera l'abast del estudi d'aquest projecte per les grans dimensions de tots els seus mòduls. Encara que això pot semblar un desavantatge per la seva corba d'aprenentatge tan alta, la resolució de les problemàtiques de cada mòdul permeten fer aplicacions molt robustes en l'àmbit de aplicacions web de client fi.

Del mòdul de WebServlet podem destacar:

- La utilització de anotacions java per facilitar la configuració dels controladors i el mapejats de les request amb sistemes d'escaneig automàtic dels controladors en temps d'inicialització del framework.

- La utilització de la Inversió de control (IoC) fa que siguin els contenidors quin executin l'acció donar les respostes al programador.

- Suport a la internacionalització amb la configuració del idioma local del usuari en la classe ResourceBundleMessageSource per part del WebAplicationContext.

- La encapsulació del model i la vista en un objecte amb la implementació de diferents tipus de resolucions de la vista lògica.

- La comunicació amb altres frameworks de presentació (Struts), frameworks de persistència (iBatis, Hibernate) i Helpers per crear vistes compostes (freemaker, velocity).

- La incorporació de interceptors en el pre-processament i post-processament de les peticions que permeten desacoblar el codi de la acció.

- La validació de formularis amb les anotacions @valid definides al mètode o mitjançant la interfície validate.

- Documentació molt extensa aportada per el projecte Spring (http://www.spring.org) .

6.- Conclusions finals de l'estudi de frameworks de presentació

Fet l'estudi dels tres frameworks més utilitzats al mercat s'han estret les següents conclusions:

- Els tres frameworks segueixen el Model Vista Controlador:

Mentre que Struts utilitza un filtre com a controlador, Spring MVC i JSF utilitzen Servlets per centralitzar la tasca de controlador de les peticions.
Els tres frameworks deixen al model la llibertat per accedir al objectes de negoci. En el cas de Spring MVC s'empaqueta en un objecte ModelAndView la vista i la resolució del model mitjançant un mapa.
El nostre framework hauria de seguir el mateix patró ja que s'ha demostrat que la centralització de les peticions en un servlet o filtre facilita la resolució de les accions que programa l'usuari del framework.

- Internacionalització i18n:

Els tres frameworks donen la possibilitat a l'usuari de fer una programació de les seves vistes de manera que siguin adaptades al Locale del navegador mitjançant fitxers de propietats i pugui adaptar les aplicacions multiidioma mitjançant un objecte resoucebundle. El framework a dissenyar ha de poder permetre adaptar una aplicació al idioma que vulgui el programador.

- Anotacions Java:

Els tres frameworks deixen la possibilitat de utilitzar el sistema de anotacions de java 5 per tal d'implementar les accions del model, encara que també permeten fer la configuració del framework via fitxer xml com feia struts 1.
El nostre framework també utilitzarà anotacions java per configurar els controladors i les seves accions ja que encara que es un sistema que carrega la memòria d'execució del programa permet fer una configuració del framework molt mes ràpida i senzilla per el programador.

- Helpers per la implementació de la vista:

Els tres frameworks implementen el patró ViewHelper amb la utilització de llibreries de tags o javabeans que permeten a l'usuari del framework la no utilització de scriptles al codi.

Aquesta no inclusió de codi a la vista facilita la seva comprensió i per tant facilita la diferenciació de rols en la programació del codi que utilitza la vista i el disseny de la seva presentació.

El framework a dissenyar ha d'implementar llibreres de tags personalitzades i javabeans per tal que les vistes siguin de fàcil comprensió i que tinguin el mínim de codi en la seva programació.

Aquest mecanismes ajudaran també al dissenyador a fer validacions en els formularis sobre camps requerits o poder fer conversions de dades sense haver de ficar codi al la vista.

- El patró AplicationController:

Es interessant deixar el FrontController lleuger i centralitzar les tasques de configuració del framework (lectura de fitxers xml, configuració del mapa de accions...) i la de peticions sobre el framework (cerca d'accions quan arriba la petició, execució i validadors de l'acció, renderització de la vista...) sobre un controlador d'aplicació de la mateixa manera que fa Struts 2.

- VistaComposta:

Struts i Spring MVC donen facilitats per utilitzar motor de plantilles per implementar el patró CompositeView mitjançant la inclusió de productes de tercers desenvolupats en java com poden ser Velocity o Freemaker. S'estudiarà la possibilitat d'incloure un d'aquest productes dins del framework a dissenyar.

7.- Disseny d'un framework de presentació (jwframework)

7.1 - Aplicació dels Patrons al framework:

Com ja s'ha comentat en el punt anterior aplicarem els patrons més adequats en el desenvolupament del framework. Entre els mes destacats considerem:

- ServletController: Utilitzarem un Servlet com a estratègia per centralitzar les peticions HttpRequest sobre el framework. Aquest servlet delegarà les seves funcions sobre un controlador d'aplicació que es el que executarà les accions provocades per les peticions i d'aquesta manera deixarem el Servlet lo mes àgil possible.

- AplicationController: Amb l'aplicació d'aquest patró pretenem que el controlador d'aplicació faci les funcions de inicialització del framework amb la configuració de les dades via un fitxer xml (tal con feia els fameworks estudiats), que ompli un mapa d'accions a partir d'un escaneig en temps real de les accions configurades per l'usuari mitjançant anotacions java en objectes pojos i que sigui capaç de invocar qualsevol d'aquestes accions a petició del mapejat d'una request sobre l'acció concreta mitjançant utilitats com el java reflection con fan els frameworks estudiats en l'apartat anterior.

  Recordem que en una estratègia "Comand handler Strategy" d'aquest patró el AplicationController resolia sobre un CommandFactory quina era l'acció a executar (command) i quan la podia instanciar la invocava amb el CommandProcessor. Aquesta serà la tècnica que utilitzarem per invocar les accions del nostre framework.

- ViewHelper: Per tal d'adaptar el model que retorna l'acció executada necessitem d'eines que facilitin la integració del model amb la vista i que no sobrecarreguin el disseny de les vistes. La utilització d'aquest patró implementant eines com una llibreria de tags o amb l'ajuda de javabeans creats per el programador li facilitarà la tasca de creació de les vistes.

- CompositeView: No es objecte directe de la creació del framework però sembla interessant la utilització d'aquest patró sobre el desenvolupament de les vistes ja que facilita la reutilització del codi.

7.2 - Diagrama de classes de jwframework:

A partir de l'estudi dels patrons anteriors implementarem un framework desenvolupat en java que solucionarà la problemàtica de presentació de la capa web estudiada. Aquest framework d'ara endavant li denominarem jwframework (java web framework).

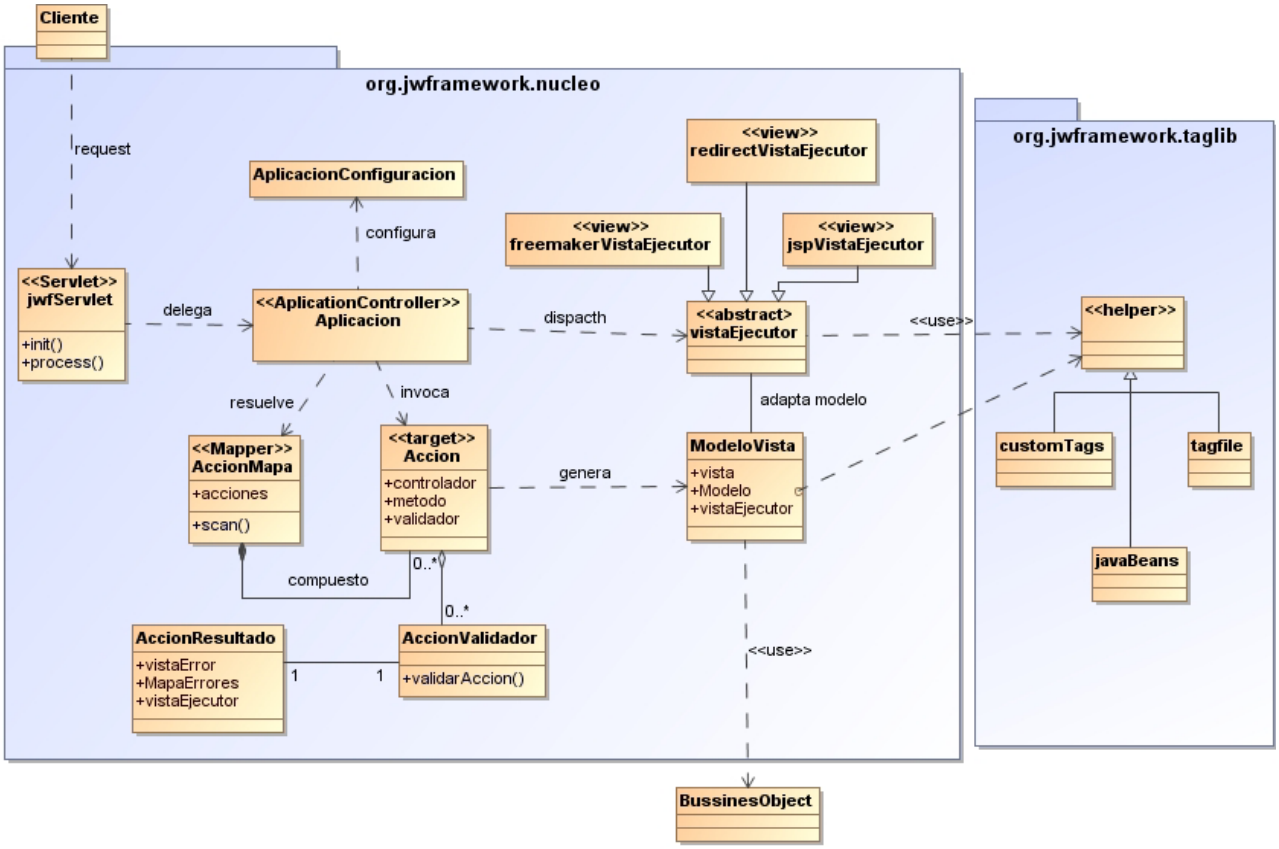
El jwframework estarà implementat a partir de un conjunt de classes relacionades integrades en diferents paquets:

- org.jwframework.taglib: contindrà les classes que una llibreria de tags feta a mida per ajudar al programador en la presentació de les vistes.

- org.jwframework.anotaciones: contindrà les interfícies de les anotacions necessàries per executar les accions en temps d'execució amb el java reflection.

- org.jwframework.excepciones: contindrà la relació d'excepcions que es poden donar en la inicialització i l'execució del framework.

- org.jwframework.nucleo: contindrà les classes del nucli del framework i serà les que explicarem amb mes detall.



[figura 23: Diagrama de classes de jwframework]

- jwfServlet: Servlet que centralitzarà les peticions sobre el framework. Aquest Servlet es configurarà en el fitxer web.xml de l'aplicació que utilitzi el framework.
    - Init() inicialitzarà el framework creant el controlador Aplicació.
    - Process() servirà les peticions del servlet ja siguin gets o pots delegant la execució d'accions i el dispatch de la vista sobre el controlador d'aplicació

- Aplicacion: Controlador d'aplicació que gestionarà el mapeig de les peticions sobre accions, la seva invocació i execució en funció de l'abast del controlador (de request o sessió), la gestió dels interceptors assignats i el control sobre els paràmetres de les accions.

- AplicacionConfiguracion: classe que llegirà el fitxer jwframework-config.xml i configurarà els paràmetres del framework.
  Destaquem entre els paràmetres: la gestió de la internacionalització de missatges mitjançant la definició especifica del locale i el seu fitxer de missatges, juntament amb un debug sobre el framework amb log4j.

- AccionMapa: Gestionarà un mapa de petició - acció de tots els controladors definits al framework per ajudar al controlador de aplicació a executar l'acció.

- Accion: acció a invocar i executar segons el mapeig del AccionMapa.

- Interceptor: Interfície que implementarà el mètode validarAccion() per tal de interceptar la petició de una acció, validar-la i redireccionar la si fos necessari. Les classes interceptores d'accions haurà d'implementar aquesta interfície.

- InterceptorResultado: classe retornada per l'interceptor amb un mapa d'errors si s'ha de redireccionar la petició sobre l'acció.

- ModeloVista: Empaquetat retornat per la execució d'una acció que conté el model mitjançant un mapa i el tipus de dispatcher de la vista.

- VistaEjecutor: Dispatcher abstracte de la vista que podrà ser de diferents tipus. S'implementarà el dispatcher jsp, el dispatcher per plantilles freemaker i un dispatcher per fer redireccions de la request a altres accions

En quant a les interficies" necessàries per la execució de les accions mitjançant anotacions amb java reflection destaquem:

- Controlador: Interfície necessària per anotar quins pojos de l'aplicació seran controladors d'accions. Tindrà les següents propietats:
  - Nombre: nom del controlador a instanciar.
  - RequestMap: path de la petició a la que respondrà el controlador.
  - Ambito: àmbit del controlador (sessió o request per defecte).

- AccionEjecutar: Interfície que identifica l'acció del controlador a invocar.

  - RequestMap: path afegit al del controlador per la invocació del mètode o acció.
  - Interceptor: Classe que implementarà la interfície Interceptor que validarà l'acció. Per defecte Null.

- AcccionEjecutarParametros: Interfície que identifica els paràmetres de l'acció. necessaris per el pas dels paràmetres de la petició del client sobre l'acció.
  - Nombre: Contindrà el nom del paràmetre

Amb aquestes tres interfícies podrem conèixer quin serà el controlador a instanciar, d'aquest controlador quina serà la acció a invocar en funció del mapeig path petició - acció i quins paràmetres necessita i s'ha de passar de la request.

Tan a les peticions de les accions com a la inicialització del jwframework es podran donar les següents excepcions que estaran recollides al paquet org.jwframework.excepciones:

- jwfException: serà la excepció pare des de on estendrà la resta d'excepcions.

- AccionDuplicadaException: es podrà donar quant en la configuració inicial del jwframework es detecti una acció duplicada en el mateix controlador. Igualment es permetrà continuar amb la inicialització sense fer cas de la segona acció, però donant l'avís al programador.

- AccionNoEncontradaException: es produirà quant el programador faci una petició al jwframework amb el nom del controlador o acció equivocat. Es mostrarà missatge pel navegador i es pararà la execució de l'aplicació.

- AccionArgumentosException: es produirà quant el programador declari arguments a la request que siguin diferents als que se necessita per la execució de l'acció. Es pararà la execució de la aplicació.

- vistaEjecutorException: es produirà en una fallida de la execució de la vista i pararà la execució de la aplicació.

- InterceptorException: es produirà en una fallida de la intercepció de la petició de la acció i pararà la execució de la aplicació.

Finalment i tal i com mostra el diagrama de classes anterior es facilitarà un helper al programador para que pugui evitar la implementació de scriptles dins de les pagines de les vistes. Aquest helper que serà similar a altres frameworks estudiats a la fase anterior estarà implementant mitjançant una llibreria de tags personalitzada amb següents tags mes importants i que podrà ser ampliada en altres fases posterior de manteniment del framework:

- FormTag: s'utilitzarà per la creació de formularis amb validació javascript de camps obligatoris i camps que siguin numèrics. El tag generarà les funcions de validació javascript necessàries.

- InputTextTag: permetrà introduir inputs de tipus text al formulari que siguin requerits amb el seu missatge de camp obligatori.

- InputPasswordTag: igual que l'anterior però en aquest cas per camps de tipus password.

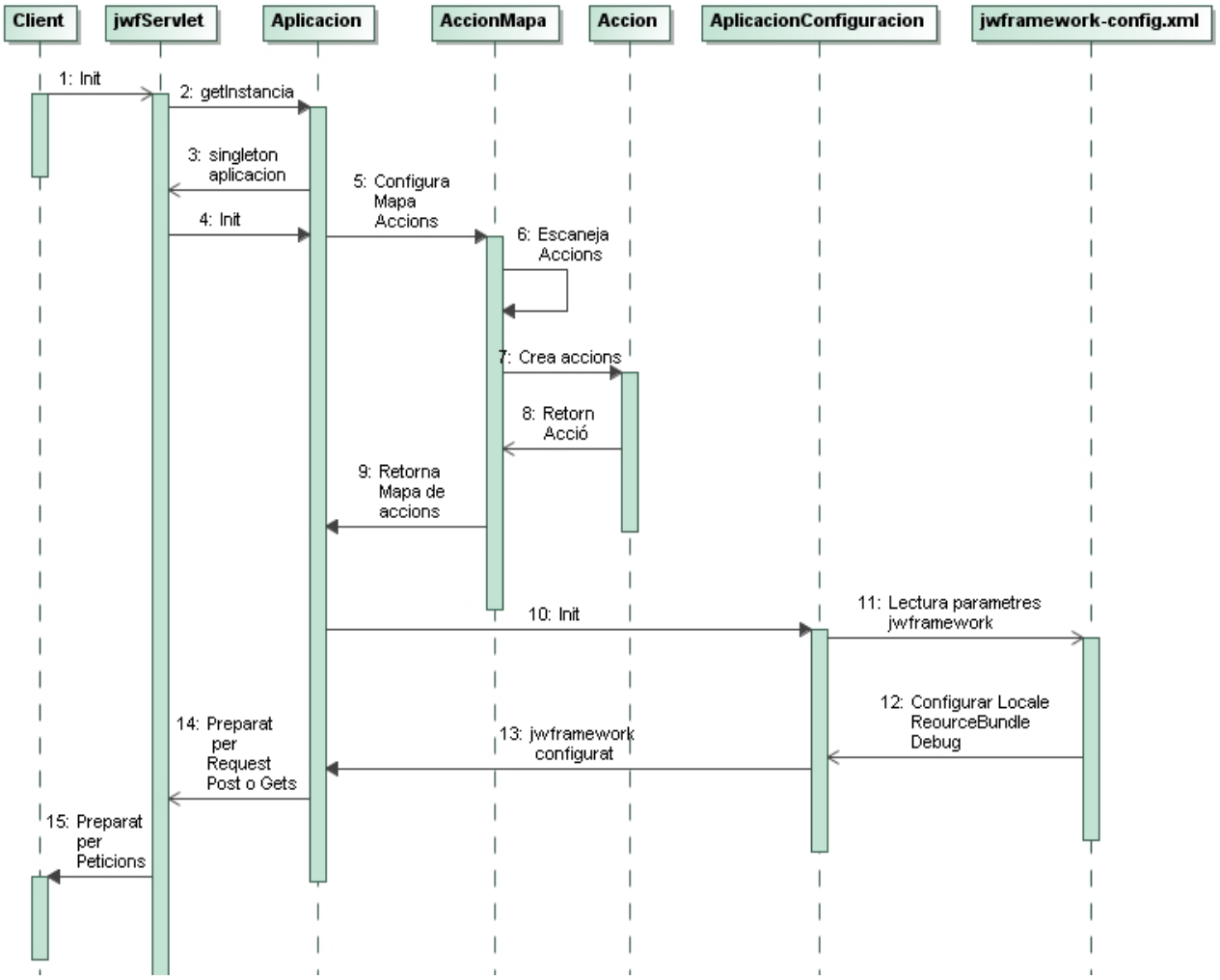
- ErrorTag: tag que permetrà mostrar sobre el navegador un mapa d'error produïts per les accions.

- TablaTag: component que permetrà incloure un iteractor per mostra una taula html passant-li una llista d'elements.

- SelectTag: component que permetrà incloure un iteractor per mostra una select html passant-li una llista d'elements.

7.3 - Diagrama de Seqüències d'inicialització del jwframework:

Seguidament faré menció a les fases d'inicialització del jwframework amb l'execució de cada una de les seves fases:



[figura 24: Diagrama de seqüències de inicialització del jwframework]

1.- Es demana la inicialització del servlet de la aplicació a partir de les dades contingudes al fitxer web.xml de l'aplicació que utilitzarà el jwframework. En aquest fitxer es configura el jwfServlet de la següent forma:

```
<servlet>
    <servlet-name>jwfServlet</servlet-name>
    <servlet-class>org.jwframework.nucleo.jwfServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>jwfServlet</servlet-name>
    <url-pattern>*.jwf</url-pattern>
</servlet-mapping>
```

Es a dir, que es configura un servlet denominat jwfServlet que rebrà peticions amb el sufix *.jwf.

2.- S'instancia l'objecte Aplicacion seguint un patró "Singleton" que serà el qui es responsabilitzi de les peticions dels jwfServlet. Aquest patró fa que només es pugui crear una instancia de l'objecte Aplicació.

4.- S'inicialitza l'objecte Aplicació:

- Es crea un mapa d'accions escanejant els objectes pojos que tinguin l'anotació de controlador de tota la aplicació web.

- Al mapa d'accions es guarden objectes "Accions" segons la seva RequestMap. Aquest objectes contenen el nom del controlador a instanciar , el mètode a invocar i els seus paràmetres.

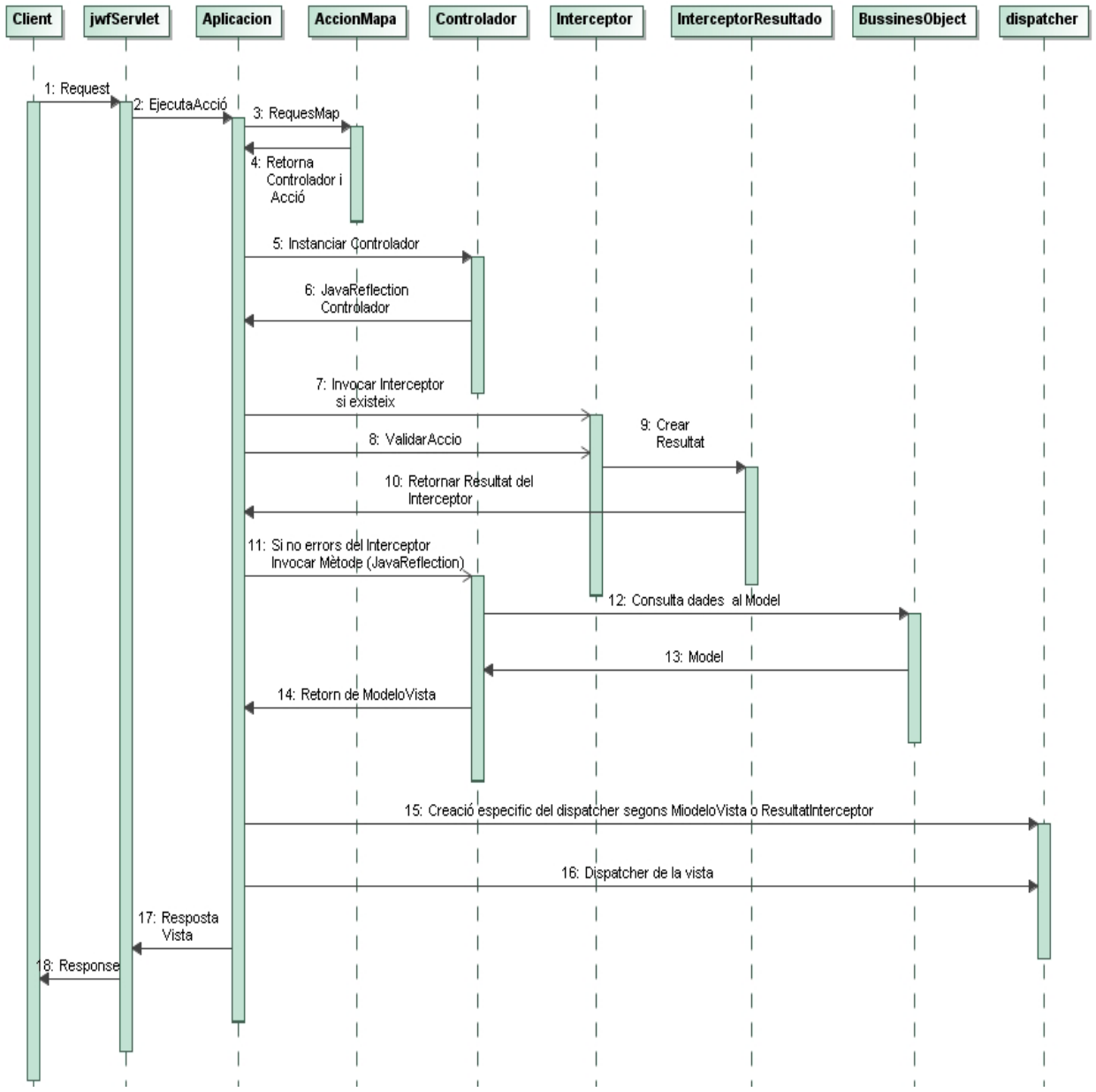
10. –S'inicialitza l'objecte AplicacionConfiguracion:
El objecte Aplicacion contindrà una instancia a un AplicacionConfiguracion que es configurarà mitjançant un fitxer jwframework-config.xml.
En aquest fitxer es configurarà l'activació del debug, la configuració d'idioma de l'aplicació, el fitxer de missatges per idioma i la execució de plantilles d'eines com el freemaker per al dispatcher freemakerVistaEjecutor.

No s'ha dibuixat a l'esquema la intercepció de possibles jwfExceptions o excepcions que es poden donar en el procés d'inicialització. Excepcions que podrien donar-se en el control de la lectura del fitxer de configuració, la duplicació de una acció en la creació del mapa d'accions....

15.- Si la inicialització del framework ha funcionat correctament ja esta preparat per rebre peticions del client web mapejades per la request del jwfServlet amb la extensió *.jwf.

7.4 - Diagrama de Seqüències de petició al jwframework:

A partir d'una inicialització correcta del jwframework aquest ja estarà preparat per rebre les peticions. El següent esquema reflexa una petició sobre el jwframework i com aquestes peticions es traduiran en accions que seran executades i retornades en forma de vista. D'aquesta manera tancarem un cicle MVC.



[figura 25: Diagrama de seqüències de petició - resposta del jwframework]

1.- A partir d'una Request del client web que arribi al jwfServlet es tradueix amb el mateix nom en una acció (traient-li el sufix jwf). Aquesta acció es executada per l'objecte Aplicacion.

2.- Per executar l'acció el objecte Aplicacion busca en funció del RequestMap o path de la petició un objecte Accion sobre el mapa d'accions.

5.-A partir d'aquest objecte Acció a través de l'anotació Interceptor consultem si te associat un interceptor a executar. Si es així s'instancia l'objecte interceptor concret i s'executa el mètode validarAccion() que podrà redireccionar la vista associada a la petició a una nova vista d'error segons el retorn del InterceptorResultado contingui un mapa d'errors produïts.

11.- Si no s'han produït errors en la fase d'intercepció, invoquem amb javaReflection el mètode o acció concreta passant els paràmetres necessaris de la request al mètode. Aquest mètode pot interrogar a objectes de negocis per tal de retornar-li el model de les dades.

14.- El retorn d'aquest mètode serà un objecte ModeloVista similar al ModelAndView de Spring que contindrà un mapa de les dades del model, el nom de la vista i un dispatcher concret que estendrà de vistaEjecutor.

17.- A partir del nom de la vista i el seu dispatcher jwfServet delegarà al objecte Aplicació la presentació de la vista retornant d'aquesta manera la resposta al client web.

En aquest cas tampoc s'ha dibuixat a l'esquema la intercepció de possibles jwfExceptions o excepcions que es poden donar en el procés processament de peticions, com podria ser el cas de Accions no trobades al mapa d'accions, paràmetres incorrectes segons l'acció...

8.- Gespro (Gestió de projectes): jwframework sobre cas real

Gespro es un cas real d'aplicació del jwframework i prenen reduir la corba d'aprenentatge de la utilització del framework en la programació d'aplicacions web de client fi.

Tracta de solucionar la gestió integrada de les incidències detectades en els projectes d'una empresa d'informàtica proporcionat les següents funcionalitats:

- Llistat de Projectes**:** Proporciona un llistat dels projectes que te la empresa actualment amb accés al manteniment (alta/baixa/modificació/ assignació de recursos al projecte).
  Només els usuaris administradors podran modificar les dades.

- Llistat de Empleats: Proporciona un llistat dels empleats de la empresa amb el seu manteniment corresponent. Només els usuaris administradors podran modificar les dades.

- Llistat de Incidències: Proporciona un llistat filtrat per treballador loginat de les incidències produïdes als projectes amb la seva introducció de hores imputades.

- Login sobre l'aplicació: No farà falta loginar-se per veure les dades bàsiques dels empleats o projectes de la empresa, però si per tasques de manteniment i introducció de incidències. La autenticació sobre l'aplicació serà amb el username i el password validat sobre la base de dades.

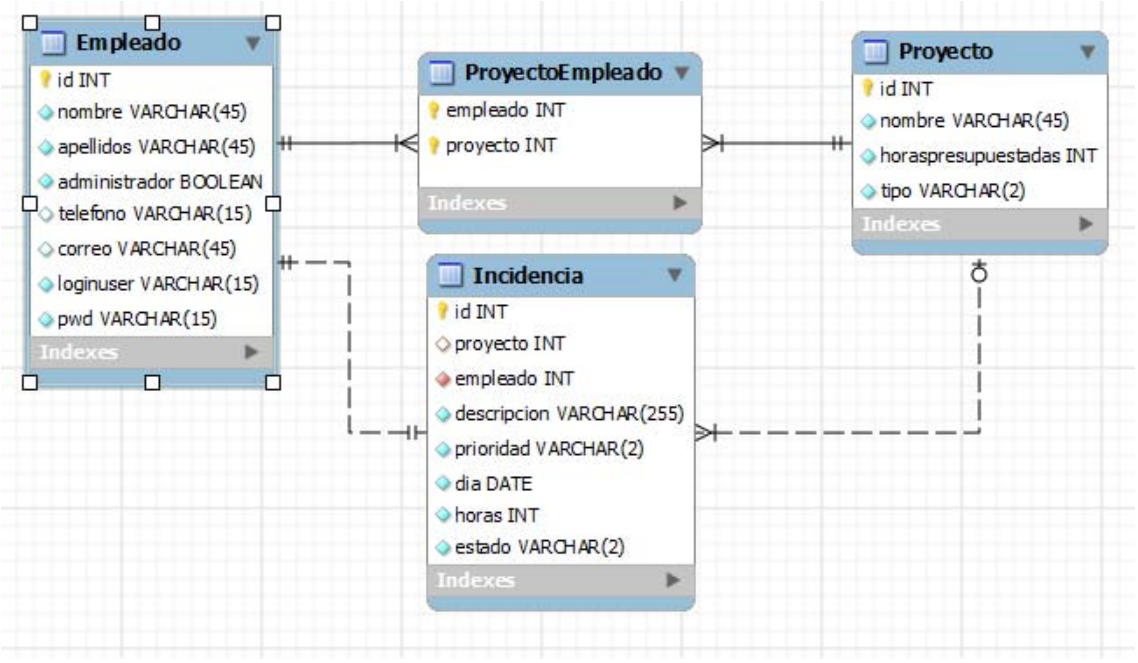
- Control de Projecte: Proporciona al cap de projecte un llistat filtrat per projecte del conjunt d'incidències i hores dedicades per tots els empleats que han treballat al projecte. Només els usuaris administradors tindrà accés.

Gespro estarà diferenciat per tres capes de disseny:

- La capa de presentació: Estarà formada per el nostre framework configurat per rebre peticions amb el sufix *.jwf i les accions de les peticions programades en el paquet org.gespro.accions que s'inicialitzarà al framework

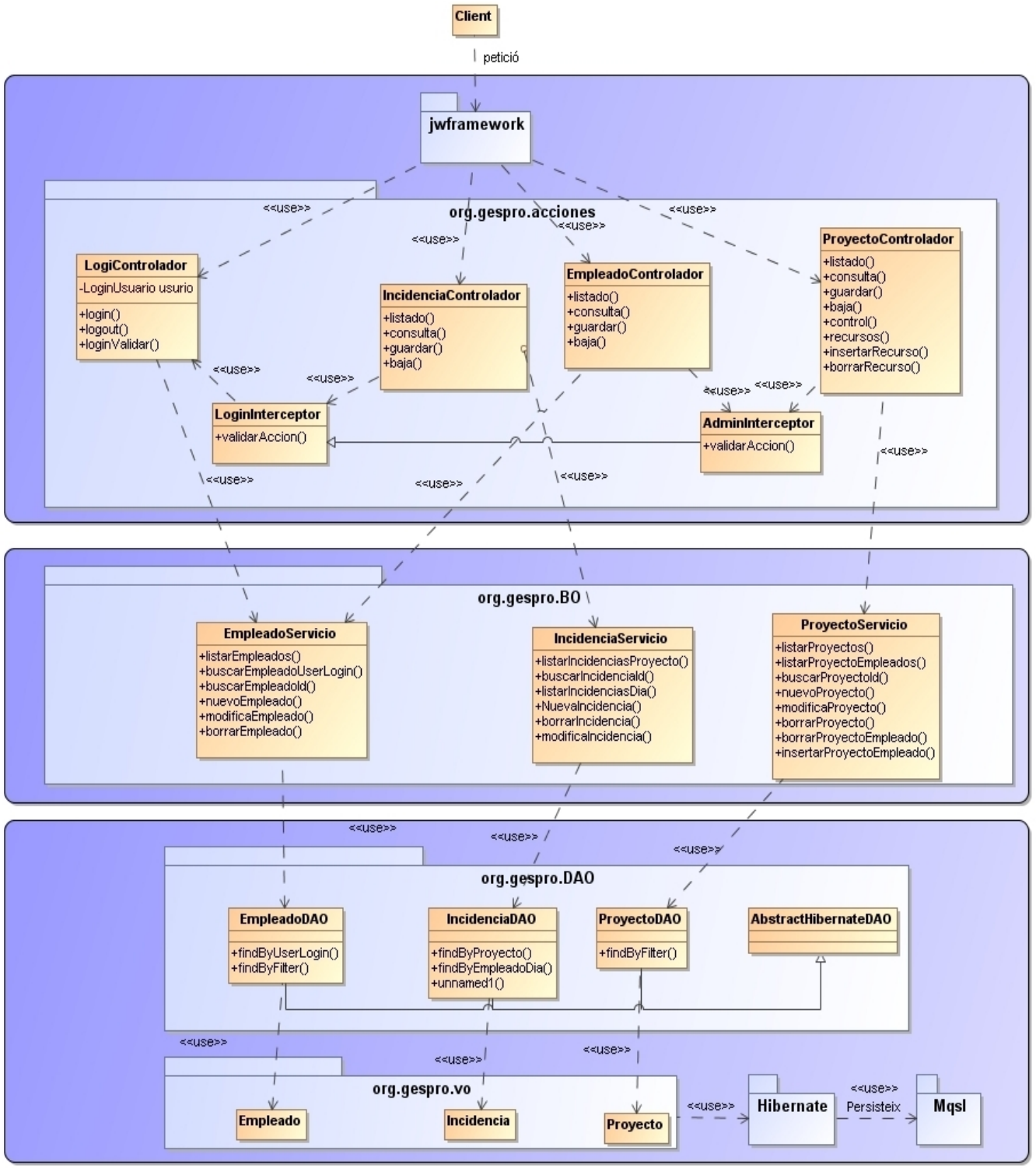
- La capa de negoci: Estarà formada per un conjunt de serveis que accediran a la capa de persistència per donar el model a la capa anterior. Aquest serveis estaran implementats en el paquet org.gespro.BO.

- La capa de persistència estarà gestionada per la base de dades Mysql gespro amb un mapejat de les seves entitat i relacions mitjançant pojos de java implementat en el paquet org.gespro.vo. Aquest mapejat es farà mitjançant el framework de persistència hibernate 3.2.5 seguint un patró DAO (Data Acces Object) per cada entitat implementat en el paquet org.gespro.DAO. El seu model Entitat-Relació serà el següent:



[figura 26: Diagrama Entitat - Relació de base de dades Gespro]

A continuació explicaré les classes mes importants de les dos capes superiors: la capa de servei i la capa web que serà on estarà aplicat el jwframework:



[figura 27: Diagrama de classes de Gespro]

A la capa de servei trobem les següents classes:

- EmpleadoServicio: Conté els mètodes necessaris per donar totes les operacions sobre la entitat empleats necessàries per la capa web. Aquestes operacions son de creació, modificació i alta d'empleats, juntament amb la validació per userlogin i password de l'usuari de gespro i el llistat filtrat d'empleats

- IncidenciaServicio: Conté els mètodes necessaris per donar totes les operacions sobre incidències a la capa superior. Aquestes operacions son de creació, alta i baixa d'incidències, llistar les incidències d'un empleat per dia per tal de fer el manteniment diari i llistar les incidències per projecte per tal de fer el control de projecte per part del cap de projecte o administrador.

- ProyectoServicio: Conté els mètodes que donen les operacions sobre la entitat de projectes(alta,baixa,modificació i llistat de projectes) i la seva entitat associada de empleats que treballen en un projecte (inserir i esborrar empleats al projecte).

Finalment quedarà la capa de servei que segons la figura presentada anteriorment trobem el jwframework com a centralitzador de totes les peticions de l'usuari de gespro i quatre controladors específics seguint la metodologia de jwframework:

- LoginControlador: Es tracta d'un controlador de sessió que guardarà les dades de l'usuari loginat a gespro i que permetrà les accions mapejades de login, logout i loginValidar per la validació amb la entitat de empleats.

- EmpleadoControlador: Es un controlador de Request que permetrà les accions de llistar de forma filtrada la entitat de empleats juntament amb la seva la consulta i modificació, i eliminació de les dades guardades a la base de dades.

- ProyectoControlador: Es un controlador de Request que permet les accions de llistat filtrat, consulta, modificació i eliminació de les dades dels projectes; juntament amb la assignació i eliminació de recursos del projecte i també el control de projecte .

- IncidenciaControlador: Controlador de Request que permet les accions de llistat, consulta, modificació i eliminació de les dades de incidències.

Aquest controladors tenen accions que necessiten de validacions sobre al base de dades tant de usuaris loginats com per saber si el tipus d'usuari es un administrador. Per aquestes validacions utilitzarem els següents interceptors:

- LoginInterceptor: el seu mètode validarAccion() consultarà al controlador de sessió LoginControlador si l'usuari s'ha autentificat en gespro per donar-li accés al manteniment d'incidències.

- AdminInterceptor: el seu mètode validarAccion() consultarà si l'usuari te permisos d'administrador per gestionar els empleats i els projectes de Gespro.

9.- Implementació del jwframework i de l'aplicació gespro

Per implementar el jwframework i l'aplicació web gespro s'ha programat en java 1.6 a través de l'IDE Netbeans en la seva versió 7.1

En la implementació es dona les carpetes jwframework i gespro que son projectes importables a Netbeans amb els fonts de les aplicacions.

Cal remarcar en la implementació del jwframework que es donen dependències a altres llibreries que s'han d'importar per la implementació de aplicacions webs. Aquestes llibreries son:

- commons-digester3-3.2.jar
- commons-logging-1.1.1.jar

  Utilitzats per la lectura del fitxer de configuració jwframework-config.xml i traspàs de dades del fitxer a objectes de memòria

- commons-beanutils-1.8.3.jar: utilitzat per facilitar el traspàs les dades del la request als beans sense necessitat d'implementar getters i setters.

- freemaker.jar: per utilitzar plantilles ftl en la implementació de dispatchers de freemaker.

- log4j-1.2.16.jar: per implementar el log del jwframework i facilitar el debug a l'usuari de les accions.

- jsp-api.jar i servlet-api.jar utilitzades per fer referències a objectes request, session i aplication des de java.

En quan a les dependencies de gespro a mes a mes d'importar les anteriors llibreries i el jwframework.jar s'ha de tenir en compte els següents punts:

- Per fer les proves de contenidors de dades s'ha utilitzat un Msql 5.0.45 amb el port estàndard 3306 en la seva configuració obert. El script de creació de la base de dades i el joc de proves es dona a la carpeta "BD Gespro".
  L'usuari per entrar en l'aplicació Gespro es el mateix de la uoc amb el mateix password (es pot veure al llistat d'empleats).

- El connector mysql-connector-java-5.0.5-bin.jar que esta a la llibreria de dependències s'utilitza en gespro per fer la comunicació amb mysql des de java.

- Per crear el mapejat de pojos que comuniquen amb mysql s'ha utilitzat hibernate 3.2.5 i s'han d'importar al projecte de Gespro. A la seva versió distribuïble ja estan dins de la carpeta lib.

- Per tal de fer una programació de les vistes en jsps mes clara s'ha utilitzat el llenguatge EL (Expression Language) i s'ha importat les seves llibreries (standar.jar i jstl.jar).

A continuació es dona la configuració bàsica del jwframework-config.xml per tal de fer una configuració inicial del jwframework en qualsevol aplicació web.

```
<?xml version="1.0" encoding="UTF-8"?>
<jwframework-config
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <debug-mode>true</debug-mode>  → false si no es vol debug
    <freemaker>false</freemaker>  → false si no es vol freemaker

    <locale-config>
      <language>ca</language> → ca per català, es per castellà
      <messages-file>
        org.gespro.acciones.messages → paquet de localització del bundle
      </messages-file>
      <resourcebundle>msg</resourcebundle> →nom del bundle
      </locale-config>
</jwframework-config>
```

I una configuració del fitxer web.xml de configuració de l'aplicació web:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

   <servlet> → configuració del servlet del jwframework
      <servlet-name>jwfServlet</servlet-name>
      <servlet-class>org.jwframework.nucleo.jwfServlet</servlet-class>
   </servlet>

   <servlet-mapping> → Mapeig del sufix de les peticions de jwframework
      <servlet-name>jwfServlet</servlet-name>
      <url-pattern>*.jwf</url-pattern>
   </servlet-mapping>

   <jsp-config>
        <taglib>→ TagLib del jwframework utilitzat

                <taglib-uri>http://jwframework.org/taglib</taglib-uri>
                <taglib-location>/WEB-INF/jwframework.tld</taglib-location>
         </taglib>
   </jsp-config>
</web-app>
```

L'aplicació exemple Gespro s'activa al desempaquetar el fitxer gespro.war sobre un servidor tomcat7 i podem utilitzar-lo a partir de la següent direcció electrònica http://localhost:8080/gespro.

Al entrar com usuaris administradors amb usuaris de la uoc i la mateixa paraula de pas podrem consultar les dades del empleats i projectes.

[figura 28: llistat de projectes de Gespro amb usuari administrador loginat]

Conclusions finals del projecte final de carrera – J2EE

El desenvolupament d'aquest projecte de final de carrera m'ha donat una nova visió en el desenvolupament d'aplicacions j2ee: lluny de la visió practica d'un projecte concret i posant-nos en la pell d'un arquitecte de programari l'estudi i desenvolupament d'un framework posa de manifest una abstracció de les problemàtiques que ens trobem tots els programadors a l'hora de fer les aplicacions de client prim.

Al inici d'aquest projecte semblava molt complicat la idea de no fer un producte concret en una aplicació j2ee i en costava fer aquesta abstracció a un nivell superior per tal de estudiar i resoldre els problemes que ens troben de cara al desenvolupament d'aquest tipus d' aplicacions. Va ser el meu cap de projecte qui en la primera tutoria va fer-me la reflexió de que aquest tipus de projectes son els que m'hauria d'enfrontar com a futur enginyer i que aquest nivell d'abstracció era bo de cara a futurs anàlisis i aplicacions en l'àmbit de j2ee.

L'estudi dels patrons j2ee va ser molt útil de cara a saber quins problemes ens trobem tots els analistes i com s'arreglen de forma genèrica. En aquest sentit el llibre "Core J2EE Patterns Best Practices and Design Strategies" que explica de forma molt clara i amb exemples com s'ha d'actuar m'ha sigut de gran utilitat. El fet d'estudiar aquest patrons facilita la idea conceptual de que disseny haurien d'utilitzar en la creació d'aplicacions en tres capes i quines son les seves responsabilitats.

Ja entrant en matèria i després de l'estudi de patrons, el fet de tenir que estudiar els productes de mercat que ja resolen les problemàtiques derivades de cada una de les capes d'una aplicació de client prim va posar de manifest conèixer eines que encara que en sonaven, no coneixia: aquestes eines son els frameworks.

No tan sols va ser necessari conèixer els frameworks de la capa de presentació a la qual esta dedicat aquest projecte. A mes a mes i, sense entrar en profunditat, també era una bona oportunitat per estudiar un framework de persistència que esta ben instaurat al mercat (hibernate).

Es a dir, que amb l'estudi de tots els frameworks de presentació i algun de persistència m'ha enriquit amb molts coneixements que en assignatures de la carrera no s'havia pogut aprofundir. Es tractava de gaudir del projecte i encara que el temps era just per el desenvolupament valia la pena per el nivell de coneixements que podia aportar.

L'estudi dels frameworks de mercat m'ha implicat una corba d'aprenentatge molt alta: esta clar que el domini en profunditat de qualsevol dels productes tractats a la fase d'anàlisi necessiten de molt mes temps de el tractat en aquest projecte. L'aproximació inicial a cada producte en va servir per conèixer la forma de resoldre els problemes de la capa de presentació i quina era la arquitectura utilitzada.

Queda clar, que a causa d'aquesta corba d'aprenentatge la configuració de un framework en aplicacions molt senzilles per part d'un equip de desenvolupament no esta sempre justificada i que necessita d'una formació del equip. Per tant, es responsabilitat del cap de projecte la utilització o no de frameworks en funció de la mida, les necessitats de l'aplicació i la formació de l'equip de desenvolupament.

Una vegada estudiat els frameworks de mercat de la capa de presentació es tractava desenvolupar un propi amb les eines que teníem de java: aquest fet no tractava de millorar un producte ja fet del mercat, ni tan sols fer una aproximació perfecte a algun d'aquest productes... es tractava de fer, amb les limitacions del temps i recursos, un producte que amb la base dels patrons estudiats reforces la idea de les funcions que son necessàries en tot framework de presentació: funcions com la centralització de les peticions en un model vista controlador, la intercepció de les peticions, la configuració senzilla per par de l'usuari mitjançant anotacions, la internacionalització...

Finalment amb el desenvolupament d'una aplicació amb el jwframework m'he adonat que amb una bona formació del framework es facilitava moltíssim la programació de la capa de presentació al programador assolint així un dels objectius del projecte de final de carrera.

El desenvolupament d'aquesta area de projecte ha sigut molt enriquidora ja que he passat de tenir un repte important difícil de aconseguir inicialment com era la abstracció i construcció d'un framework, a adquirir un bons coneixement de les eines que faciliten el desenvolupament d'aplicacions j2ee i que m'ajudaran com a futur enginyer.

Glossari

Ajax (Asynchronous JavaScript And XML): tècnica de desenvolupament web per crear aplicacions interactives (Rich Internet Applications) que s'executen a la part del client

Aplicació de client prim: son les aplicacions on la lògica del programa estan a la part del servidor i el client només fa d'interfície de la lògica del negoci.

Anotacio java: forma d' afegir metadades al codi font java que esta disponible per l'aplicació en temps de execució.

Controller: Element d'un patró Model Vista controlador que respon als esdeveniments (normalment accions de l'usuari) i que fa peticions al model i a la vista

DAO: component de programari que dona un interfície comú entre l'aplicació i un o mes dispositius d'emmagatzematge (fitxers, bases de dades...)

Dispatcher: Component que s'encarrega de donar la vista quan fa la petició el controlador en un patró model vista controlador.

EDT: Estructura detallada de treball de les tasques d'un projecte, on queda clar quins terminis i productes es donaran en cada tasca.

EL (Expression Language): Llenguatge de scripts que permet accedir a components de java a través de la jsp.

Framework: Estructura conceptual i tecnològica normalment creada a base d'artefactes i mòduls de software concrets en base a que qualsevol altre projecte de programari pugui ser mes fàcilment organitzat i desenvolupat.

Freemaker: mòdul java que permet crear un motor de plantilles per aplicacions web. Les plantilles segueixen una extensió .flt

Hibernate: framework java de la capa de dades per mapejar objectes relacionals de una base de dades i el model d'objectes d'una aplicació mitjançant fitxers xml o anotacions.

Internacionalització i18n: procés per desenvolupar programari per tal que pugui adaptar-se a diferents idiomes i regions sense necessitat de fer canvis de enginyeria o de codi.

Javabean: Model de components java per encapsular objectes simples de java.

Jars: Extensió de fitxer que compacta un conjunt de classes java complicades (fitxers .class) juntament amb els recursos i els arxius de metadades per poder fer la distribució del programari.

Jndi (Java naming and directory interficie): permet al client d'aquest servei buscar objectes i dades mitjançat el seu nom i poder invocar-los posteriorment

Patrons: Donen solucions genèriques a les problemàtiques comuns en el desenvolupament de programari.

Singleton: patró que garanteix que una classe només s'instanciï un vegada i proporciona un únic punt d'accés global a ella.

Mapper: classe java formada per un hasmap que guarda la informació de elements i claus a través d'una taula de hashing.

View: Element del patró model vista controlador que dona la presentació de les dades al client web.

Wars: Fitxer empaquetat de java similar a jars que serveix per distribuir una aplicació web amb tots els seus components com pagines jsps, recursos, classes java....

Bibliografia utilitzada

- Estudi de patrons:

  http://java.sun.com/blueprints/patterns/MVC-detailed.html
  http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html
  http://java.sun.com/developer/technicalArticles/J2EE/J2EEevolution/

  Core J2EE Patterns Best Practices and Design Strategies
  Sun Microsystem Press
  ISBN 0-13-142246-4

- Struts:

  http://struts.apache.org/
  http://www.roseindia.net/struts/
  http://www.vaannila.com/struts/struts-tutorial/struts-tutorial.html

  Programming Jakarta Struts
  Chuck Cavaness
  O'Really
  ISBN: 0-596-00651-9

- JSF:

  http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
  http://docs.oracle.com/javaee/5/tutorial/doc/bnaph.html
  http://www.sicuma.uma.es/sicuma/Formacion/documentacion/JSF.pdf

- Spring:

  http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/overview.html
  http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-introduction

- TagLibraries:

  http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=jspavanzado
  http://java.sun.com/products/jsp/tutorial/TagLibrariesTOC.html
  http://www.programacion.com/articulo/jsp_custom_tags_etiquetas_a_medida_133

- Hibernate:

  http://netbeans.org/kb/docs/web/hibernate-webapp.html
  https://community.jboss.org/wiki/GenericDataAccessObjects
  http://nileshk.com/2006/05/09/hibernate-dao-using-java-generics.html

- Java:

  http://pdf.coreservlets.com/
  http://java.sun.com/developer/technicalArticles/ALT/Reflection/
  http://www.javahispano.org/storage/contenidos/reflection.pdf

Java2: Inteficies gráficas y aplicaciones para Internet
Francisco Javier Ceballos Sierra
Ra-Ma, cop 2008
ISBN: 9788478978595