Estudis d'Informàtica i Multimèdia

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

Estudiant : Miquel Vaquer Menorca

Enginyeria en Informàtica

Consultor : Josep Maria Camps Riba

Palma de Mallorca, Juny 2013

Índex

Índex de figures

Índex de Taules

Capítol 1. Introducció

1.1 Descripció del Projecte de Final de Carrera

Aquest projecte de final de carrera consisteix en el disseny i la implementació d'un marc de treball (*Framework*) que s'utilitzarà per confeccionar la capa de presentació d'una aplicació web *J2EE*; cada un dels components proporcionarà l'abstracció d'algun concepte, a més han d'esser reutilitzables i basats en tecnologies estàndard com *Servlets, JSPs, JavaBeans, XML*, etc...

Un *Framework Web* és un conjunt de components (classes *java*, descriptors i arxius de configuració *XML*) amb un disseny reutilitzable que facilita i agilitza el desenvolupament d'aplicacions *web*.

Els objectius principals d'un *Framework web* són els següents : Accelerar el procés de desenvolupament, reutilitzar codi existent i esser un model de bones pràctiques com l'ús de patrons.

Es realitzarà l'estudi del patró Model-Vista-Controlador (**MVC**), la majoria de *Frameworks Web* ofereixen una capa de controladors seguint el disseny d'aquest patró, facilitant la integració amb altres eines per a la implementació de les capes de presentació i de negoci. Aquest patró separa el Model de Negoci de les vistes (interfícies d'usuari) i del Controlador (Accions).

A continuació es farà una avaluació dels diferents *Frameworks* existents al mercat que implementen el patró MVC per a la plataforma J2EE, quines funcionalitats presenten i quines característiques tenen en comú. Un dels *Frameworks* que destaquen i que són més utilitzats al mercat és **Struts**, el qual serà tema d'estudi en aquest document.

El pròxim pas és la part més important del projecte, s'ha de realitzar el disseny i la implementació d'un *Framework* propi d'aquest estil que implementi les funcionalitats bàsiques. Precisament es fa l'estudi dels diferents *Frameworks* que hi ha al mercat per a conèixer com funcionen i tenir una bona referència per a dissenyar el nostre propi.

El *Framework* dissenyat, estarà format per un conjunt de classes o components i cada un d'ells proporcionarà l'abstracció d'algun concepte. Aquests components han de ser reutilitzables i basats en tecnologies estàndard com *Servlets, JSPs, JavaBeans, XML*, etc... L'aplicació ha de mostrar clarament com es desenvoluparia fent servir aquests components i com interaccionen entre ells.

La darrera tasca d'aquest projecte serà construir una aplicació d'exemple que mostri l'ús del *Framework* implementat. L'aplicació ha de mostrar clarament com es desenvoluparia fent servir aquests components i com interaccionen entre ells.

1.2 Objectius generals i específics

Com a objectius principals o generals d'aquest Projecte de Final de Carrera, en primer lloc s'haurà d'aprofundir molt en els nostres coneixements de *J2EE*. Per realitzar aquesta tasca hi haurà d'haver una feina molt profunda de investigació en la matèria escollida que té que veure amb totes les tecnologies implicades en els *Frameworks* de la capa de Presentació.

I no només adquirir coneixements, si no que s'haurà d'anar més enllà i a partir de l'aprenentatge que s'haurà obtingut, aporta un disseny de *Framework* propi de la capa de Presentació amb possibles millores, utilitzant els patrons *J2EE* que es considerin més adequats.

Més específicament, en aquest projecte destaquen alguns aspectes més importants i que serveixen per enunciar aquests objectius específics :

- **Tecnologies per a la implementació d'aplicacions que están lligades a J2EE** : Com *JSP (Java Server Pages)*, *EJB (Enterprise Java Beans)*, *Servlets*, *XML*,... i que també s'utilitzaran per al desenvolupament del present projecte. Per tant, també seran objecte d'estudi i implementació.

- **Patrons de disseny J2EE** : Punt molt important i que més dificultat portarà a aquest projecte serà l'estudi d'alternatives possibles per triar l'esquema de patrons pel Framework de presentació. S'hauran d'estudiar els patrons i estratègies de la capa de presentació. Alguns d'ells són per exemple : *Intercepting Filter*, *Front Controller*, *Application Controller*, *Service To Worker*, *Composite View*, etc.

- **Model MVC (Model Vista Controlador)** : És el patró o model de referència per aplicacions web, per suposat que serà el model de referència d'aquest projecte i serà objecte d'estudi i implementació.

- **Disseny i implementació d'un Framework propi** : Objectiu final d'aquest projecte, una vegada estudiats a fons els punts anteriors, s'haurà de realitzar l'anàlisi, disseny i implementació d'un Framework propi amb la finalitat de poder esser utilitzat pels desenvolupadors de programari i que faciliti la tasca de implementació de la capa de presentació a una aplicació web.

- **Implementació d'una aplicació de test del Framework**: Una vegada acabada la implementació del *Framework*, el darrer objectiu serà poder testejar el *Framework* i que funcioni correctament.

1.3 Pla de treball amb fites i temporització

A continuació es mostra la planificació del projecte de final de carrera, la qual ve donada pel el calendari indicat en l'assignatura. En primer lloc, es mostren les diferents dates d'entrega i planificació per cada marc temporal; i en segon lloc s'adjunta un *diagrama de Gantt* de la planificació del present projecte.

1.3.1 Pla de treball (27 Febrer 2013 – 11 Març 2013)

Contingut :

- ✓ Descripció del PFC
- ✓ Objectius generals i específics
- ✓ Fites i temporització

Documents a entregar :

- ✓ PAC 1 : Pla de treball (*mvaquerme_pladetreball.pdf*)

1.3.2 PAC 2 : Anàlisi i Disseny (12 Març 2013 – 15 Abril 2013)

Contingut :

- ✓ Concepte de patró
- ✓ Catàleg de patrons *J2EE* de la capa de presentació
- ✓ El patró Model-Vista-Controlador
- ✓ Concepte de *Framework*
- ✓ Estudi dels *Frameworks* destacats al mercat actual (*Struts, Struts2, Java Server Faces (JSF), Spring, Google Web Toolkit (GWT)*).
- ✓ Anàlisi i Disseny del *Framework* propi

Documents a entregar :

- ✓ PAC 2 : Anàlisi i disseny (*mvaquerme_pac2.pdf*).

1.3.3. PAC 3 : Implementació Framework (16 Abril 2013 - 3 Juny 2013)

Contingut :

- ✓ Requisits
- ✓ Implementació del *Framework*
- ✓ Implementació de l'aplicació de test

Documents a entregar :

- ✓ PAC 3 : Implementació *Framework* (*mvaquerme_pac3.pdf*).

1.3.4 Memòria i Presentació (4 Juny 2013 - 17 Juny 2013)

Contingut :

- ✓ Confecció de la documentació (Memòria del projecte)
- ✓ Confecció del document de la presentació (format diapositives)

1.3.5 Entrega del projecte (17 Juny 2013)

Documents a entregar :

- ✓ Memòria projecte (*mvaquerme_memoria.pdf*).
- ✓ Presentació del projecte (*mvaquerme_presentacio.pdf*)
- ✓ Codi font *java Framework* (Projecte *Eclipse*)
- ✓ Codi font *java* aplicació test (Projecte *Eclipse)*

A continuació es mostra el diagrama de Gantt corresponent a la planificació del projecte

Nombre	Duración	Inicio	Terminado
Projecte de final de carrera	79 days	**27/02/13 8:00**	**17/06/13 17:00**
PAC 1 : Pla de treball	9 days	27/02/13 8:00	11/03/13 17:00
PAC 2 : Anàlisi i Disseny	25 days	**12/03/13 8:00**	**15/04/13 17:00**
Estudi dels Frameworks al mercat actual	7 days	12/03/13 8:00	20/03/13 17:00
Anàlisi i Disseny del Framework propi	18 days	21/03/13 8:00	15/04/13 17:00
PAC 3	35 days	**16/04/13 8:00**	**3/06/13 17:00**
Implementació del Framework propi	28 days	16/04/13 8:00	23/05/13 17:00
Implementació aplicatiu test Framework	7 days	24/05/13 8:00	3/06/13 17:00
Memòria i Presentació	10 days	4/06/13 8:00	17/06/13 17:00
Entrega Projecte	1 day	17/06/13 7:00	17/06/13 17:00

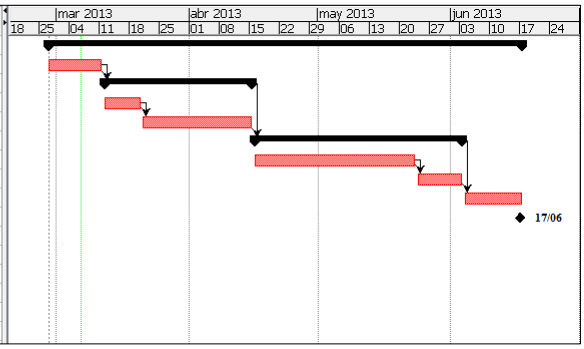


Figura 1: Diagrama de Gantt

Capítol 2. Estudi previ Frameworks J2EE

2.1 Concepte de Patró

En el camp de l'Enginyeria del Programari, un patró és una solució ja provada i aplicable a un Problema, que es presenta una vegada i una altra en el desenvolupament de diferents aplicacions i en diferents contextos. És important destacar que, un patró no és en general una solució en forma de codi directament "llest per a usar", sinó més aviat una descripció de com resoldre el problema i a quines circumstàncies és aplicable.

La utilització dels Patrons de Disseny en el desenvolupament d'aplicacions, millora la seva qualitat, centralitzant i encapsulant alguns mecanismes com per exemple, serveis de seguretat, recuperació de continguts i navegació, fent l'aplicació molt més mantenible i senzilla.

Amb l'aparició del *J2EE*, varen aparèixer tot un nou catàleg de patrons de disseny. Des que J2EE és una arquitectura, per sí mateixa, involucra altres arquitectures, incloent *Servlets*, *JavaServer Pages*, *Enterprise JavaBeans*, etc, mereix el seu propi conjunt de patrons específics per a diferents aplicacions empresarials.

2.2 Patrons de disseny J2EE

Els patrons de disseny de *J2EE* enunciats per *Sun Microsystems*, estan dividits en capes (presentació, negoci, integració) segons la seva funcionalitat. Els patrons de la capa de presentació, contenen tota la informació relacionada amb els *Servlets* i tecnologia *JSP*. Els patrons de la capa de Lògica de Negoci, contenen tota la informació relacionada amb els *Enterprise Java Beans (EJB)*. Finalment, els patrons de la capa de Integració, contenen tota la informació relacionada amb el *Java Message Service* i la tecnologia per a connexió amb base de dades de *Java JDBC*.

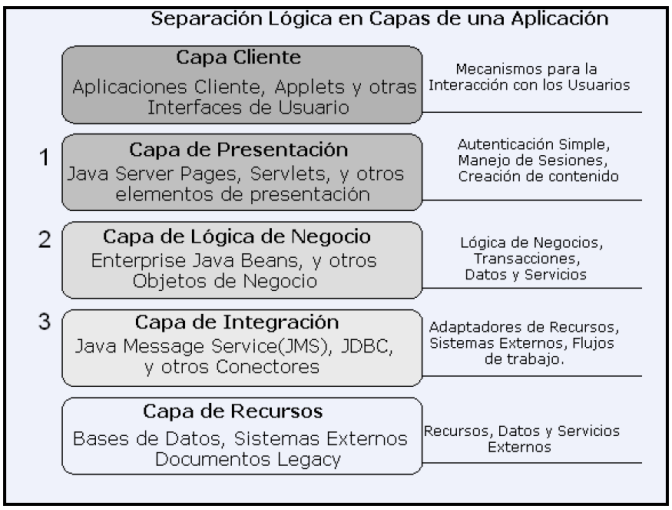


Figura 2: Distribució per capes d'una aplicació

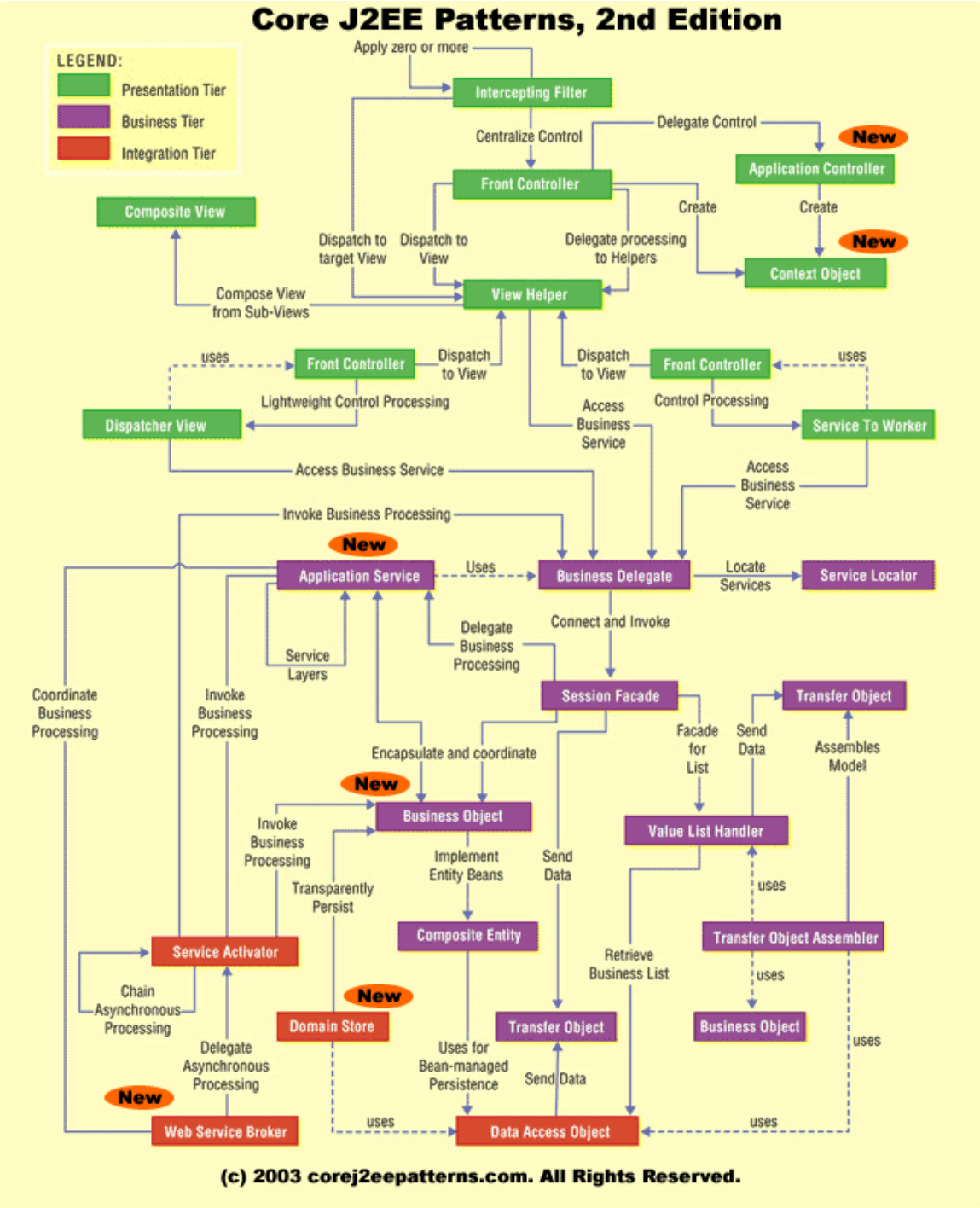


Figura 3: Patrons de disseny J2EE

2.3 Catàleg de Patrons J2EE capa de Presentació

2.3.1 Patró Intercepting Filter

La majoria dels sistemes existents a Internet manegen sol·licituds web. El processament d'aquestes sol·licituds involucra un nombre variable de serveis en el sistema.

El problema central en el qual s'enfoca aquest patró, és el **preprocessament i post processament de peticions**, per la qual cosa s'ha de proveir un mecanisme que permeti afegir i remoure components que serveixin per realitzar el processament de les peticions que arriben al sistema.

La idea principal d'aquest patró, és la de crear una sèrie de filtres que es connectin entre si, per realitzar el processament de serveis comuns del sistema en una forma estàndard, sense la necessitat de canviar el codi que es troba en el nucli de l'aplicació, per al processament de sol·licituds.

Amb això, es pretén "decorar" el processament principal amb una varietat de serveis comuns, com ara seguretat, registre de *logs*, depuració, etc. Aquests filtres són components independents del codi central de l'aplicació, la qual cosa permet que puguin ser remoguts o afegits fàcilment.

Aquest patró ens permet modificar la sol·licitud abans de ser lliurada pel controlador corresponent i modificar la resposta abans de ser lliurada al client. Entre els possibles usos dels filtres podem destacar:

- Registrar informació important sobre cada sol·licitud
- Autenticar usuaris
- Transformar les dades d'entrada abans del processament
- Transformar les dades de la resposta per a un dispositiu específic
- Validar les dades del formulari i interrompre el processament si fos necessari

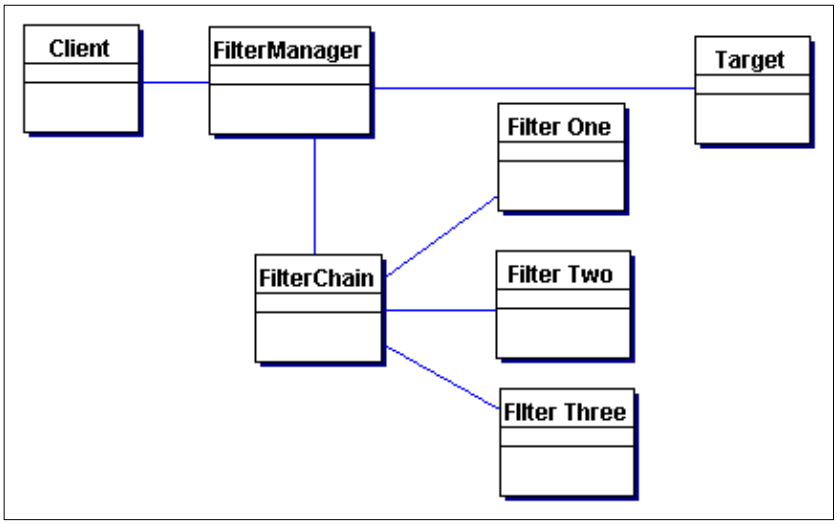


Figura 4: Diagrama de classes patró Intercepting Filter

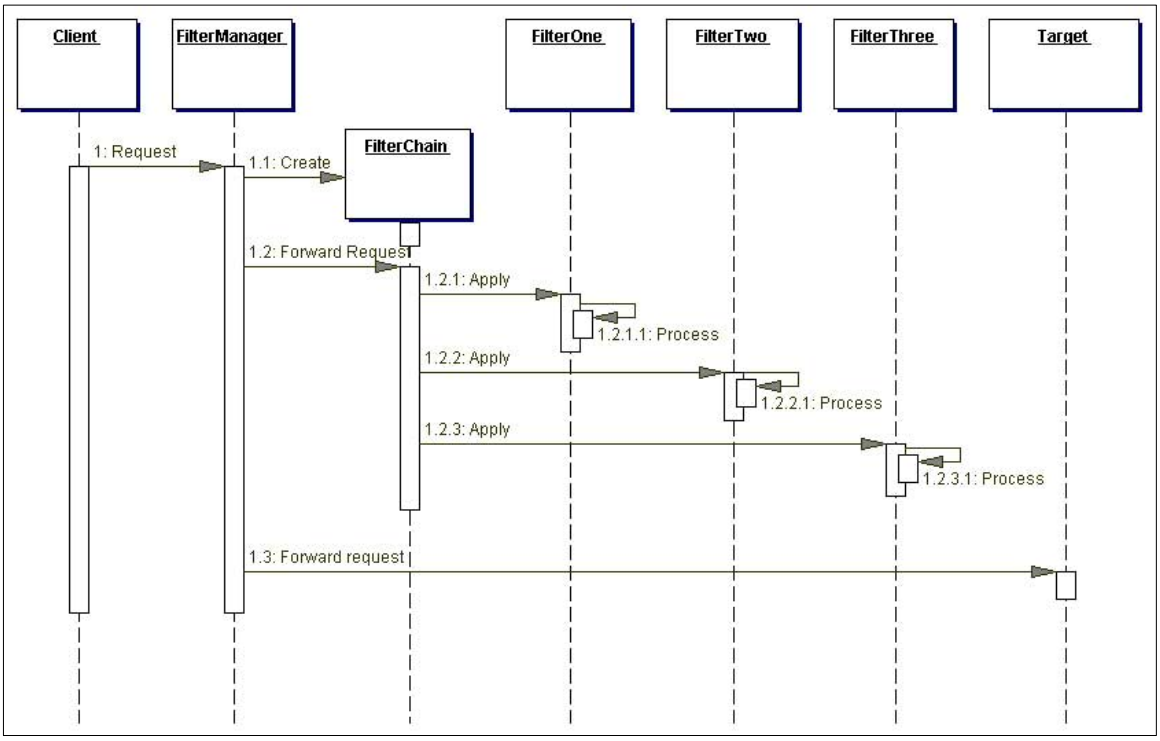


Figura 5: Diagrama de seqüència Intercepting Filter

2.3.2 Patró Front Controller

Aquest patró sorgeix amb la finalitat de proveir un mecanisme que permeti a la capa de presentació controlar i coordinar el processament de cada usuari al llarg de múltiples sol·licituds *web*, a més de permetre administrar aquests mecanismes d'una forma centralitzada o distribuïda.

La solució d'aquest patró, és utilitzar un controlador com el punt inicial de contacte per al maneig d'una sol·licitud. El controlador administra la sol·licitud, incloent els serveis de seguretat tals com autenticació i autorització, delega processos de lògica de negocis així com també la selecció d'una vista apropiada, maneig d'errors i estratègies de creació de continguts.

El controlador proveeix un punt d'entrada centralitzat que controla i administra el maneig de sol·licituds *web*, mitjançant controls i punts de decisió centralitzats, el controlador, a més ajuda a reduir la quantitat de codi *Java*, que es troba contingut dins d'una pàgina *JSP*.

Típicament, un controlador es coordina amb un component despatxador (*dispatcher*). Els despatxadors són responsables del maneig de les vistes i de la navegació. Així, un despatxador escull la pròxima vista per l'usuari i els vectors de control per al recurs. Poden estar encapsulats dins el controlador directament o poden ser col·locats en un component diferent.

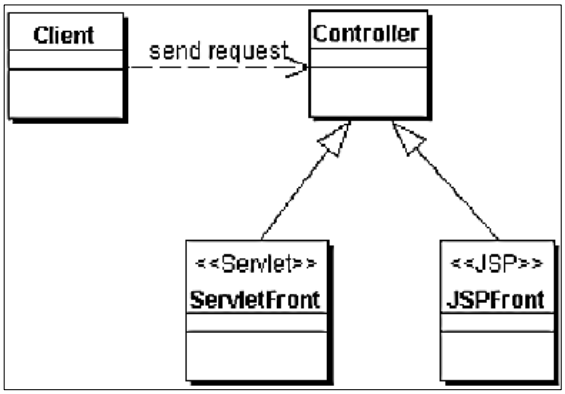


Figura 6: Diagrama de classes patró Front Controller

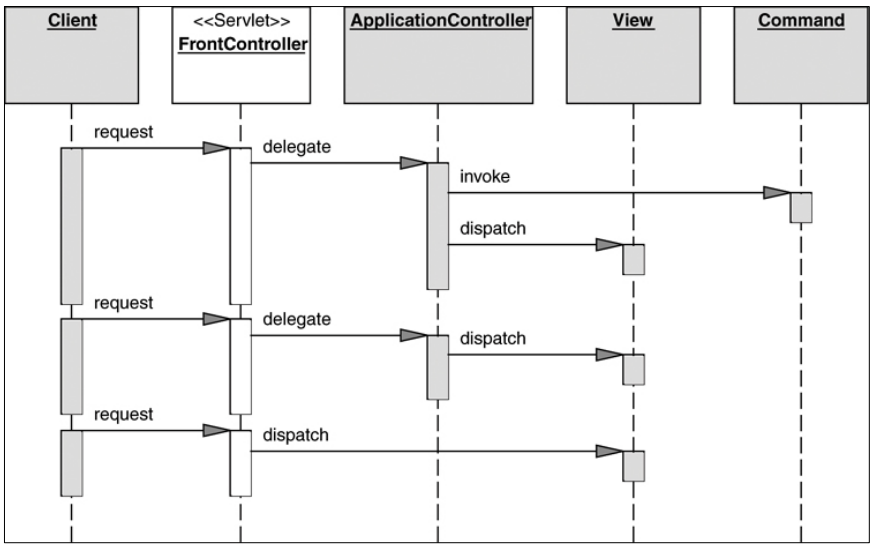


Figura 7: Diagrama de seqüència Front Controller

2.3.3 Patró View Helper

Els sistemes manegen sol·licituds *web*, per tant els processos de la capa de presentació requereixen la generació d'una vista basada en una plantilla i un model dinàmic.

És freqüent fer canvis en la capa de presentació d'una aplicació, i això és difícil de fer quan la lògica d'accés a dades de negoci, la lògica de formateig de la presentació està barrejada en aquesta capa.

La solució que presenta aquest patró, consisteix en que una vista contengui el codi de format, delegant les seves responsabilitats de processament a les seves classes *helper*, implementades generalment com *JavaBeans* (o *Custom Tags*). Els *helpers* també guarden els models de dades intermedis de la vista i serveixen com adaptadors de les dades de negoci.

Existeixen múltiples estratègies per a la implementació del component de vistes. Una de les estratègies més comunes és la d'utilitzar una pàgina *JSP* com el component de vistes. Una altra estratègia molt utilitzada és l'anomenada *ServletViewStrategy*, la qual utilitza un *servlet* com a vista.

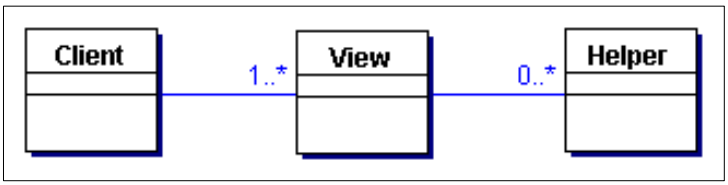


Figura 8: Diagrama de classes patró View Helper

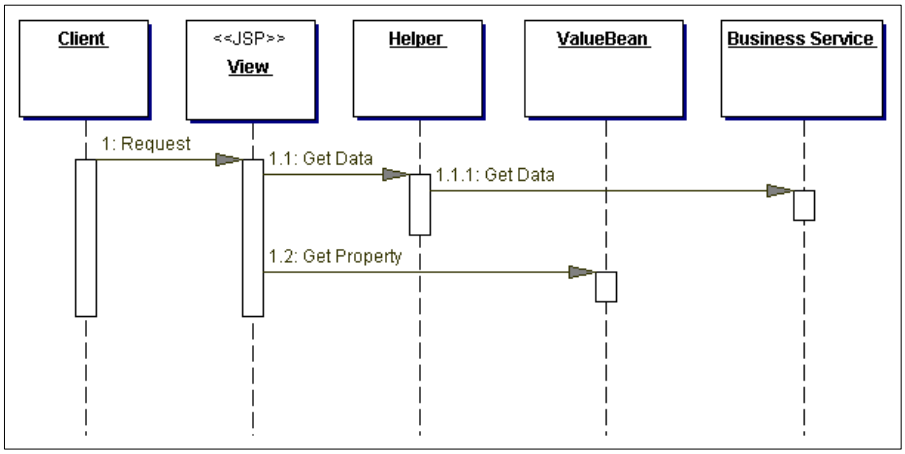


Figura 9: Diagrama de seqüència View Helper

2.3.4 Patró Composite View

La finalitat d'aquest patró, és crear vistes compostes de diverses subvistes de forma modular, flexible i extensible per construir vistes de pàgines *JSP* per a aplicacions *J2EE*.

Quan un usuari navega a través d'aplicacions gràfiques, les dades i el contingut entre les diferents pàgines varia, però molts elements, com una capçalera comuna o una barra lateral, romanen intactes en totes les vistes. L'estructura i disposició de cada pàgina pot ser la mateixa i alguns elements o seccions de la pàgina poden aparèixer en diverses pàgines diferents. Quan aquests elements i grups es codifiquen directament en l'aplicació, es torna molt difícil la tasca de modificar les vistes i es pot incórrer en inconsistències.

Les pàgines *web* sofisticades presenten contingut de diverses fonts de dades, utilitzant una vista composta (*Composite View*) formada per altres subvistes. Qualsevol canvi realitzat en una subvista, és reflectit automàticament a cada vista composta que la utilitzi.

Composite View també maneja la disposició de les seves subvistes i proporciona una plantilla, donant una aparença consistent i facilitats a l'hora de modificar-la i mantenir-la al llarg de tota l'aplicació.

Per tant, es pot utilitzar aquest patró quan:

- Diverses vistes compostes utilitzen subvistes similars.
- Les porcions atòmiques del contingut d'una vista canvien amb freqüència.

Un *Composite View,* es pot implementar amb una estrategia *JSP page View,* o be, amb l'estrategia *Servlet View*, que utilitza un *servlet* com a vista.

Aquest patró, té desavantatges, una de les més evidents, és la de la possible sobrecàrrega d'execució produïda per l'increment de flexibilitat que aquest patró proveeix. A més, l'ús d'una distribució d'elements sofisticada, pot generar problemes de desenvolupament, a causa del gran nombre d'artefactes que s'han de mantenir

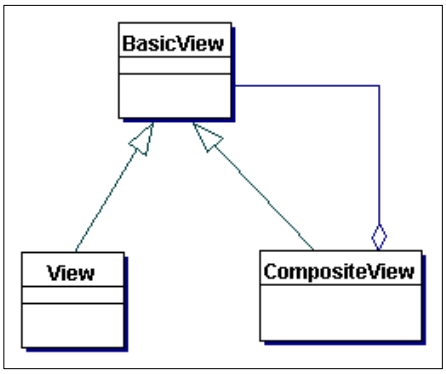


Figura 10: Diagrama de classes patró Composite View

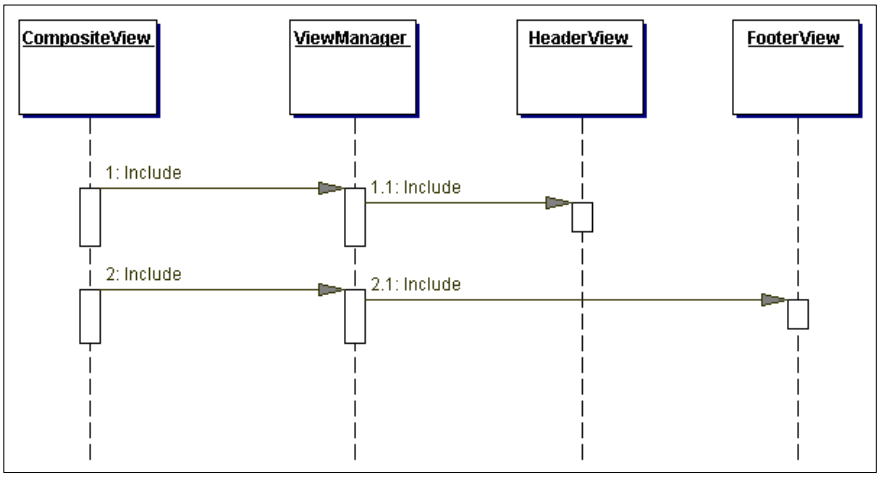


Figura 11: Diagrama de seqüència Composite View

2.3.5 Patró Service To Worker

El patró *Service to Worker*, igual que el patró *Dispatcher View*, descriu una combinació comuna d'altres patrons del catàleg. Aquests dos macro-patrons descriuen la combinació d'un controlador i un *dispatcher* amb vistes i *helpers*.

Degut a que els sistemes manegen sol·licituds *web*, els processos de la capa de presentació requereixen la generació d'una vista basada en una plantilla i un model dinàmic.

Els canvis de la capa de presentació ocorren sovint i són difícils de desenvolupar i mantenir, a causa de la combinació de la lògica de negoci per a l'accés a les dades i la lògica de format de la presentació, això fa que el sistema sigui menys flexible , menys reutilitzable, i menys adaptable als canvis.

Les porcions de lògica de negoci que són barrejades amb les vistes, han de ser adaptades a un model intermedi per a la seva visualització; Barrejar la lògica de negoci amb el processament de les vistes, també redueix la modularitat i disminueix la separació de rols a l'hora d'assignar tasques als membres d'un equip de desenvolupament de programari o d'aplicacions *web*.

L'estructura i solució d'aquest patró, consisteix en combinar un *dispatcher* amb vistes i *helpers,* per manejar les sol·licituds dels clients i preparar una presentació dinàmica com a resposta.

Un *dispatcher,* és responsable del maneig de vistes i la navegació, pot ser o no encapsulat dins d'un controlador, o a un component independent que treballi de forma sincronitzada amb els components interns.

Aquest patró i el *Dispatcher View* tenen una estructura similar, però tot i així, cada patró suggereix una divisió diferent de les tasques que ha de realitzar cada component. En aquest patró, el controlador i el *dispatcher* tenen majors responsabilitats.

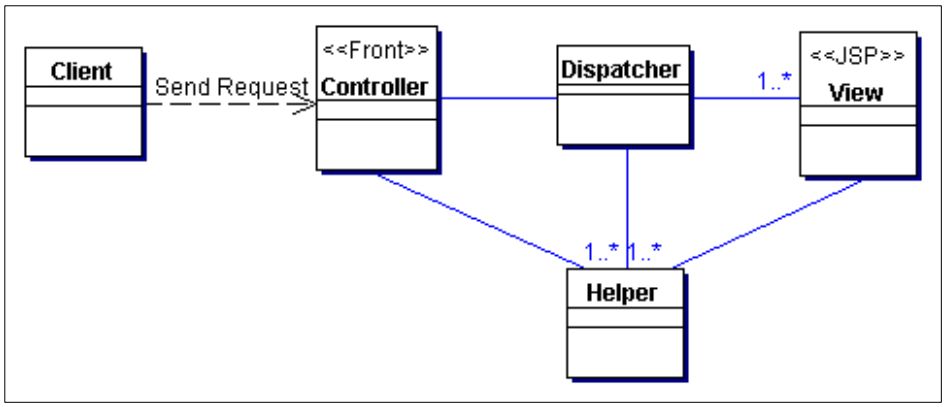


Figura 12: Diagrama de classes patró Service To Worker

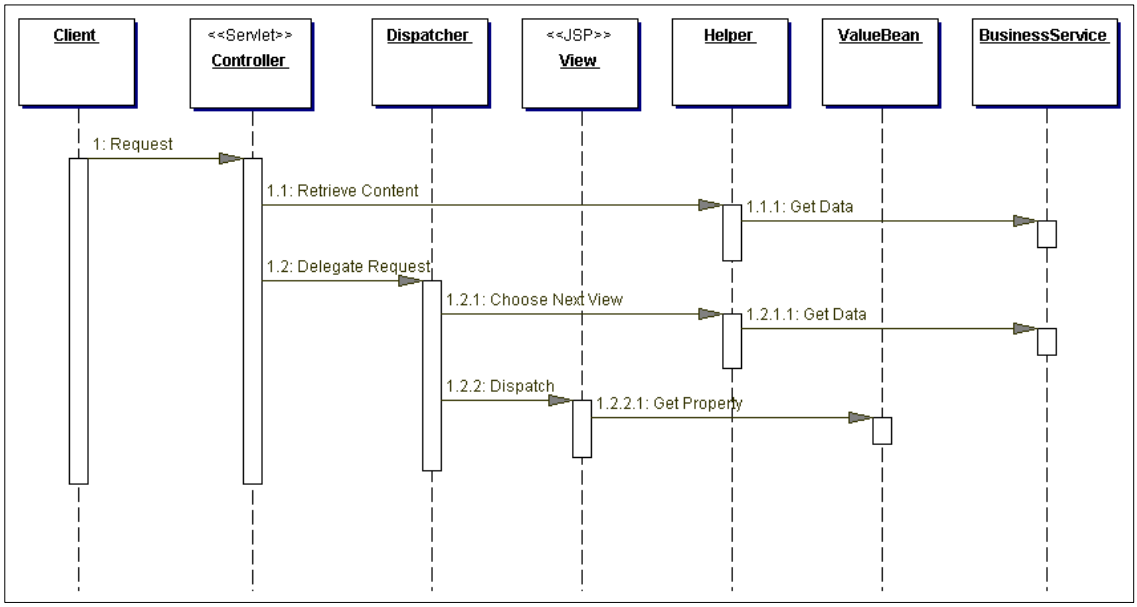


Figura 13: Diagrama de seqüència Service To Worker

2.3.6 Patró Dispatcher View

Aquest patró, combina un *dispatcher* amb *views* i *helpers* per manejar les sol·licituds dels clients i preparar una presentació dinàmica com a resposta. Un *dispatcher,* és responsable del maneig de vistes i la navegació, pot ser o no encapsulat dins d'un controlador, o pot ser un component independent que treballi de forma sincronitzada amb els components interns.

Aquest patró i el *Service to Worker,* tenen una estructura similar, tot i que cada patró suggereix una divisió diferent de les tasques que ha de realitzar cada component. El controlador i el *dispatcher,* tenen responsabilitats limitades (com es va esmentar en el patró anterior), pel fet que el processament principal i el maneig de vistes és molt bàsic. Encara més, si el control centralitzat dels recursos subjacents és considerat innecessari, llavors el controlador és remogut i el *dispatcher* pot ser mogut a una vista.

Al patró *Dispatcher View*, el *dispatcher* té un rol limitat pel que fa al maneig de vistes. Al patró *Service to Worker*, el *dispatcher* té un rol molt més rellevant pel que fa al maneig de vistes; un rol limitat per al *dispatcher,* ocorre quan no s'utilitzen recursos externs per seleccionar una vista. La informació encapsulada en la sol·licitud és suficient per determinar les vistes a utilitzar per resoldre una petició. Per exemple en aquest cas:

http://some.server.com/servlet/Controller?next=login.jsp

La responsabilitat del *dispatcher* és la de proporcionar accés a la vista '*login.jsp*'

Un exemple del *dispatcher* amb rol moderat, és el cas en el qual el client envia una petició directament a un controlador, amb un paràmetre d'una consulta que descriu una acció que ha de ser completada:

http://some.server.com/servlet/Controller?action=login

La responsabilitat del *dispatcher* en aquest cas, és la de traduir el nom lògic '*login*' al nom d'un recurs d'una vista apropiada, tal com ho és *'login.jsp'* i així proveir l'accés a aquesta vista. Per dur a terme la seva traducció, el *dispatcher* pot accedir a recursos, com un arxiu *XML* de configuració que especifiqui la vista que ha de mostrar.

D'altra banda, en el patró "*Service to Worker*", el *dispatcher* ha de ser més sofisticat, ja que aquest ha de invocar les funcions de negoci necessàries per determinar les vistes que es mostraran.

L'estructura compartida d'aquests dos patrons, que es va esmentar anteriorment, està formada per un controlador que treballa amb un *dispatcher, views i helpers*.

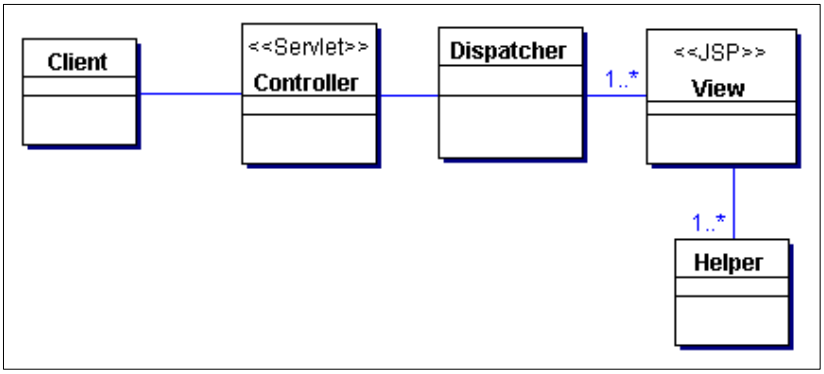


Figura 14: Diagrama de classes patró Dispatcher View

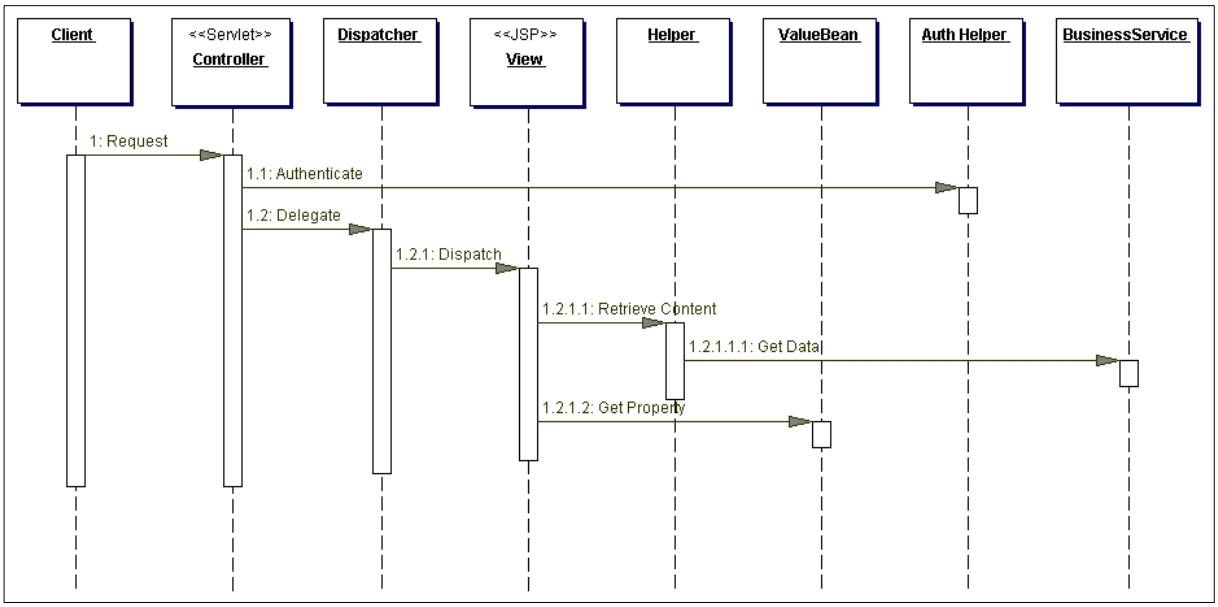


Figura 15: Diagrama de seqüència Dispatcher View

2.4. El patró Model-Vista-Controlador (MVC)

Model Vista Controlador (*MVC*) és un patró o model d'abstracció de desenvolupament de programari que separa les dades d'una aplicació, la interfície d'usuari i la lògica de negoci en tres components diferents. La finalitat és, que les modificacions que es facin al component de la vista (*view*), no tenguin cap impacte sobre el model de dades.

2.4.1 Descripció

- **Model**: Aquesta és la representació específica de la informació amb la qual el sistema opera, conté la lògica de negoci de l'aplicació..
- **Vista:** Aquest presenta el model en un format adequat per interactuar, usualment la interfície d'usuari.
- **Controlador:** Aquest respon a esdeveniments, usualment accions de l'usuari, i invoca peticions al model i, probablement, a la vista.

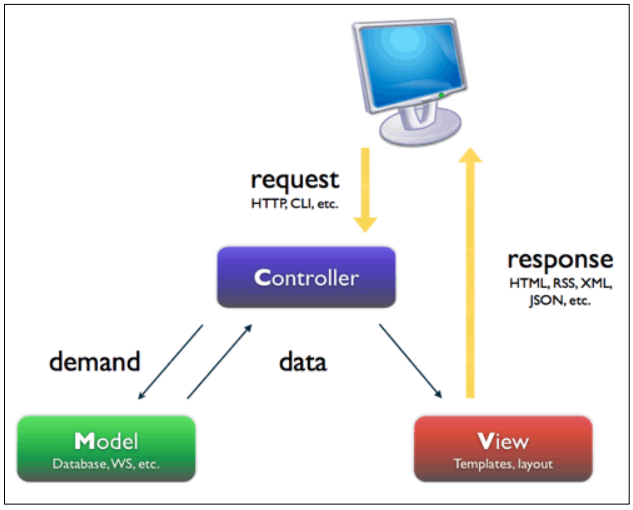


Figura 16: Patró MVC

2.4.2 MVC en aplicacions web J2EE

Les aplicacions web estan organitzades seguint una arquitectura de 3 capes, on la capa client, implementada mitjançant pàgines web, té com a missió la captura de dades de l'usuari i enviar-los a la capa intermèdia, així com la presentació de resultats procedents d'aquesta.

És la capa intermèdia, que constitueix el veritable nucli de l'aplicació web, encarregant-se del processament de les dades de l'usuari i de la generació i enviament de les respostes a la capa client. Durant aquest procés, la capa intermèdia ha d'interaccionar amb la capa de dades, per a l'emmagatzematge i recuperació de la informació manejada per l'aplicació.

Un dels esquemes i, amb tota seguretat, el més utilitzat pels desenvolupadors que utilitzen J2EE, és l'arquitectura Model Vista Controlador (*MVC*), la qual proporciona una clara separació entre les diferents responsabilitats de l'aplicació.

Quan es parla d'arquitectura Model Vista Controlador, es refereix a un patró de disseny que especifica com ha de ser estructurada una aplicació, les capes que compondran la mateixa i la funcionalitat de cadascuna.

Segons aquest patró, la capa intermèdia d'una aplicació web, pot ser dividida en tres grans blocs funcionals:

1. Controlador
2. Vista
3. Model

Controlador :Es pot dir que el controlador és el que dirigeix l'aplicació. Totes les peticions a la capa intermèdia que es realitzin des del client, són dirigides al controlador, la missió és determinar les accions a realitzar per a cadascuna d'aquestes peticions i invocar a la resta dels components de l'aplicació (Model i Vista) perquè realitzin les accions requerides en cada cas, encarregant-se també de la coordinació de tot el procés.

Avantatges

1. Fa que el desenvolupament sigui més senzill i net.
2. Facilita el posterior manteniment de l'aplicació fent-la més escalable.
3. Facilita la detecció d'errors en el codi.

En aplicacions *J2EE*, el controlador, és implementat mitjançant un s*ervlet* central que, depenent de la quantitat de tipus de peticions que ha de gestionar, es pot recolzar d'altres *servlets* auxiliars per processar cada petició.

Vista : És l'encarregada de generar les respostes (*XHTML*) que han de ser enviades al client. Quan aquesta resposta ha d'incloure dades proporcionades pel controlador, el codi *XHTML* de la pàgina, no serà fix, sinó que haurà de ser generat de forma dinàmica, pel que la seva implementació anirà a càrrec d'una pàgina *JSP*.

Les pàgines *JSP,* són molt més adequades per a la generació de les vistes dels *servlets* doncs, en ser documents de text, resulta senzilla la inclusió de blocs estàtics *XHTML* i poden ser fàcilment mantenides per dissenyadors *web* amb escassos coneixements de programació.

Quan la informació que s'enviarà és estàtica, és a dir, no depèn de dades extretes d'un emmagatzematge extern, podrà ser implementada per una pàgina o document *XHTML*.

Model : En l'arquitectura *MVC*, la lògica de negoci de l'aplicació, incloent l'accés a les dades i la seva manipulació, està encapsulada dins el model. El model, el formen una sèrie de components de negoci independents del controlador i de la vista, permetent així la seva reutilització i el desacoblament entre les capes.

En una aplicació *J2EE,* el model pot ser implementat mitjançant classes estàndard *Java* o través *d'Enterprise JavaBeans*.

2.4.3 Model 2

Model 2, és la variació de l'arquitectura *MVC*, recomanada per a les aplicacions *web* desenvolupades sobre *J2EE*. Aquest patró de disseny, utilitzat en el disseny d'aplicacions *web Java,* separa la visualització del contingut, de la lògica utilitzada per obtenir i manipular el citat contingut.

En aquest model, les peticions del navegador del client es passen al controlador. El controlador, que està format per un únic *servlet*, centralitza totes les peticions del sistema i realitza la lògica necessària per obtenir el contingut correcte per a la seva visualització, basant-se en la *URL* i l'estat actual del sistema. A continuació, col·loca el contingut a la petició (*comunment* en la forma d'un *JavaBean* o *POJO*) i decideix a quina vista passarà la petició.

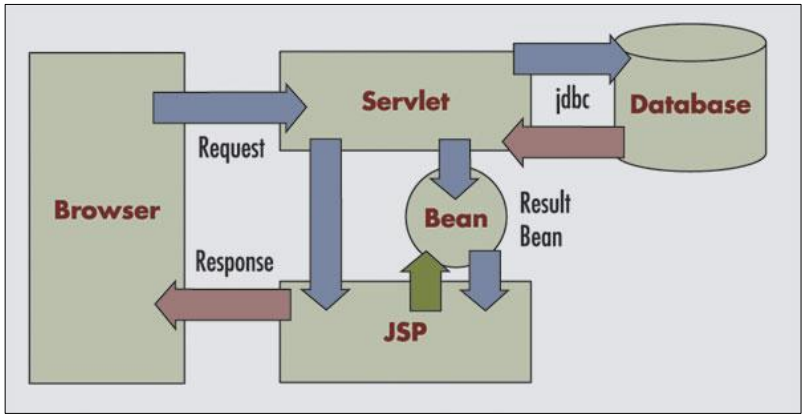


Figura 17: Patró MVC Model-2

2.5. Frameworks

2.5.1 Concepte de Framework

En el desenvolupament de programari, un *framework* és una estructura de suport definida, en la qual un projecte de programari pot ser organitzat i desenvolupat. Típicament, un *framework* pot incloure suport de programes, biblioteques i un llenguatge de *scripting* per ajudar a desenvolupar i unir els diferents components d'un projecte.

En general, amb el terme *framework*, es refereix a una estructura de programari composta de components personalitzables i intercanviables per al desenvolupament d'una aplicació. En altres paraules, es pot considerar com una aplicació genèrica incompleta i configurable, a la que es poden afegir les últimes peces per construir una aplicació concreta.

Els objectius principals que persegueix un *framework* són: accelerar el procés de desenvolupament, reutilitzar codi ja existent i promoure bones pràctiques de desenvolupament com l'ús de patrons. Un *framework web* es pot definir com un conjunt de components (per exemple classes en *java,* descriptors i arxius de configuració en *XML*) que componen un disseny reutilitzable que facilita i agilitza el desenvolupament de sistemes *web*.

2.5.2 Framework Struts

Introducció

Quan es van inventar els *servlets Java*, molts programadors es van donar compte, de que eren més ràpids, més potents, més portables i més extensibles que el *CGI* estàndard. Però, escriure infinites sentències *println()* per enviar *HTML* al navegador era problemàtic. La resposta van ser les *Java Server Pages(JSP)*, les quals deixen escriure *servlets* dins d'elles.

Ara, els desenvolupadors podien barrejar fàcilment *HTML* amb codi *Java*, i tenir tots els avantatges dels *servlets*. Les aplicacions *web Java* es van centrar en *JSP*, això, per sí sol no era dolent, però no es feia res per resoldre problemes de control de flux i altres problemes propis de les aplicacions *web*.

Clarament es necessitava un altre model, alguns desenvolupadors es van adonar que les *Java Server Pages* i els *servlets* es podrien utilitzar junts per desplegar aplicacions *web*. Els *servlets* podrien ajudar amb el control de flux, i les *JPSs* podrien enfocar-se en escriure *HTML*.

Utilitzar *JSPs* i *servlets* junts, es coneix com el *Model-2* (en el *Model-1* només s'utilitzaven *JSPs*). El projecte *Struts,* el va llançar al maig del 2000, *Craig R. McClanahan* per a proporcionar un marc de treball *MVC* estàndard a la comunitat *Java*.

Qué és struts?

És un *framework* de la capa de presentació, que implementa el patró *MVC* en *Java*. Evidentment, com tot *framework,* intenta simplificar notablement la implementació d'una arquitectura segons el patró *MVC*. *Struts* separa molt be la gestió del flux de treball de l'aplicació, del model d'objectes de negoci i de la generació d'interfície.

Struts implementa el patró *MVC,* per tant, ha de donar accessibilitat a un controlador, al model i a la vista; el controlador ja es troba implementat per *Struts*, encara que si fos necessari pot heretar i ampliar o modificar, i el flux de la aplicació es pot programar des d'un arxiu *XML*.

Las accions que s'executaran sobre el model d'objectes de negoci, s'implementen basant-se en classes predefinides pel *framework*. La generació d'interfície, es suporta mitjançant un conjunt de *tags* predefinits per *Struts,* amb l'objectiu d'evitar l'ús de *Scriplets* (els trossos de codi *Java* entre "<%" i "%>"), la qual cosa genera avantatges de mantenibilitat.

Struts, separa clarament el desenvolupament d'interfície, del flux de treball i de la lògica de negoci, permetent desenvolupar les dues en paral·lel o amb personal especialitzat. També és evident, que potencia la reutilització, suport de múltiples interfícies d'usuari (*html, shtml, wml, desktop applications,* etc.) i de múltiples idiomes, localismes, etc.

Funcionament de Struts

El navegador, genera una sol·licitud que és atesa pel Controlador (un *servlet* especialitzat). El mateix s'encarrega d'analitzar la sol·licitud, seguir la configuració que se li ha programat en la seva *XML* i cridar al *Action* corresponent passant-li els paràmetres enviats. El *Action* instanciarà i/o utilitzarà els objectes de negoci per concretar la tasca. Segons el resultat que retorni el *Action*, el Controlador derivarà la generació d'interfície a una o més JSPs, les quals podran consultar els objectes del model, per mostrar la informació corresponent .

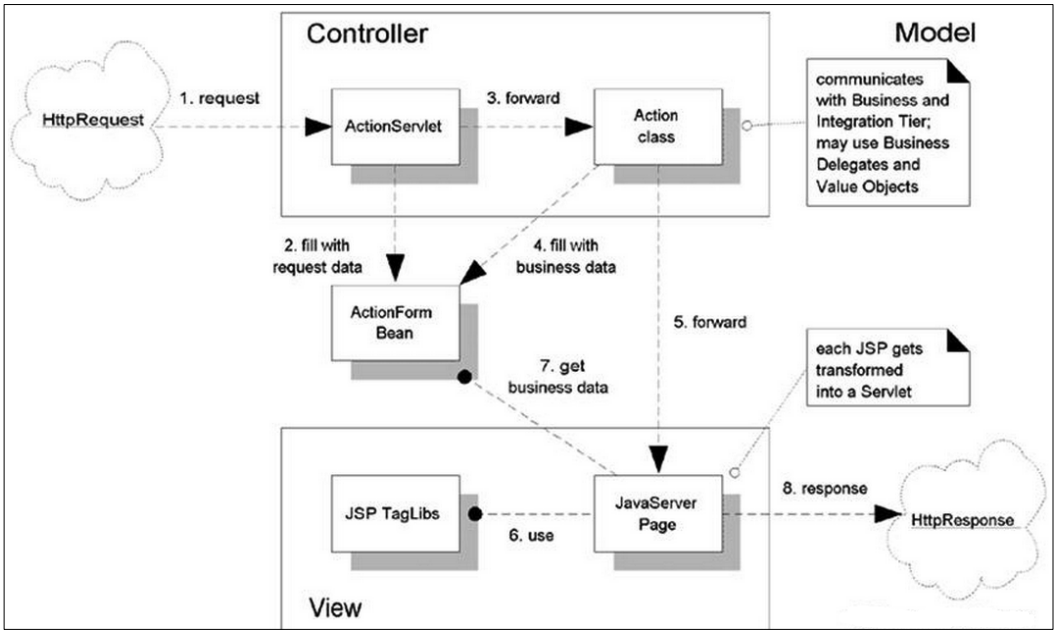


Figura 18: Funcionament del framework Struts

Capa Vista

Quan un usuari completa un formulari i l'envia, el *Controller* cerca al *scope* especificat el *ActionForm Bean* corresponent (tot això configurat al fitxer *struts-config.xml*), si no el troba, el crea. Després fa un *set* per a cada *input* del *form* i finalment crida el mètode *validate*.

Si aquest retornés un o més errors, el *Controller* cridaria al *JSP* del formulari, perquè, aquest ho tornés a generar (amb els valors establerts per l'usuari) i inclogués el o els missatges d'error corresponents. Si tot estigués bé, cridaria al mètode *execute* de *l'Action* (també configurat al *struts-config.xml*) passant-li el *ActionForm Bean* com a paràmetre perquè sigui utilitzat per obtenir els valors de les dades.

Tot i que *l'ActionForm* té característiques que corresponen al model, els *ActionForm* pertanyen a la vista. Justament un d'aquests punts comuns, és la validació de dades i amb la finalitat d'evitar la duplicació de funcionalitat.

DynaActionForm és una subclasse especialitzada de *ActionForm* que permet la creació de *ActionFormbeans* amb sets dinàmics de propietats, sense requerir que el desenvolupador creï una classe *java* per cada tipus de *form bean.*

Els elements que utilitza Struts són els següents :

- **Java Server Pages (JSP)** : Són el principal component de la capa vista, les pàgines *JSP* estan formades per codi *HTML* estàtic i llibreries d'etiquetes *JSP* que generen codi *HTML* dinàmic, les quals conjuntament són enviades al navegador, per mostrar les dades a l'usuari.
- **ActionForm Beans** : Són classes que agafen la informació recollida en els formularis i són utilitzades per la capa de control per posar aquestes dades a servei del model per al seu processament.
- **ActionForms :** Classe que permet passar les dades d'entrada cap endavant i enrere entre l'usuari i la capa de negoci.
- **Struts tags (Llibreries d'etiquetes)** : component que proporciona un conjunt de biblioteques d'etiquetes *JSP* personalitzades, que ajuden als desenvolupadors a crear formularis interactius. Existeixen etiquetes per a la visualització de missatges d'error, per tractar amb *beans* i *ActionForm.*
  Struts taglib es compon de quatre biblioteques d'etiquetes diferents:
  - *Bean* (definir nous *beans*)
  - *HTML* (crear formularis *HTML*)
  - *logic* (lògica condicional a les pàgines *JSP*)
  - *nested* (té com a principal funció encadenar etiquetes d'altres grups. Per exemple realitzar un bucle d'etiquetes per a mostrar les propietats d'una forma coherent (un empleat pot tenir més d'un càrrec)
- **ResourceBundle** : La característica de mantenir varis llenguatges diferents, està proporcionat a través de les classes *ResourceBundle*. Són fitxers de text amb extensió *.properties,* que són llegits per l'aplicació en temps d'execució; en lloc de tenir una classe *ResourceBundle* per a cada idioma suportat, es descriuen les cadenes corresponents a cada idioma amb aquests fitxers de text. Quan la classe *ResourceBundle* sigui carregada, es crearà en base a l'arxiu de propietats.

Capa Controlador

En controlador *Struts* està implementat en el *servlet* base *ActionServlet*, el qual reb totes les peticions dels clients i que està configurat en base a l'arxiu de configuració *struts-config.xml*, el qual indica les accions a realitzar.
Els components d'aquesta capa són els següents :

- **ActionServlet** : classe que s'estén de *javax.servlet.http.HttpServlet* responsable del empaquetat i encaminament del trànsit HTTP cap al gestor apropiat dins del entorn. El controlador (*org.apache.struts.action.ActionServlet*) és una subclasse genèrica de *javax.servlet.http.HttpServlet*, la seva responsabilitat consisteix en inicialitzar totes les aplicacions Struts en temps d'execució, segons el definit en l'arxiu de configuració *struts-config.xml*.
  A més, és una implementació del patró *Front Controller,* ja que centralitza en una única instància l'accés de les peticions des de la capa de Presentació i les delega a la representació *struts* del patró *Application Controller*.

- **RequestProcessor** : classe que permet desacoblar el procés de petició (*Request Process*) del *ActionServlet* i així poder modificar com es processa la petició (fent un *subclass* de *RequestProcessor*). Els *processors* (instàncies de *org.apache.struts.action.RequestProcessor*), centralitzen, modularitzen i administren els accessos als recursos d'Accions i vistes de la Capa de Presentació. D'acord amb la configuració de l'entorn *Struts*, hi haurà un *processor* per a cada subaplicació, cada una de les quals es configura de forma independent en el seu propi arxiu de configuració *struts-config.xml*.

- **Action** : classe que independitza la petició del client del model de negoci. És una extensió del component de control i permet fer funcions com autorització, validació de sessió, abans d'invocar la lògica de negoci. Les accions (subclasses de *org.apache.struts.action.Action*), són components *multithread* que proveeixen una funcionalitat cohesiva d'acord amb el patró *Service To Worker.*

  El *processor* que invoca les Accions configurades per el desenvolupador en l'arxiu *struts-config.xml*, coneix la interfície pública d'aquestes accions (mètodes, arguments d'entrada, dades de sortida). La lògica de cada una s'ha de codificar únicament sobreescrivint el mètode *execute* (...) i dependrà de la funcionalitat específica (funcional) dels passos que realitzi. Les accions serveixen per invocar els components que fan lògica de negoci per una funcionalitat particular i per retornar el control a la vista següent.

- **ActionMapping** : classe que representa una acció de mapejat, la qual es defineix al fitxer de configuració de *Struts*. Indica al controlador quina instància de *Action* s'ha de invocar a cada petició.

- **ActionForward** : classe que representa el destí al qual el controlador ha d'enviar el control una vegada que una *Action* s'ha completat (evita introduir el *forward()* a la pàgina *JSP*).

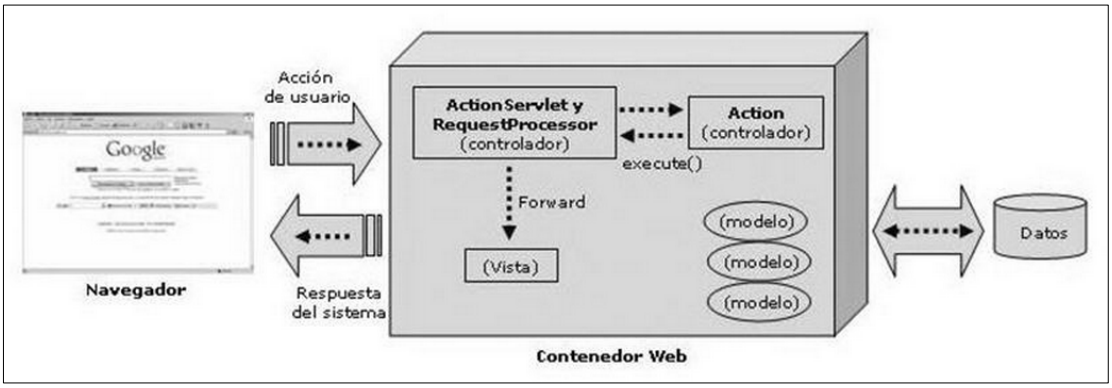


Figura 19: Capa controlador a Struts

Capa Model

Struts no va esser dissenyat per treballar amb aquesta capa, no obstant això, és obvi que *Struts* rebrà informació d'aquesta (encara que no sàpiga com està implementada). Així, la millor manera de solucionar com *Struts* es comunica amb la capa de negoci, de manera que sigui completament independent de la tecnologia de implementació d'aquesta, és utilitzar un patró de disseny, per exemple *DTOs (Data Transfer Objects*), també anomenats *Value Objects (VOs)*.

La idea és que *Struts* rebi la informació en forma de vistes (*VOs*), però no sàpiga com s'han creat aquestes. Per això, es necessitarà implementar un nou patró de disseny molt conegut, el *Business Delegate*. Amb aquest patró, es crearà un servei pertanyent a la capa de negoci, que servirà de nexe d'unió amb la capa de control i serà a través del qual *Struts* demanarà i rebrà els únics objectes que entén i amb els que sap treballar : els *VOs*.

Per altra banda, la capa de negoci només sabrà que pot rebre peticions del servei d'accés a BD a través de *JDBC* i retornarà *VOs*, sense importar-li ni qui els demana, ni qui els utilitza.

2.5.3 Framework Struts 2

Introducció

Struts 2 no es basa en el codi de *Struts 1*, sinó en el d'un altre *framework* de desenvolupament *web* en *Java* anomenat *Webwork*, un *framework* que el creador de *Struts* considerava superior a *Struts 1* en diversos aspectes.

Struts 2 està basat en el patró *MVC*, està pensat per facilitar les tasques pròpies del cicle de vida del programari, incloent la construcció, desenvolupament i manteniment de la solució. És fàcilment extensible, ja que tots els elements estan basats en interfícies i classes bases. La plataforma bàsica que requereix *Struts 2* requereix les versions *Servlet API 2.4, JSP API 2.0* i *Java 5*.

Una altra característica nova que inclou *Struts2,* és un conjunt de llibreries desenvolupades per facilitar la creació d'aplicacions web dinàmiques, facilitant la creació d'elements de client que permetin mostrar informació, validar, localitzar de manera senzilla i sense que impacti en els temps de desenvolupament, fent els desenvolupaments més senzills de mantenir.

Arquitectura

Struts 2 està concebut per ser un *framework* extensible i elegant per als desenvolupadors d'aplicacions *J2EE*. Com a primer pas, es veurà quin és el cicle de vida d'una petició en una aplicació desenvolupada sota aquesta arquitectura:

Cicle de vida d'una petició

1. **Un usuari envia una petició:** Un usuari fa la petició d'un recurs dins del servidor.
2. **L'element FilterDispatcher determina l'acció que haurà de respondre**: El *framework* disposa dels elements necessaris perquè el *dispatcher* sigui capaç de determinar quin *Action* és el responsable de rebre la petició i processar-la. Per això es recolza en el *framework* per la publicació del recurs, i per a la seva execució.
3. **S'apliquen els interceptors definits**: Existeixen diferents *interceptors* que es poden configurar perquè executin diferents funcionalitats com *workflows*, validacions, upload de fitxers, etc.
4. **S'executa el Action**: Després de l'execució dels diferents *interceptors*, el mètode específic del *Action* és executat, realitzant aquelles operacions i accions que s'hagin definit. El *Action* acaba retornant un resultat el qual s'utilitza per determinar la pàgina a retornar.
5. **Es renderitza la sortida**: Després de l'execució de l'*Action,* es determina quina és la pàgina que retorna i s'executa el *forward* cap aquesta.
6. **Es retorna la petició**: Per a realitzar la devolució s'executen els *interceptors* que corresponguin i es procedeix a retornar la petició al client. D'aquesta manera és possible afegir lògica externa als servidors també en la devolució.
7. **Es mostra el resultat al client final**: Finalment el control és retornat al client, que pot visualitzar el resultat en el seu navegador.

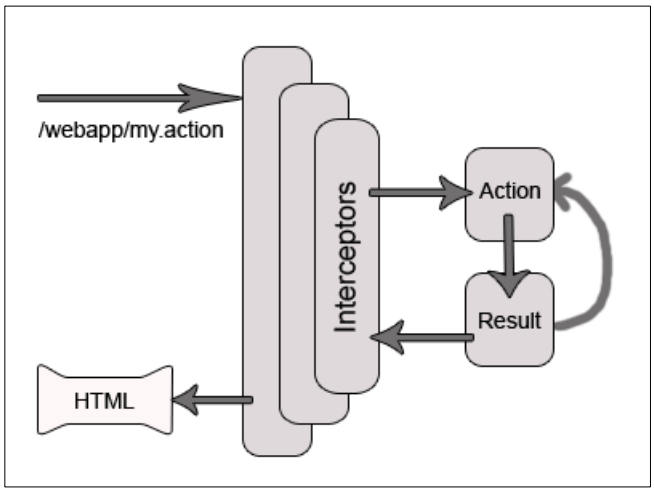


Figura 20: Cicle de vida petició Struts 2

Capa Controlador

És el responsable de rebre les peticions dels clients i a més :

1. Processar les peticions, determinant quina classe és la que ha de respondre.
2. Modificar el model en funció dels paràmetres que rep, executant la lògica de negoci que el desenvolupador hagi definit.
3. Redirigir a la vista en funció de la lògica de negoci.

Com no podia ser d'una altra manera, el controlador permet l'ús de tecnologies estàndards com ara *Java Filters, Java Beans, ResourceBundles, XML*, etc.

Capa Model

És la part responsable de la gestió de la informació. En aquest punt se solen incloure aquelles classes, eines, llibreries, etc., que permeten l'accés a les dades, així com el modelatge de les mateixes. Algunes de les tecnologies que se solen emprar són : *JDBC, EJB, Hibernate o JPA*, entre d'altres.

Capa Vista

És la responsable de la percepció que tenen els usuaris finals de la aplicació. S'inclouen les pàgines *Html, JS,* etc, i en general tots els elements de visualització de l'aplicació. Rep la informació que el controlador ha captat del model, per confeccionar les pàgines finals. Algunes de les tecnologies que s'utilitzen són: *Html, JSP, XSLT, PDF,templates*, etc.

Gestor de Excepcions

Aquest element existia en *Struts* i està pensat per definir procediments de gestió d'excepcions tant locals com globals, de manera que quan es produeix un error, es pugui definir què s'ha de fer, redirigint a pàgines genèriques d'error, etc.

Interceptors

Els *interceptors* són classes que s'utilitzen per especificar el cicle de vida d'una petició, i estan pensats per afegir funcionalitat extra als *Actions*. Podríem dir, que un *interceptor* és semblant a un filtre de *J2EE,* llevat que el seu ús és més localitzat i es pot fer dependre de *Actions* concrets i que no s'executi de manera global. Per configurar-los es fa de forma declarativa.

Els *interceptors* se solen utilitzar per realitzar validacions, *workflows*, etc ..., i quan es defineixen sobre un *Action*, s'executen com els filtres, és a dir, de forma consecutiva com una cadena.

Gestió de Configuració

El gestor de la configuració és l'element encarregat de llegir el fitxer de control (*struts.xml*), analitzar-lo i en el cas que sigui correcte, desplegar aquells elements que s'esmenten.

A continuació i per mostrar com és l'arquitectura de *Struts 2,* s'afegeix un diagrama en el qual es pot veure de forma gràfica quins elements són els que intervenen dins del processament d'una petició *HTTP*, distingint quins són elements d'estàndards de *J2EE*, quins són elements propis de *Struts 2* i quins són creats per l'usuari.

Els elements que intervenen en una petició HTTP són ;

1. Una petició arriba a un *servlet* (*HttpServletRequest*) el qual, és la part del controlador que rebrà la petició.

2. El *servlet* delega en el *ActionMapper,* per determinar quina és l'acció que es va a executar.

3. A continuació s'executa la "cadena de filtres", entre els quals destaquen :
   a. *Action Context Clean Up Filter*. Aquest filtre s'utilitza quan hi ha de realitzar una integració amb altres tecnologies (compartició de *cookies*, etc.)
   b. *FilterDispatch.* Aquest és encarregat de determinar si la resolució de la petició requereix l'execució o no d'un *Action*. S'ha de tenir en compte, que hi ha redireccions o altres que no necessiten d'un *action* que les implementi. En el cas que calgui usar un *Action*, es delega en el *Action Proxy*.

4. *Action Proxy*: s'executa quan la petició requereix l'execució d'un *Action*. El que fa és utilitzar l'Administrador de Configuracions per crear *ActionInvocation*. Aquest és una classe que implementa el patró *Command*, el qual s'executa.

5. *Action Invocation*: es llança després de llegir el fitxer de configuració, i per tant és coneixedor de quina classe s'ha d'executar, així com dels *interceptors* que intervenen. Per això, és responsable d'executar cada un dels *interceptors*, i després executarà el *Action* corresponent. Aquest *Action* contindrà la lògica de negoci que s'ha d'executar. Com a resultat de l'execució del *Action* s'obtindrà un *Result*, el qual s'utilitzarà per determinar quins dels elements de la vista s'han de cridar.
6. S'executa l'element que ha de configurar la vista i el resultat es torna, desfent el camí fins al client.

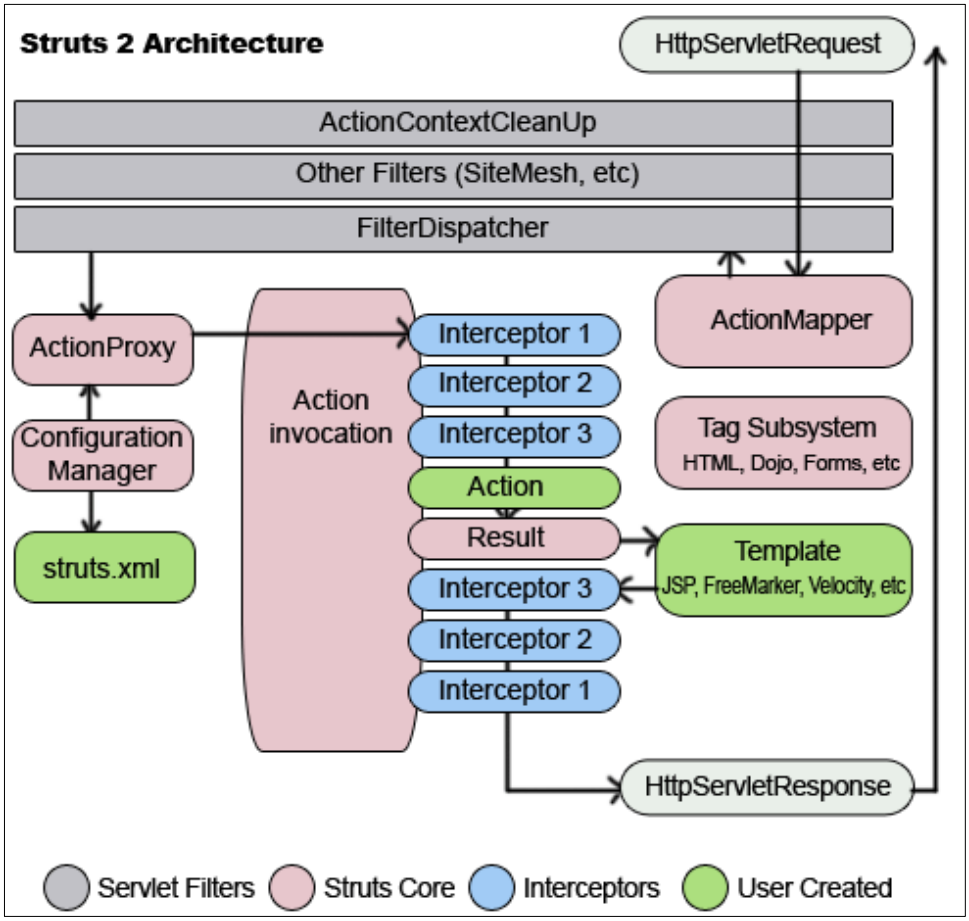


Figura 21: Arquitectura a Struts 2

Struts 1 vs Struts 2

En línies generals, es pot afirmar que *Struts 2* és més simple i eficient que *Struts 1*. No només explota millor les potencialitats de *J2EE*, sinó que també fa més fàcil el desenvolupament de noves aplicacions i la integració amb altres tecnologies i eines.

Algunes de les millores de la nova versió són:

- **Dependència del Servlet** : En la versió 1 de *Struts* hi ha una gran dependència amb *l'API Servlet*, cosa que es posa de manifest que els paràmetres del mètode *execute* necessiten els objectes *HttpServletRequest* i *HttpServletResponse*.

En *Struts1* s'ha definit així, per possibilitar l'accés a la sessió, al contexte del *servlet*, etc. En *Struts2*, els *actions* estan desacoblats del contenidor, essent el *servlet Controller* el responsable de realitzar els *set* de paràmetres. Aquesta característica fa que sigui senzill el testeig dels *actions* de forma unitària, ja que no es depèn del context del *servlet*. Això també passa amb la sessió, la qual s'estableix en *Struts2* amb el mètode *setSession*, i no amb l'objecte *Request*.

- **Classe Action** : El gran canvi que s'ha produït en el disseny dels *Actions,* és, que en comptes d'estendre classes abstractes com es feia en la versió 1, es fa utilitzant interfícies, la qual cosa salva algunes de les restriccions en quant al modelat del programari, que sorgien amb la versió anterior. A més, també hi ha l'opció de tenir accions que directament no implementen cap classe, ja que s'usa el l'*API* d'introspecció per determinar el mètode a executar.

  A més, el *Action* es pot comportar com a *JavaBean* per llegir la informació des de les pantalles, fent més fàcil realitzar les accions que siguin requerides.

- **Validacions** : Tant *Struts1* com *Struts2* permeten la validació manual amb el mètode *validate*. En el cas de *Struts1* aquest mètode és a la classe *ActionForm*, o bé fa la validació amb l'extensió del *Validator*. *Struts2* permet la validació manual amb el mètode *validate* que es troba al *Action*, o bé pot realitzar validacions usant el *framework XWork*. Aquesta tecnologia, permet fer validacions encadenades, usant regles definides per l'usuari.

- **Model de fils** : En *Struts1,* els *Actions* són sincronitzats utilitzant el patró *Singleton*. Això fa, que només una instància de classe vagi a respondre totes les peticions que li arribin a aquest *action*. En el cas de *Struts2,* s'instancia un objecte per a cada petició, amb la qual es pot tenir el cas, que el mateix codi ho executi més d'un fil.

- **Recuperació de dades d'entrada** : Per a la captura de les dades d'entrada, *Struts1* utilitza classes que s'estenen de la classe genèrica *ActionForm*. Això fa, que un *Java bean* no pugui ser utilitzat en *Struts1*, de manera que els desenvolupadors han de replicar el codi del *Bean* al *ActionForm* corresponent.

  A *Struts2* s'utilitzen propietats del *Action* amb els seus *sets* i *gets* per poder recuperar la informació. D'aquesta manera, si es té un *JavaBean*, només s'ha de col·locar al *Action* amb els seus corresponents *gets* i *sets* per poder utilitzar-los en la pantalla, evitant replicar codi.

- **Tag Libraries i Expressions de Llenguatge per JSP** : *Struts1* integra *JSTL*, de manera que es té la possibilitat d'utilitzar *JSTL-EL* (llenguatge d'expressions de *JSTL*). *Struts2* inclou no només *JSTL*, sinó també un conjunt de llibreries que permeten una integració efectiva amb les propietats dels *actions*. A més, les etiquetes permeten múltiples configuracions utilitzant els *themes*. Aquests permeten tenir diferents resultats de les mateixa etiqueta gràcies al llenguatge *OGNL (Object Graph Notation Language)*.

- **Ús de valors en les vista** : Per treballar amb les vistes, *Struts1* utilitza el mecanisme estàndard de *JSP* per vincular objectes a la propietat en el context de pàgina. Les *JSP* que requereixin accedir a altres contextos, hauran de fer-ho de forma programativa. *Struts2* utilitza una tecnologia "*ValueStack*", que permet recuperar els valors amb independència del context (sessió, aplicació, *request*) en el qual estigui guardat el valor. Així les vistes són més fàcils de reutilitzar.

- **Control de l'execució dels actions** : *Struts1* permet separar el processador de les peticions que gestiona el cicle de vida dels *actions* per a cada mòdul, és a dir, tots els *actions* dins d'un mòdul, tenen el mateix cicle de vida. En *Struts 2* es pot parametritzar a nivell de *action*, amb una granularitat major.

2.5.4 Framework Java Server Faces (JSF)

Introducció

L'objectiu de la tecnologia *Java Server Faces,* és desenvolupar aplicacions *web* de forma semblant a com es construeixen aplicacions locals amb *Java Swing*, *AWT (Abstract Window Toolkit), SWT (Standard Widget Toolkit)* o qualsevol altra *API* similar.

JavaServer Faces, pretén facilitar la construcció d'aquestes aplicacions, proporcionant un entorn de treball (*framework*) via *web*, basat en el patró *MVC*, que gestiona les accions produïdes per l'usuari a la seva pàgina *HTML* i les tradueix a esdeveniments que són enviats al servidor, amb l'objectiu de regenerar la pàgina original i reflexar els canvis pertinents provocats per aquestes accions.

En definitiva, es tracta de fer aplicacions *Java*, en les quals el client no és una finestra de la classe *JFrame* o similar, sinó una pàgina *HTML*.

Característiques principals

Els principals components de la tecnologia *JavaServer Faces* són:

- Una *API* i una implementació de referència per a representar components d'interfície d'usuari (*UI-User Interface*), manejar el seu estat, manejar esdeveniments, validar al costat del servidor i convertir dades, definir la navegació entre pàgines, suportar internacionalització i accessibilitat, i proporcionar extensibilitat per totes aquestes característiques.

- Una llibreria d'etiquetes *JavaServer Pages* (*JSP*) personalitzades per dibuixar components UI dins una pàgina *JSP*.

Aquest model de programació ben definit i la llibreria d'etiquetes per components UI, facilita de manera significativa la tasca de la construcció i manteniment d'aplicacions *web* amb UIS al costat servidor. Amb un mínim esforç, és possible:

- Connectar esdeveniments generats en el client a codi de l'aplicació, en el costat servidor.
- Mapejar components *UI* a una pàgina de dades, en el costat servidor.
- Construir una interfície d'usuari amb components reutilitzables i extensibles.

Com es pot apreciar a la figura 21, la interfície d'usuari que es crea amb la tecnologia *JavaServer Faces* (representada per *miUI* en el gràfic) s'executa al servidor i es renderitza al client.

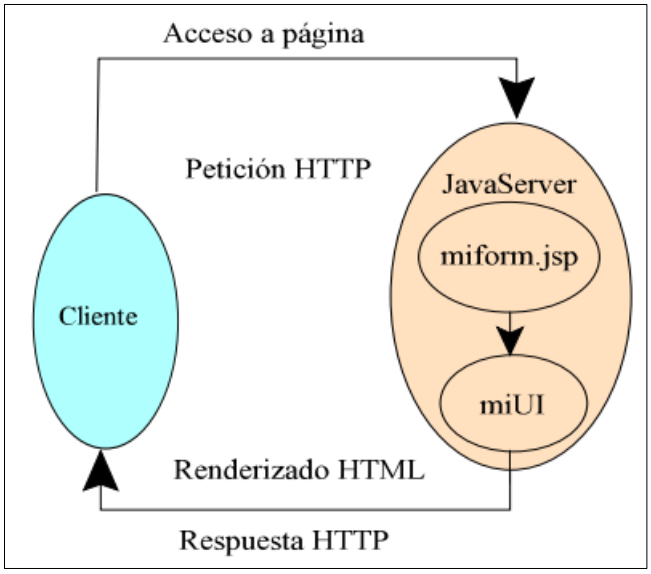


Figura 22: Diagrama aplicació JSF

La pàgina *JSP*, *miform.jsp*, especifica els components de la interfície d'usuari mitjançant etiquetes personalitzades definides per la tecnologia *JavaServer Faces*. L'*UI* de l'aplicació *web* (representada per *miUI* a la figura), maneja els objectes referenciats per la *JSP*, que poden ser dels següents tipus:

- Objectes components que mapegen les etiquetes sobre la pàgina *JSP*.
- Oients d'esdeveniments, validadors i convertidors registrats i associats als components.
- Objectes del model, que encapsulen les dades i les funcionalitats dels components específics de l'aplicació (lògica de negoci).

Comparativa amb Tecnologies similars

Igual que *Struts*, *JSF* pretén normalitzar i estandaritzar el desenvolupament d'aplicacions *web*. Cal tenir en compte que *JSF* és posterior a *Struts* i, per tant, ha eliminat algunes les seves deficiències.

JSF és molt similar a *Struts*, ja que la interfície d'usuari la implementa amb pàgines JSP, amb l'avantatge que pot treballar amb diverses eines de desenvolupament, que poden facilitar l'elaboració de la interfície de manera visual, però això és opcional, podent fer mitjançant editors de text, editant cadascun dels fitxers que componen l'aplicació.

Tanmateix, *JSF* no pot competir en maduresa amb *Struts* (en aquest punt *Struts* és clar guanyador), però sí que pot ser una opció molt recomanable per a nous desenvolupaments, sobretot, si encara no es té experiència amb *Struts*.

Capa Model

El model, és l'objecte que representa i treballa directament amb les dades del programa: gestiona les dades i controla totes les seves transformacions. El model no té coneixement específic dels diferents controladors i/o vistes, ni tan sols conté referències a ells. És el propi sistema, el que té encomanada la responsabilitat de mantenir enllaços entre el model i les seves vistes, i notificar a aquestes quan han de reflectir un canvi en el model.

Per exemple, es podria definir el model que contengui les dades de l'usuari, en el fitxer *UsuarioBean.java*, i definir la classe *UsuarioBean*, que conté els elements necessaris per gestionar les dades de l'aplicació: és necessari un camp, que emmagatzemi el nom de l'usuari que va entrar, i un altre per la corresponent contrasenya, juntament amb els seus respectius mètodes d'assignació i recuperació:

```
public class UsuarioBean {
private String nombre;
private String password;

// ATRIBUTO: nombre
public String getNombre() { return nombre; }
public void setNombre(String nuevoValor) { nombre = nuevoValor; }

// ATRIBUTO: password
public String getPassword() { return password; }
public void setPassword(String nuevoValor) { password = nuevoValor; }
}
```

Aquest model se li comunica al sistema *JSF* mitjançant el fitxer *faces-config.xml*, on es detalla la part de *managed-bean*, també s'aprecia un *bean* denominat *usuari*, que està recollit en la classe *UsuarioBean*, i amb un àmbit de sessió:

```
<managed-bean>
<managed-bean-name>usuario</managed-bean-name>
<managed-bean-class>UsuarioBean</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Capa Vista

La vista és l'objecte que maneja la presentació visual de les dades gestionades pel Model. Genera una representació visual del model i mostra les dades a l'usuari. Interacciona amb el model a través d'una referència al propi model. *JSF* connecta la vista i el model, un component de la vista pot lligar-se a un atribut d'un *bean* del model, com per exemple :

<h:inputText value="#{usuario.nombre}"/>

Es pot observar com es declara un camp de text d'entrada (*inputText*) a la vista, aquest camp de text, recull el seu valor d'entrada a l'atribut *nom* d'un *bean* denominat *usuari*. D'aquesta manera s'estableix el vincle d'enllaç entre la vista i el model.

Capa Controlador

El controlador, és l'objecte que proporciona significat a les ordres de l'usuari, actuant sobre les dades representades pel model. Entra en acció quan es realitza alguna operació, ja sigui un canvi en la informació del model o una interacció sobre la vista. Es comunica amb el model i la vista a través d'una referència al propi model.

A més, *JSF* opera com un gestor que reacciona davant els esdeveniments provocats pel usuari, processa les seves accions i els valors d'aquests esdeveniments, i executa codi per actualitzar el model o la vista.

Seguint amb l'exemple anterior, una part del controlador recull les línies de codi del fitxer *index.jsp,* que captura les dades del nom d'usuari, contrasenya, i el botó de *Acceptar*:

```
<h:inputText value="#{usuario.nombre}"/>
<h:inputSecret value="#{usuario.password}"/>
<h:commandButton value="Aceptar" action="login"/>
```

i la part del dichero *hola.jsp*, que mostra el nom per pantalla:

```
<h:outputText value="#{usuario.nombre}"/>
```

D'altra banda, hi ha el control per a les regles de navegació, contingut en el fitxer *faces-config.xml*, on per exemple, s'indica que, estant *index.jsp*, si ocorre una acció anomenada *login*, es navegarà a la pàgina *hola.jsp*; aquesta acció comentada, és un *string* que es declara a la vista com un atribut del botó d'acceptar que apareix al formulari de l'exemple bàsic. El fitxer *faces-config* seria el següent:

```
<navigation-rule>
        <from-view-id>/index.jsp</from-view-id>
        <navigation-case>
        <from-outcome>login</from-outcome>
        <to-view-id>/hola.jsp</to-view-id>
        </navigation-case>
</navigation-rule>
```

Finalment, s'acaba de definir el controlador de l'aplicació amb el *servlet faces*, definit en el fitxer *web.xml*:

```
<web-app>
        <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        </welcome-file-list>
</web-app>
```

Amb la directiva *<servlet>,* s'estableix l'únic *servlet* de l'aplicació, que és el propi del *framework JSF*. La directiva *<servlet-mapping>,* correspon al camí *(url)* per accedir al *servlet* definit anteriorment. Qualsevol pàgina JSP que es vulgui visualitzar, si conté l'extensió *faces*, es visualitzarà a través de *JSF*.

Finalment amb *<welcome-file-list>* s'estableix com a pàgina d'arrancada de la aplicació *index.html*. Posteriorment, aquesta pàgina es redirecciona a *index.faces*, això es fa així perquè el *framework JSF*, a través del seu *servlet* principal prengui el control. Si no es fes d'aquesta manera, el servidor d'aplicacions mostraria una simple pàgina *JSP* com a tal, i la compilació d'aquesta pàgina, fora del marc *JSF*, provocaria errors.

Cicle de vida d'una pàgina Java Server Faces

El cicle de vida d'una pàgina *JSF* és similar al d'una pàgina *JSP*: El client fa una petició *HTTP* de la pàgina i el servidor respon amb la pàgina traduïda a *HTML*. No obstant això, a causa de les característiques extres que ofereix la tecnologia *JavaServer Faces*, el cicle de vida proporciona alguns serveis addicionals mitjançant l'execució d'alguns passos extres.

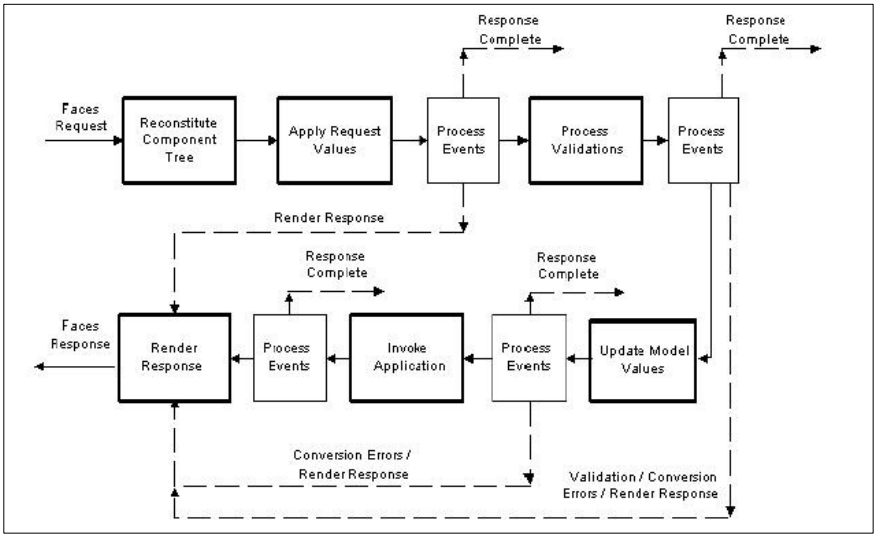


Figura 23: Cicle de vida petició-resposta a JSF

Els passos del cicle de vida s'executen depenent de si la petició es va originar o no des d'una aplicació *JavaServer Faces* i si la resposta és o no generada amb la fase de representació dels cicle de vida de *JavaServer Faces*. A la figura següent es mostra el cicle de vida d'una petició JSF :

2.5.5 Framework Spring

Introducció

Spring és un *framework* lleuger, per construir aplicacions empresarials. Es troba dividit en diferents mòduls, cadascun dels quals s'encarrega de parts diferents de l'aplicació. Aquesta separació en mòduls permet utilitzar només les parts que es necessiten, sense tenir la càrrega dels que no es fan servir.

Spring està dissenyat per no ser intrusiu, això vol dir que no cal que l'aplicació s'estengui o implementi alguna classe o interfície de *Spring*, així el codi de lògica quedarà lliure i completament reutilitzable per a un projecte sense *Spring*. Gràcies a això, és possible usar un *POJO* o un objecte *Java* per fer coses que abans només podien fer-se amb *EJBs*. No obstant això, la utilitat de *Spring* no és només per al desenvolupament d'aplicacions web, o no només al servidor. Qualsevol aplicació Java es pot beneficiar de l'ús de *Spring*.

A més, si es fa servir *Spring* de la forma correcta (la qual cosa no és difícil), l'aplicació quedarà dividida en capes ben delimitades, i amb bones pràctiques de programació.

IOC (Inversion Of Control)

El nucli de *Spring* està basat en un principi o patró de disseny anomenat Inversió de Control (*IOC* per les seves sigles en anglès). Les aplicacions que fan servir el principi de *IOC* es basen en la seva configuració (que en aquest cas pot ser en arxius *XML* o amb anotacions com a *Hibernate*), per descriure les dependències entre els seus components, és a dir, els altres objectes amb els quals interactua. En aquest cas "inversió" significa que l'aplicació no controla la seva estructura; permet que sigui el *framework* de *IOC* (en aquest cas *Spring*) qui ho faci.

Per a crear qualsevol objecte que s'executarà al contenidor IOC, s'haurà de crear una interfície amb el conjunt de les operacions que aquest proporciona. Aquesta interfície podrà ser implementada per moltes classes que ofereixen la mateixa funcionalitat. Mitjançant la inversió de control s'especifica quina d'aquestes implementacions utilitzar, fent servir un fitxer de configuració *XML*.

La principal utilitat, és poder separar la implementació de l'especificació de cada component. Això permetrà modificar quina classe utilitzar, sense haver de modificar el codi de l'aplicació.

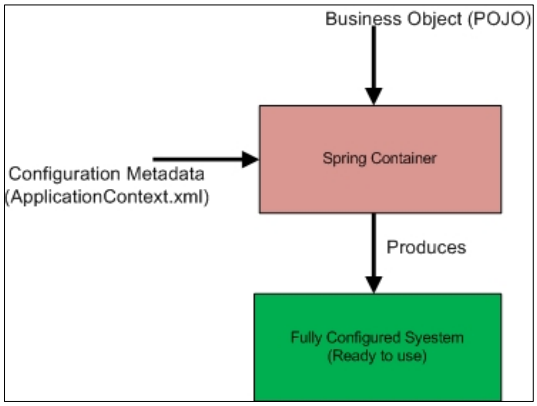


Figura 24: Contenidor IOC a String

La versió 3 de *Spring,* és una versió revisada i millorada de la versió estable anterior (2.5), que inclou noves característiques, entre les quals s'inclouen:

- **Suport per a Java 5**: Proporciona configuració basada en anotacions, a més la part *web* és compatible amb les versions 1.4 i 5 de *Java EE*. A causa d'aquesta nova característica, ara cal tenir el *JRE* versió 5 o superior.
- **Llenguatge d'Expressions (Spel):** En aquesta nova versió s'inclou un llenguatge d'expressions que pot ser usada quan es defineixen *beans*, tant en *XML* com amb anotacions i també dóna suport a través de tots els mòduls de *Spring*.
- **Suport per Serveis Web REST**: Ara *Spring* suport serveis *web* de tipus REST.
- **Suport per a Java EE6**: Ofereix suport de característiques com *JPA 2.0, JSF 2.0 i JRS 303* (validacions de *Beans*).
- **Suport per a bases de dades embegudes**: Un suport convenient per a bases de dades embegudes com *HSQL, H2 i Derby*.
- **Suport per a formateig de dades mitjançant anotacions**: Ara els camps de data, divises, etc., seran formatejats automàticament i convertits usant anotacions.
- **Nova organització dels mòduls**: Els mòduls han estat revisats i separats en diferents paquets, més organitzats, d'acord a la seva funcionalitat.

Spring MVC

Spring és un *framework* que abasta moltes funcionalitats utilitzades per a la construcció d'aplicacions. Cadascuna d'aquestes són oferides per *Spring* en forma de mòdul, que es pot utilitzar de forma independent.

Un d'aquests mòduls és *Spring MVC*, el qual ofereix una solució per al tractament de les peticions de l'usuari i la presentació d'informació provinent del model de l'aplicació.

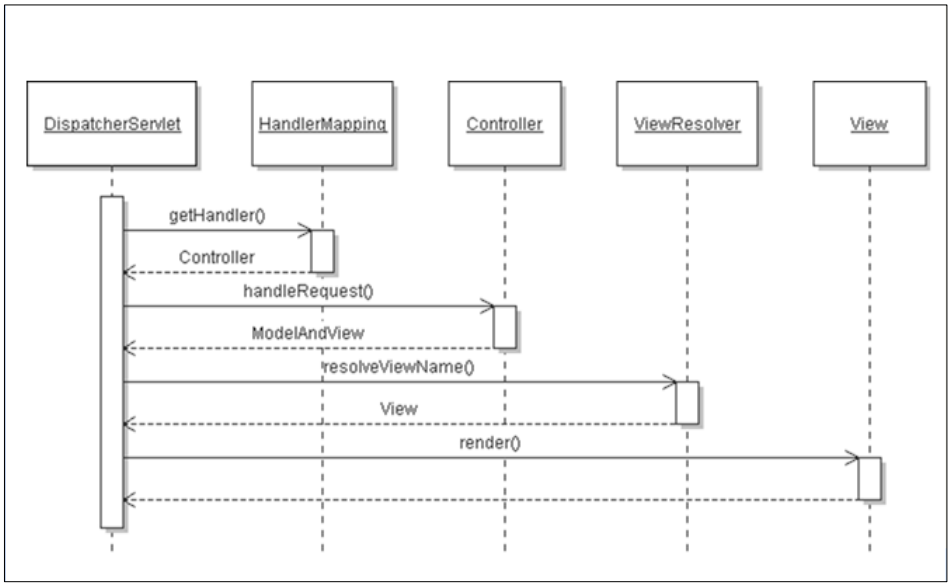


Figura 25: Diagrama de seqüència a Spring MVC

Els elements principals del mòdul són:

- **Dispatcher Servlet**: L'element principal d'aquest mòdul és el *Dispatcher Servlet,* que és un component tipus *Servlet* que acompleix l'especificació *J2EE* i que serà l'encarregat de rebre les peticions *web* que ha generat el client de l'aplicació. Està definit al fitxer de configuració **web.xml**.

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

<servlet-mapping>
  <servlet-name>SpringServlet</servlet-name>
  <url-pattern>*.spr</url-pattern>
</servlet-mapping>
```

  Tal i com es pot observar, es crea un *Servlet* anomenat *SpringServlet,* que està implementat per la classe *DispatcherServlet* i que capturarà totes les peticions que acabin amb *.spr*. El nom del *DispatcherServlet,* és el que utilitzarà *Spring* per a cercar el fitxer de configuració del mòdul..

- **Controller** : Un controlador és un element encarregat del tractament d'una determinada petició. Cadascun d'aquests elements han de cumplir l'especificació de la interfície *Controller*.

  Aquesta interfície, descriu el comportament bàsic que ha de tenir cada element d'aquest tipus. Per a utilitzar comportaments més específics i avançats, s'hauràn d'utilitzar interfícies que heretin d'aquesta com: *AbstractController*, *AbstractCommandController* i *SimpleFormController*.

Aquestes últimes defineixen comportaments més específics que suporta el *framework* i que s'han d'utilitzar de manera més concreta per una situació més particular. Un exemple de definició de controlador associat a una *url* seria:

```
<bean name="/prova.html" class="es.lycka.prova.control.provaController"/>
```

O sigui, es defineix un controlador amb nom *prova.html,* que és implementat per la classe *provaController* i que és una implementació de la interfície *Controller*.

- **View Resolvers**: Són elements utilitzats per transformar el nom d'una vista en una URL física on aquesta s'ubica. Per exemple :

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

En l'exemple anterior, tenim una element *viewResolver* que està implementat per la clase *UrlBasedViewResolver* i que cercarà totes les vistes al directori */WEB-INF/jsp* i que acabin amb *.jsp*.

Avantatges principals de Spring

Els principals punts forts de Spring i que el converteixen en una molt bona alternativa a *Struts* són :

- **Exhaustiu i modular** : Spring abarca una gran varietat de funcionalitats i aporta solucions a diferents aspectes d'una aplicació *web*. Al tenir una arquitectura modular, fa que es pugui treballar de forma aïllada cada part de l'aplicació.

- **Extensible** : Es pot ampliar el seu comportament de forma fàcil i fins i tot ampliar les funcionalitats oferides pel marc de treball.

- **Contenidor més lleuger** : Malgrat la potencia de IOC, el contenidor és força lleuger i ràpid.

- **Facilitat d'ús** : Tot i tenir una corba d'aprenentatge inicial força elevada, el seu ús facilita la creació d'aplicacions *web* i permet que un cop aprés, es puguin automatitzar moltes tasques.

- **Facilitat d'Integració amb altres tecnologies** : Permet integració amb tecnologies com *JSP/JSTL, Tiles, Velocity, Struts, FreeMarker*,etc.

2.5.6 Resum dels Framework estudiats

Una vegada analitzats els principals *framework* del mercat, es resumeixen les seves principals característiques :

Struts

- Excel·lent sistema de missatges
- Tecnologia molt madura
- Bones eines de configuració (*Struts GUI, Struts Console*)
- Entorn *multi-thread.*
- Comunitat molt grossa de desenvolupadors

Struts 2

- *Framework* basat en *Actions*.
- Té un gran grup de desenvolupadors i una gran comunitat d'usuaris.
- Anotacions i opcions de configuració *XML*.
- Objectes POJO, basats en accions que són fàcils d'usar.
- Integració amb *Spring, SiteMesh* i *Tiles*.
- Integració amb llenguatge d'expressió *OGNL* (**O**bject-**G**raph **N**avigation **L**anguage).
- Temes basats en els *tags libraries* i *Ajax*.
- Múltiples opcions de vista (*JSF, Freemarker, Velocity* i *XSLT*).
- *Plug-ins* per ampliar i modificar el *framework*.

Java Server Faces

- Basat en components (facilita la reutilització)
- Eines visuals de desenvolupament
- Potents eines de desenvolupament : *Faces Console, JSF formbuilder*

Spring

- Injecció de Dependències (DI): Configuració automàtica de dependències d'una classe.
- Programació Orientada a Aspectes: Addició de funcionalitats comuns al llarg d'un sistema (ex. Sistema de *Logging*).
- Plantilles : Permeten reutilitzar codi i reduir l'escriptura de funcions repetitives.

Capítol 3. Anàlisi i Disseny Framework TOMIR

El *framework* propi dissenyat en el present projecte al que s'ha anomenat TOMIR[1], és un nou marc de treball que facilita la creació d'aplicacions *web*, resolent part de la problemàtica associada a la capa de presentació. Per tant, s'han implementat les funcionalitats més importants, amb una arquitectura que serveixi com a prototip i que permeti incorporar noves millores de forma senzilla per al desenvolupador.

De tots els marcs de treball estudiats, s'ha fet servir *Struts* com a model de referència, ja que és el més senzill i el que aporta una documentació més completa. A continuació, en els següents apartats s'analitza el *framework* TOMIR.

3.1 Característiques generals

Els *frameworks* poden ser vistos com implementacions de patrons de disseny que faciliten la reutilització de disseny i codi. El *framework* TOMIR, implementa les següents funcionalitats que són inherents a qualsevol *framework J2EE* :

- **Validació de camps** : Implementa a la *Interface Action* el mètode **validate**, on els desenvolupadors podran introduir el codi de validació corresponent a la seva aplicació, per exemple, comprovar si un registre ja està donat d'alta.

- **Control d'errors (Excepcions)** : S'ha creat una classe de tipus *Exception* (*TomirException*) que permet crear excepcions pròpies amb missatges personalitzats.

- **Internacionalització (missatges Locale)** : S'ha creat la classe *ResourceBundleConfig* que gestiona la internacionalització (**i18n**). Aquesta classe parsea els valors de *i18n* (idioma, país i ruta dels fitxers de *properties*) indicats al fitxer de configuració del *framework* (*tomir-config.xml*).

- **Mecanisme de creació de TagLibs (llibreries d'etiquetes) :** S'han creat classes Java que hereten de la classe *Tagsupport* i que implementen els missatges que s'hauran de mostrar a les pàgines *jsp*, fent usar el sistema de internacionalització (*i18n*). Així el desenvolupador, de manera transparent i còmoda podrà implementar missatges a les pàgines *jsp*.

- **Sistema de configuració** : A partir del fitxer de configuració del *framework* (*tomir-config.xml*), es pot configurar l'execució del *framework* de forma declarativa, definint les accions, formularis, camps i internacionalització.

- **Components d'interfície d'usuari (UI Components**), S'han implementat algunes extensions enriquides d'elements *HTML*: *inputs i css*.

[1] TOMIR identifica a una de les muntanyes més emblemàtiques de la Serra de Tramuntana a la illa de Mallorca

L'enfocament que s'utilitzarà per al desenvolupament del *framework* TOMIR és el basat en accions, concretament el *MVC* Model *2*. Els avantatges d'utilitzar aquest tipus de *frameworks* són:

- Utilitzant el *framework*, l'aplicació adopta la separació de la vista i el model.
- Proporciona un control centralitzat de les accions.
- Facilita el desenvolupament i el manteniment de l'aplicació
- Facilita als desenvolupadors treballar de forma separada les diferents parts
- Permet la reutilització de components.

3.2 Requeriments

Per al desenvolupament del *framework TOMIR,* s'ha utilitzat el llenguatge de programació *Java*, fent ús de components compatibles amb les especificacions *J2EE*. Cal una màquina virtual de *Java* versió 1.6 o superior.

El producte generat, serà una llibreria *JAR,* de nom *frameworkTOMIR.jar,* que s'ha d'incloure dins de cada projecte web que faci ús del *framework* i executat en un contenidor d'aplicacions compatible amb les especificacions *J2EE*. El desenvolupament s'ha fet sobre un servidor *Apache Tomcat.*

3.3 Dependències

El present projecte, necessita altres llibreries externes per a poder funcionar. A continuació, es mostra un diagrama de dependències, que especifica quines són les que necessita el *Framework Tomir*.

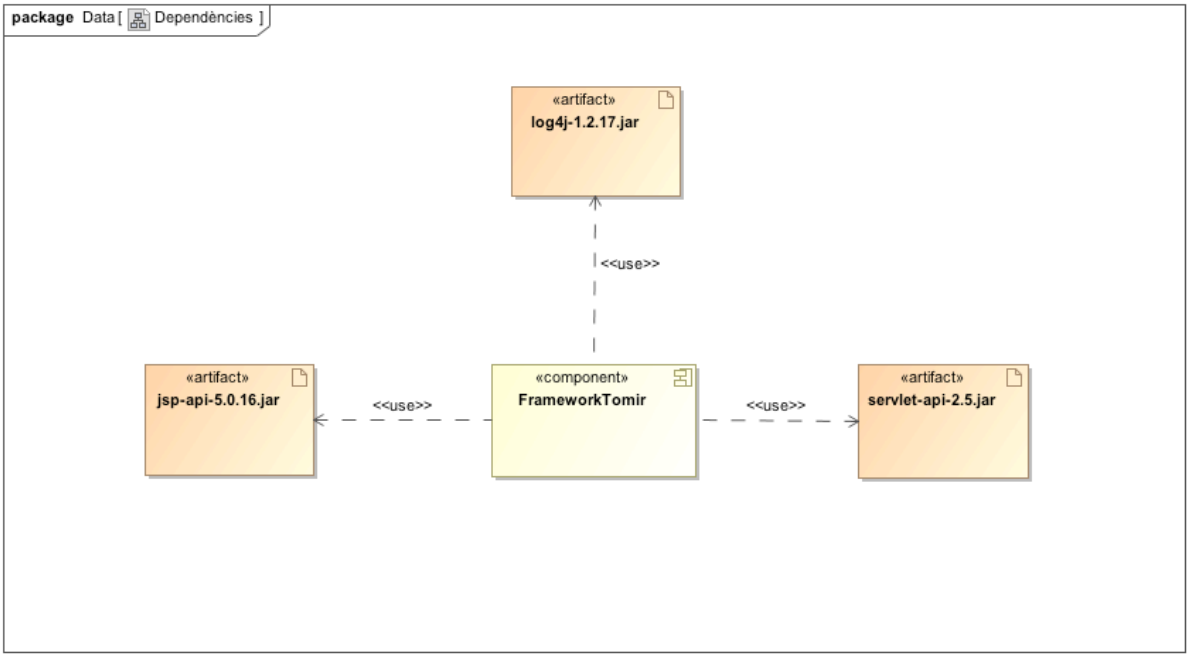


Figura 26: Diagrama de dependències Framework Tomir

3.4 Estructura i jerarquia de paquets

El projecte s'estructura en diversos paquets *java(packages*), evitant la duplicitat de classes, i agrupant-les per la seva funcionalitat. En la següent taula, es pot veure la llista dels paquets i la seva descripció:

Tabla 1: Llista de paquets del framework

Nom del paquet	Descripció
fw.action	Classes Action (Interface de les accions) i ActionForm(Formularis)
fw.config	Classes de Configuració (General del Framework, Formularis i Camps)
fw.controller	Classes de recepció i control de les peticions (FrontController i Application Controller).
fw.exceptions	Classe per el control de les excepcions (TomirException)
fw.i18n	Classe per la gestió de la internacionalització
fw.tags	Classes on es defineixen les etiquetes personalitzables

A continuació, s'adjunta el diagrama de paquets del *framework* Tomir :

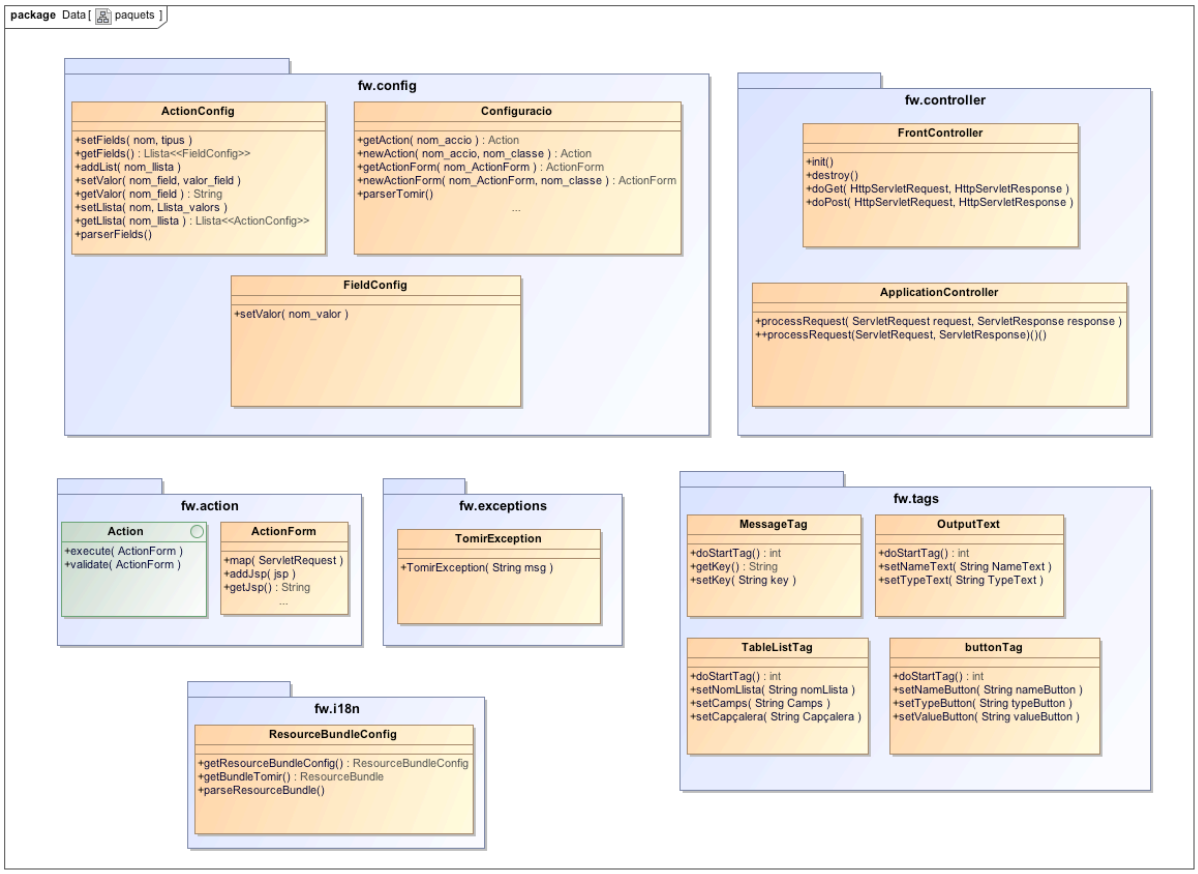


Figura 27: Diagrama de paquets

3.5 Configuració del Framework

El els següents punts, s'explica com està definit el sistema de configuració del *framework* i el disseny que s'ha seguit per a implementar-ho.

3.5.1 Fitxer de configuració del Framework

Tal i com s'ha explicat en el punt 6.1, una de les característiques del f*ramework*, és el seu sistema de configuració declaratiu, especificat en un únic fitxer *XML* (**tomir-config.xml**). Aquesta centralització, té l'avantatge a l'hora de fer canvis, ja que no s'ha de tornar compilar i empaquetar, només caldrà modificat el fitxer *XML*.

Una vegada que arranca el *Servlet* controllador, es llegeix el fitxer *XML*, utilitzant el *parser DOM*. Aquest, llegeix les etiquetes del fitxer *XML* i, a continuació, utilitzant tècniques de reflexió, es creen els objectes *java* en temps d'execució.

Per tant, els desenvolupadors que utilitzin aquest *Framework*, hauran d'incloure aquest fitxer de configuració amb la següent estructura :

```
<?xml version="1.0" encoding="UTF-8"?>

<configuracio>
   <accio>
      <nom></nom>
      <classe></classe>
   </accio>

   <formulari>
      <nom></nom>
      <action></action>
      <classe></classe>
   </formulari>

   <camp>
      <nom></nom>
      <tipus></tipus>
   </camp>

   <i18n>
      <idioma></idioma>
      <pais></pais>
      <path></path>
   </i18n>
</configuracio>
```

Com es pot veure en l'estructura *XML* anterior, els objectes que els desenvolupadors podran definir seran:

- **Accions** : Nom de l'acció (*Action*) i classe (paquet on estarà definida l'acció)
- **Formularis** : Nom del formulari (*ActionForm*), nom de l'acció associada, i classe (paquet on estarà definit el formulari)
- **Camps** : Nom i tipus dels camps del model de dades de l'aplicació.
- **i18n** : Valors que defineixen la internacionalització del *Framework* (idioma, país i ruta dels fitxers *.properties*).

3.5.2 Parser DOM

Un *parser* o analitzador sintàctic llegeix el document *XML* i verifica que és XML ben format, alguns també comproven que el codi *XML* sigui vàlid.

Mitjançant el *parser,* no només es pot comprovar si els documents són ben formats o vàlids, sinó que també, es poden incorporar a les aplicacions, de manera que aquestes puguin manipular i treballar amb documents *XML*.

El *Model d'Objectes del Document* o *Document Object Model (DOM)* és un model d'objectes estandarditzat per a documents *HTML* i *XML*. *DOM* és un conjunt d'interfícies per descriure una estructura abstracta per un document XML.

L'analitzador DOM, construeix una estructura d'arbre que representa un document *XML*. Llavors, l'aplicació pot manipular els nodes d'aquest arbre. *L'API DOM* defineix el mecanisme per a consultes, travessar i manipular el model d'objectes de construcció.

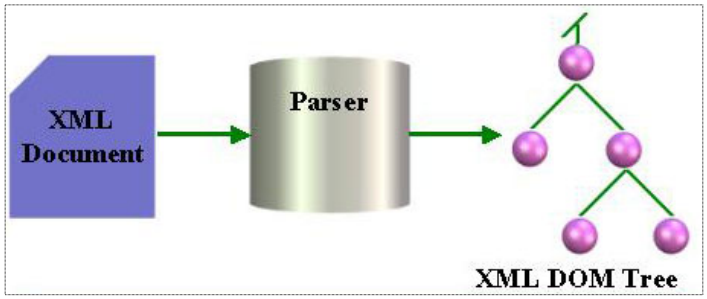


Figura 28: Estructura parser DOM

3.5.3 Tècnica de Reflexió en Java

La reflexió en *java*, és la capacitat que té un programa per conèixer la seva estructura d'alt nivell. Es pot saber a partir d'una classe ja compilada tots els seus mètodes, propietats, instanciar-la, executar els seus mètodes, ...

Quan el codi font d'un programa es compila, perd la informació sobre l'estructura del programa conforme es genera el codi de baix nivell. La reflexió ens permet, en temps d'execució, tornar a conèixer l'estructura d'alt nivell.

Mitjançant el mètode estàtic *forName (String)* de la classe *Class,* es pot crear el motlle d'una classe a través del seu nom.

```
Class MyClass = Class.forName ("className");
```

I, mitjançant el mètode *newInstance()* es crea un objecte d'aquesta classe. A partir d'aquest moment, ja es pot accedir a les propietats i mètodes d'aquesta classe.

```
myClass.newInstance ();
```

Una vegada llegit pel *parser DOM,* el fitxer *XML* del *Framework*, es creen en temps d'execució les classes corresponents a les etiquetes definides en el fitxer **tomir-config.xml**, per exemple per a crear una nova classe *Action*, el codi java seria el següent :

```
private Action newAction(String nom_accio, String nom_classe)
{
    String nomClass = nom_classe + "." + nom_accio;

    Class classe;
    Action action = null;
    log.info("newAction-->nom_accio= "+nom_accio);

    try {
        classe = Class.forName(nomClass);
        action = (Action) classe.newInstance();
    }
    catch (Exception e)
    {
        log.error("newAction-->Error in Action " + nom_accio);
        throw new TomirException("Error creant l'acció "+e.getMessage());
    }
    return action;
}
```

3.6 Disseny del Framework

L'estudi realitzat en els apartats anteriors dels principals *frameworks* del mercat, és de gran ajuda, per entendre l'arquitectura d'un *framework J2EE* i ha servit com a base per a dissenyar el *framework* TOMIR. En els pròxims apartats es mostren les funcionalitats en el disseny de TOMIR.

3.6.1 Patró Singleton

El *Patró Singleton,* és un patró de tipus creacional, que s'utilitza, per no permetre que hi hagi múltiples instàncies d'una classe, concretament només una. En el present projecte aquest patró s'utilitza en la classe **Configuracio** que només s'ha de instanciar una vegada al inici del procés de la petició.

Una vegada instanciada, el patró *Singleton* controla que no se puguin crear més instancies des de l'exterior. L'estructura d'aquest patró és la següent :

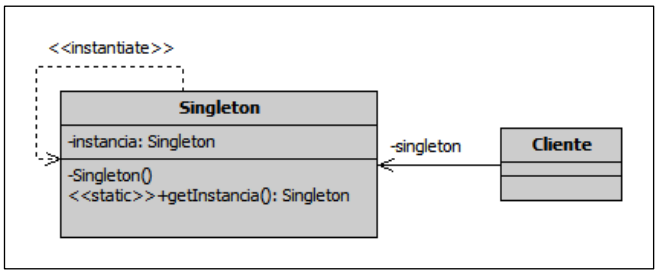


Figura 29: Estructura Patró Singleton

El codi font de la classe **Configuracio** on s'aplica aquest patró, és el següent :

```
public static Configuracio getConfiguracio()
{
    if (configuracio == null) configuracio = new Configuracio();
    return configuracio;
}
```

3.6.2 Processament de les peticions (Patró Service To Worker)

El *framework* TOMIR, es basa en el patró *Service To Worker*, el qual utilitza altres patrons ja coneguts, el control centralitzat i procés de les peticions ho realitza utilitzant el patró *Front Controller*, que centralitza la lògica en un únic lloc, evitant així la distribució de les peticions entre les vistes o altres components.

El controlador, delega al patró *Application Controller* la funció del maneig de les accions a executar a partir de la *URL*, invocant l'acció seguint el patró *Command*. *L'Application Controller* determina quina és la següent vista a mostrar.

Per últim, mitjançant el patró *View Helper*, es separa la lògica de negoci de la composició de la vista; les classes específiques de *etiquetes* (*JSP Custom Tags*) fan aquesta funció.

A continuació, es mostra el diagrama de seqüència que segueix el *framework* TOMIR, basat en el patró *Service To Worker*, el qual representa la interacció dels elements i l'intercanvi de missatges que aquests realitzen.

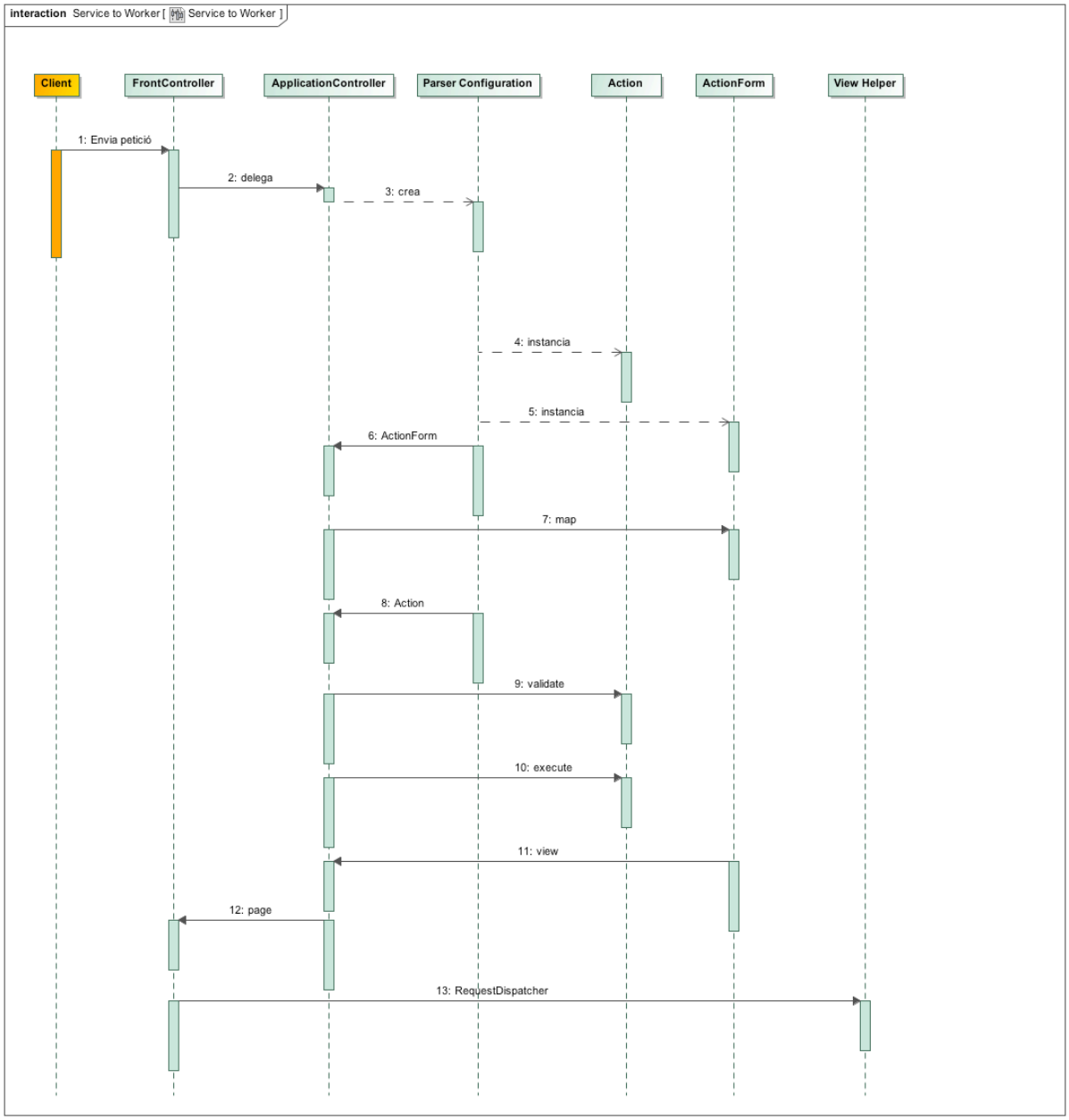


Figura 30: Diagrama de seqüència del framework

3.6.3 FrontController (Controlador)

La classe *FrontController,* fa la funció de controlador com el punt inicial de contacte per gestionar les peticions *http* dels clients; hereta de la classe *HttpServlet* del sistema i mitjançant la configuració del descriptor de desplegament (**web.xml**), captura totes les peticions que es fan segons un determinat format de *url*, el qual codifica el nom de la comanda a executar.

L'estratègia per implementar el controlador serà la de **Servlet Front Strategy**, que suggereix la implementació del controlador com un *servlet,* tal i com s'ha comentat en l'apartat de patrons, és realitzarà desenvolupant un *servlet* que s'encarregarà de centralitzar les peticions *HTTPRequest* sobre el *framework* i que serà desplegat en el servidor d'aplicacions.

El patró *Front Controller,* sol treballar juntament amb un component de tipus *Dispatcher,* que és el responsable del maneig de les vistes i la navegació. La seva responsabilitat és triar quina serà la següent vista i dirigir el control al recurs. En aquest projecte, el paper de *Dispatcher* el farà un altre component basat en el patró **Application Controller**. Les tasques que es realitzaran seran la resolució de la petició accedint a l'acció apropiada, la posterior invocació d'aquesta i finalment el lliurament del control a la vista apropiada.

3.6.4 ApplicationController (Processament de les peticions)

El *framework* TOMIR es basa en el patró *Front Controller*, en lloc d'encapsular el maneig de les accions i vistes en el mateix controlador, és millor moure aquest comportament a un conjunt separat de classes, integrades en el patró **Application Controller**. Aquest, és el responsable de centralitzar el control del flux de navegació, és a dir, de les accions i vistes. La seva missió és triar l'acció apropiada i la seva vista corresponent per satisfer la petició.

Aquest patró té les següents responsabilitats :

- Parsear els valors de configuració del Framework (accions, formularis, camps i internacionalització), de manera declarativa, a partir del fitxer **tomir-config.xml**.
- Cercar i crear l'acció que sol·licita l'usuari.
- Crear el formulari associat a cada acció.
- Omplir el formulari amb les dades de la petició
- Realitzar les validacions simples que té associades el formulari
- Executar l'acció associada a la petició.
- Controlar les excepcions que es produeixen a les accions creades pel desenvolupador *web*.
- Escollir quina vista és la següent a mostrar.

3.6.5 Classes de gestió dels formularis (ActionForm i ActionConfig)

Els formularis, són elements utilitzats per a fer arribar les dades de l'usuari al servidor, facilitant l'encapsulació de les dades i el seu tractament en el codi de les accions. Cada formulari, ha d'estar assignat a una acció i es configura de forma declarativa en el fitxer de configuració del *framework* (**tomir-config.xml**)

La classe *ActionConfig,* defineix els objectes *hashmap* on es guardaran els valors dels camps. Implementa mètodes *set/get* i un mètode per llegir els camps que s'hauran definit de manera declarativa el fitxer de configuració del framework (**tomir-config.xml**).

La classe *ActionForm*, que hereta de *ActionConfig*, té la funció d'emmagatzemar la pròxima vista (**jsp**) a executar, i també de mapejar els valors dels camps del formulari.

3.6.6 Patró Command (Classe Action).

Aquest objecte, és l'encarregat de les accions que s'executen en el servidor. Segueix el patró de comportament *Command*, el qual disposarà de dos mètodes (**execute()** i **validate()**). El mètode *execute*, en funció de les dades rebudes en el formulari, executarà la lògica de negoci i retornarà el codi de resultat de la vista a generar. El mètode *validate*, té com a funció validar les dades del formulari, aquest procés es fa abans de l'execució, el desenvolupador podrà introduir en aquest mètode les seves validacions.

Les accions, són reconegudes pel *framework* per un identificador que forma part de la *URL* de la petició. L'aplicació *web* està parametritzada en el fitxer descriptor **web.xml,** perquè totes les URL que es dirigeixen a una acció siguin capturades pel *servlet* controlador (**FrontController**).

Un cop pres el control pel **FrontController**, s'analitzarà la *URL* de petició, per obtenir l'identificador de l'acció . Amb aquesta informació es localitzarà en la configuració quina és la classe corresponent a la comanda associada a aquest identificador, i serà la classe **Action** que actuarà com executora de la lògica de negoci.

La classe corresponent a l'acció, serà instanciada dinàmicament en temps d'execució i invocada per l'**ApplicationController**, totes les accions han implementar la interfície *Action*, la qual reb com a paràmetres el propi context de *servlet* juntament amb un objecte **ActionForm** que representa . Prèviament a la execució del *Action* es realitza una validació mitjançant la invocació al mètode *validate* per part del **ApplicationController** .

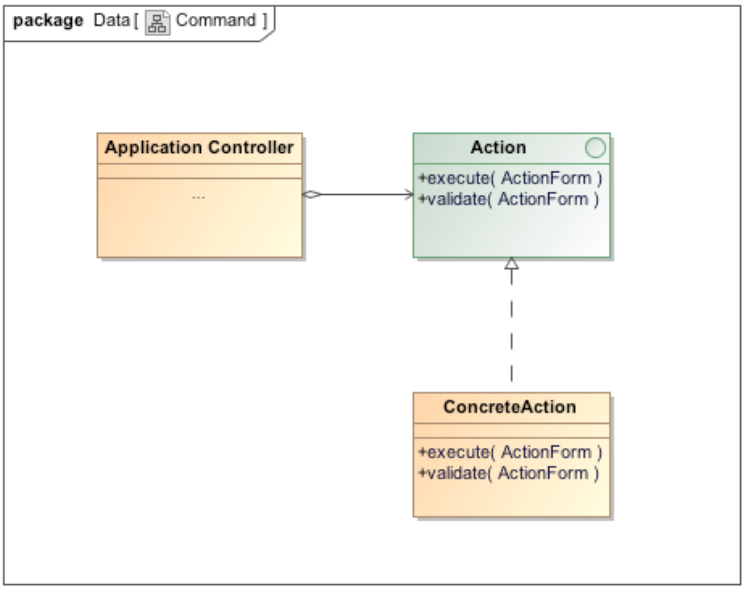


Figura 31: Diagrama de classes patró Command

3.6.7. Internacionalització (i18n)

El servei d'internacionalització, facilita la traducció d'aplicacions *web* en múltiples idiomes, per aconseguir això d'una manera eficient i ben feta, el primer que s'ha d'evitar és escriure les traduccions directament a les vistes. El primer pas per configurar aquesta característica de la web, és escriure els textos en fitxers de propietats (**.properties**) agrupats per idioma.

El nom del fitxer, ha d'acabar amb el *Locale* de l'idioma que representa. Així doncs, el primer dels fitxers de l'exemple representa els textos en català i el segon en castellà.

Un exemple podria esser com el següent :

- **MessagesBundle_ca_ES.properties**

  client.nom=Nom
  client.adreça=Adreça
  client.poblacio=Poblacio
  client.telefon=Telefon

- **MessagesBundle_es_ES.properties**

  client.nom=Nombre
  client.adreça=Direccion
  client.poblacio=Poblacion
  client.telefon=Telefono

Al fitxer de configuració del framework (**tomir-config.xml**), s'ha de introduir l'etiqueta corresponent per a configurar el sistema de internacionalització i el paquet on estan les propietats (fitxers **.properties**)

Classe ResourceBundleConfig

Quan es carrega la configuració del *framework*, es crea un objecte de la classe **ResourceBundleConfig** i aquest, és afegit a la configuració general del *framework*. Aquesta classe, parseará els valors de idioma, país i ruta on estaran els fitxers de propietats (.**properties**).

A continuació, definirà el *Locale* indicat en la configuració, i configurarà les dades de internacionalització a l'objecte. Per últim, per a mostrar els missatges en les vistes (*jsp*), només cal instanciar aquesta classe i agafar els valors corresponents del fitxer de *properties*.

3.6.8 Tractament d'excepcions.

Durant l'execució del codi d'una acció, es poden produir excepcions que han de ser capturades i tractades per a que es mostri informació aclaridora a l'usuari.

En el *framework*, s'ha definit una classe específica per tractar les excepcions, anomenada **TomirException**. Implementa un mètode, que reb com a paràmetre el missatge de error que es mostrarà quan es llanci una excepció.

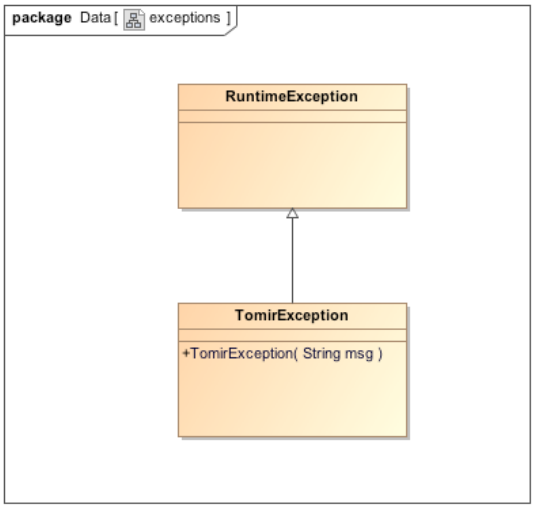


Figura 32: Diagrama de classes TomirException

3.6.9 Patró View Helper i llibreries d'etiquetes (JSP Custom Tags)

Aquest patró, separa la lògica de negoci del codi de formateig, i delega la responsabilitat de processament a les classes d'etiquetes personalitzades (*JSP Custom Tags*). Per a implementar el component Vista (*View*), s'utilitza l'estratègia *JSP View*, que utilitza una pàgina *JSP* com a component de Vista.

Les *JSP Custom Tags* (Llibreries d'etiquetes), són una eina molt útil que permeten separar la lògica de negoci de la presentació visual, el procés per implementar-les al *framework* és el següent :

- Crear la classe que implementi la funcionalitat de l'etiqueta
- Referenciar la classe al fitxer de configuració .tld
- Referenciar el fitxer .tld a la pàgina .jsp.

El Framework TOMIR implementa una llibreria d'etiquetes que conté els següents *JSP Custom Tags* :

- MessageTag: Mostra un missatge del fitxer de properties (i18n).
- OutputText: Mostra un element de tipus camp d'entrada al formulari.
- TableListTag: Mostra una llista de valors en format de taula amb capçalera.
- buttonTag: Mostra un element de tipus botó al formulari.

A continuació es mostra el diagrama de classes de les etiquetes :

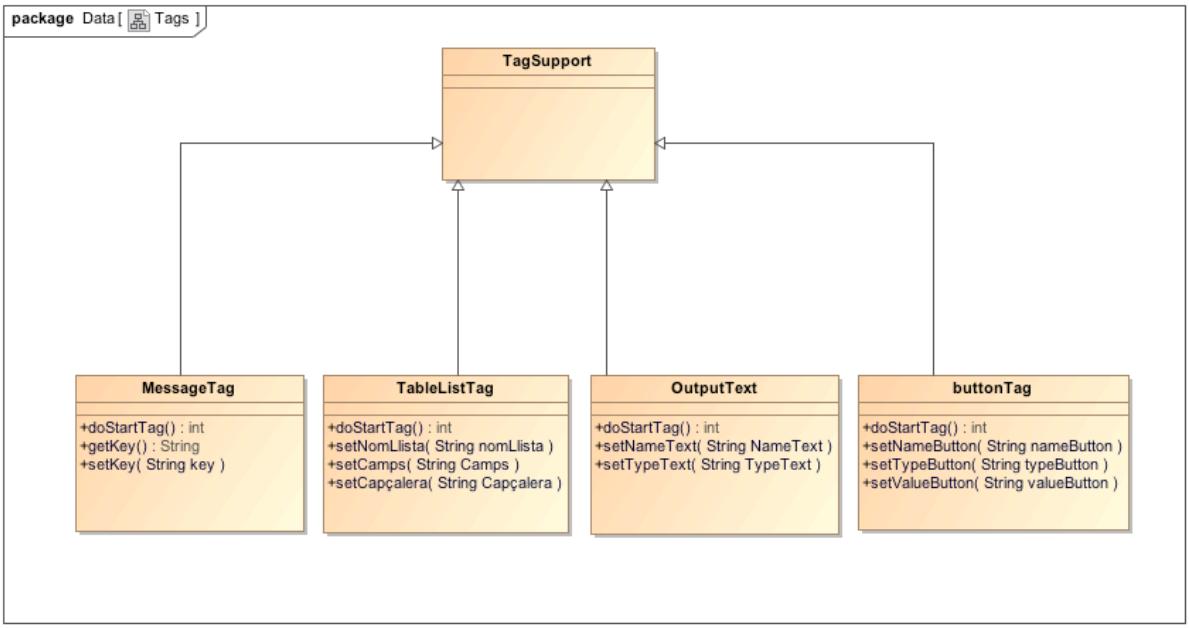


Figura 33: Diagrama de classes JSP Custom Tags

3.6.10 Diagrama de classes Framework Tomir

Una vegada descrits tots els aspectes i funcionalitats del *framework*, es mostra el seu diagrama de classes general:

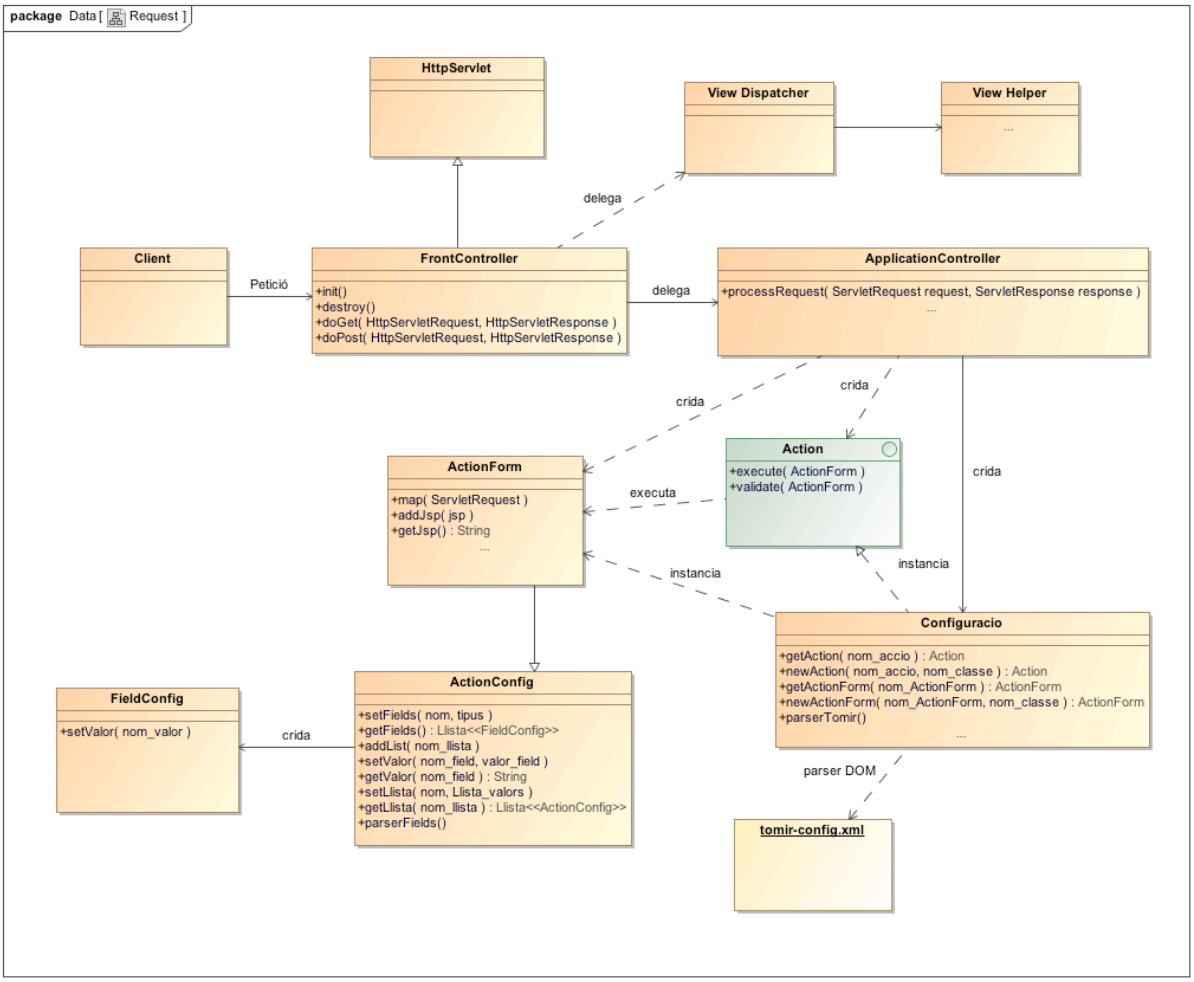


Figura 34:Diagrama de Classes Framework Tomir

Capítol 4. Aplicació test del Fwk Tomir

S'ha desenvolupat una petita aplicació web de prova, per testejar les funcionalitats del *Framework Tomir*. En els pròxims apartats, es podrà veure la descripció general de l'aplicació web, així com els diagrama de casos d'ús, descripció de formularis, accions, model de dades, i18n i en definitiva tota la funcionalitat de l'aplicació.

4.1 Descripció de l'aplicació web

L'aplicació treballa amb un model de dades de Clients, sobre el que es pot treure per pantalla un llistat i també donar d'alta nous Clients.

4.2 Requeriments

Aquesta aplicació web, ha estat desenvolupada utilitzant l'eina de desenvolupament *java*, *Netbeans 7.3*. Per tal de poder executar l'aplicació s'han d'acomplir els següents requisits:

- Disposar d'una màquina virtual Java (JRE) a partir de la versió 1.6.
- Servidor Web amb contenidor de Servlets o servidor d'aplicacions que implementi l'especificació Java. En el present projecte s'ha utilitzat Tomcat 7.0 com a contenidor J2EE.
- Navegador web, per a les proves s'ha utilizar *Google Chrome*.

4.3 Diagrama de casos d'ús

El diagrama de casos d'ús és molt simple, mostra les operacions que pot realitzar un usuari, treure un llistat de Clients i donar d'alta un nou Client.

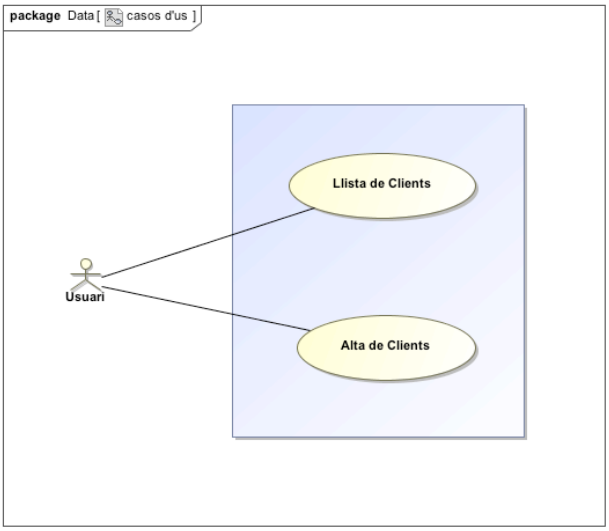


Figura 35: Diagrama de casos d'ús aplicació web de prova

4.4 Estructura i jerarquia de paquets

El projecte s'estructura en diversos paquets *java(packages*), evitant la duplicitat de classes, i agrupant-les per la seva funcionalitat. En la següent taula es pot veure la llista dels paquets i la seva descripció:

Tabla 2: Llista de paquets de l'aplicació web de prova

Nom del paquet	Descripció
web.actionForms	Formularis (ListClientForm i AltaClientForm)
web.actions	Classes Action (AltaCLient i ListClient)
web.controller	Classe Servlet controladora de l'aplicació.
web.dades	Magatzem de dades de prova de l'aplicació (Clients)
web.i18n	Fitxers de properties per a la gestió de la internacionalització
web.model	Model de dades (clients)

A continuació, s'adjunta el diagrama de paquets del *framework* Tomir

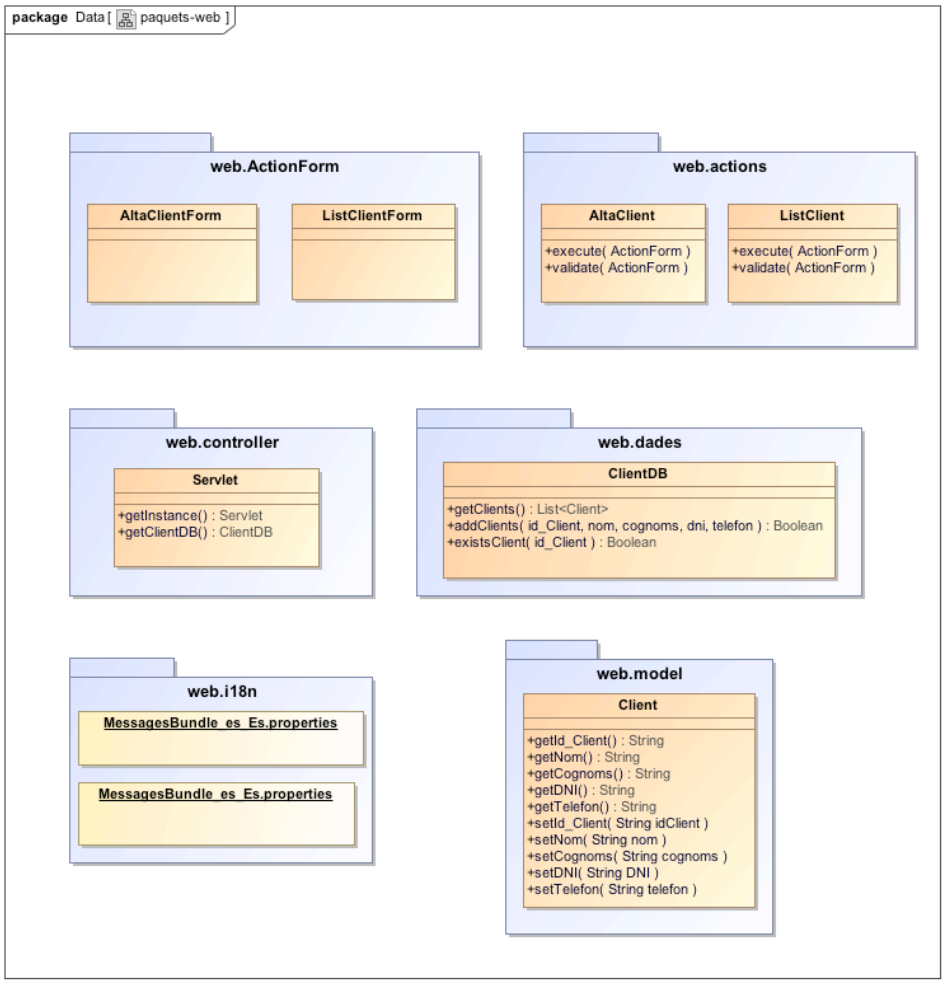


Figura 36: Diagrama de paquets aplicació web de prova

4.5 Dependències

Aquesta aplicació web de prova necessita altres llibreries externes per a poder funcionar, una d'elles és el fitxer *.jar* del *framework* (**FrameworkTomir.jar**). A continuació, es mostra un diagrama de dependències que especifica quines són les que necessita.

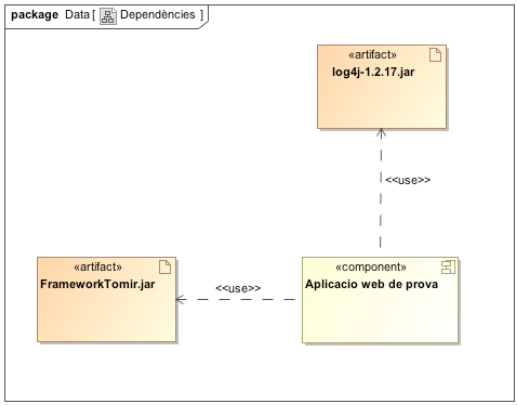


Figura 37: Diagrama dependències aplicació de prova

4.6 Fitxer de Configuració (tomir-config.xml)

A l'apartat 6.5, es detallava l'estructura del fitxer de configuració del *framework*, a continuació es mostra com està definit aquest fitxer amb les dades necessàries per l'aplicació web de prova :

```
<?xml version="1.0" encoding="UTF-8"?>
<configuracio>
  <accio>
    <nom>ListClient</nom>
    <classe>web.actions</classe>
  </accio>
  <accio>
    <nom>AltaClient</nom>
    <classe>web.actions</classe>
  </accio>
  <formulari>
    <nom>ListClientForm</nom>
    <action>ListClient</action>
    <classe>web.actionForms</classe>
  </formulari>
  <formulari>
    <nom>AltaClientForm</nom>
    <action>AltaClient</action>
    <classe>web.actionForms</classe>
  </formulari>
  <camp>
    <nom>id_Client</nom>
    <tipus>String</tipus>
  </camp>
  <camp>
    <nom>nom</nom>
    <tipus>String</tipus>
  </camp>
  <camp>
    <nom>cognoms</nom>
    <tipus>String</tipus>
  </camp>
```

```
  <camp>
    <nom>dni</nom>
    <tipus>String</tipus>
  </camp>
  <camp>
    <nom>telefon</nom>
    <tipus>String</tipus>
  </camp>
  <i18n>
    <idioma>es</idioma>
    <pais>ES</pais>
    <path>web.i18n.MessagesBundle</path>
  </i18n>
/configuracio>
```

Com es pot veure al fitxer de configuració anteriors, es defineixen els següents elements de configuració del *framework* TOMIR:

- Accions : **ListClient** (Llistat de Clients) i **AltaClient** (Alta de Clients), ubicats en el paquet **web.actions** del projecte.
- Formularis : **AltaClientForm** (Formulari de Alta de Clients), **ListClientForm** (Formulari de Llistat de Clients).
- Camps del formularis : **id_Client** (identificació del client), **nom** (nom de client), **cognoms** (cognoms del client), **dni** (DNI del client), **telèfon** (telèfon del client). Estan definits tots de tipus cadena (String).
- Internacionalització (i18n) : **es** (idioma castellà de Espanya), **ES** (país, Espanya), situats al paquet **web.i18n.MessagesBundle** del projecte.

4.7 Controlador (Servlet) i fitxer descriptor web.xml

Les aplicacions *web Java,* utilitzen un arxiu de descriptor d'implementació per determinar la forma en què les *URL* s'assignen als *servlets* i les adreces *URL* que necessiten autenticació, entre altra informació. Aquest fitxer es diu **web.xml** i es troba en el *WAR* de l'aplicació, concretament, en el directori *WEB-INF/.* El fitxer **web.xml** forma part de l'estàndard de *servlet* per les aplicacions *web*.

web.xml defineix les assignacions entre les rutes d'*URL* i els *servlets* que controlen les sol·licituds amb aquestes rutes. El servidor web, utilitza aquesta configuració per identificar el *servlet* que controla una sol·licitud concreta i, a continuació, invoca el mètode de classe que concorda amb el mètode de sol·licitud (per exemple, el mètode *doGet()* de les sol·licituds de tipus *GET HTTP*).

Per assignar una *URL* a un *servlet*, es declara el servlet amb l'element <*servlet*> i, a continuació, defineix una assignació des d'una ruta d'*URL* fins a una declaració de servlet amb l'element <*servlet-mapping*>. Es poden declarar diversos *servlets,* mitjançant la mateixa classe, utilitzant diferents paràmetres d'inicialització. El nom de cada *servlet* ha de ser únic en tot el descriptor d'implementació.

<*welcome-file-list*> indica al servidor que comprovi **/index.jsp** i **/index.html** abans d'informar que la *URL* no existeix:

A continuació es mostra el fitxer descriptor **web.xml** definit en l'aplicació web de prova :

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Framework TOMIR web</display-name>
  <servlet>
    <servlet-name>Servlet</servlet-name>
    <servlet-class>fw.controller.FrontController</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>

  <!-- Mappings dels servlets -->
  <servlet-mapping>
     <servlet-name>Servlet</servlet-name>
     <url-pattern>/Servlet/*</url-pattern>
  </servlet-mapping>

  <session-config>
     <session-timeout>30</session-timeout>
  </session-config>

 <!-- The Welcome File List -->
 <welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>

 <error-page>
    <exception-type>fw.exceptions.TomirException</exception-type>
    <location>/errorTomir.jsp</location>
 </error-page>
</web-app>
```

4.8 Internacionalització (Fitxers .properties)

A continuació, es mostren els dos fitxers .properties que s'utilitzen per definir els missatges en els dos idiomes configurats (català i castellà):

```
# Fitxer MessagesBundle_ca_ES.properties
#Traduccions al CATALÀ
#====================

#Títols de les pantalles
		title=Llistat de Clients
		title.alta.clients=Alta Client

#Botons
		button_alta_client=Alta Client
		button.tornar=tornar

#Errors
		error.alta.client=S'ha produit un error donant d'alta el client

#Columnes del llistat de Clients
		client.id=id Client
		client.nom=Nom
		client.cognoms=Cognoms
		client.dni=DNI
		client.telefon=Telèfon

#Validació de formulari
		client.altaOK=Client donat d'alta correctament
		client.tornar=Tornar a la pàgina principal
```

```
# Fitxer MessagesBundle_es_ES.properties
#Traducció CASTELLÀ

#Títols de les pantalles
		title=Listado de Clientes
		title.alta.clients=Alta Cliente

#Botons
		button_alta_client=Alta Cliente
		button.tornar=volver

#Errors
		error.alta.client=Se ha producido un error al dar de alta el cliente

#Columnes del llistat de Clients
		client.id=id Cliente
		client.nom=Nombre
		client.cognoms=Apellidos
		client.dni=DNI
		client.telefon=Teléfono

#Validació de formulari
		client.altaOK=Cliente dado de alta corréctamente
		client.tornar=Volver a la página principal
```

4.9 Navegació i Pantalles

En la següent figura, es mostra el diagrama d'activitat de l'aplicació web de prova del *framework*. Com es pot observar, les accions estan en color vermell, els formularis en color verd i les pàgines *jsp* en color blau.

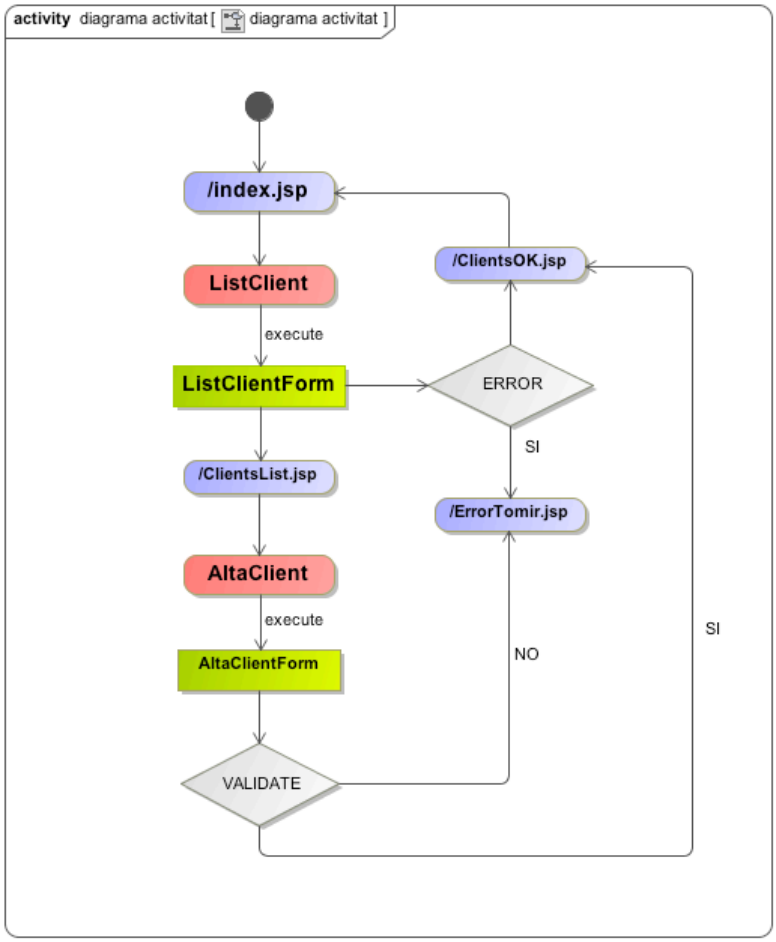


Figura 38: Diagrama d'activitat de l'aplicació web de prova

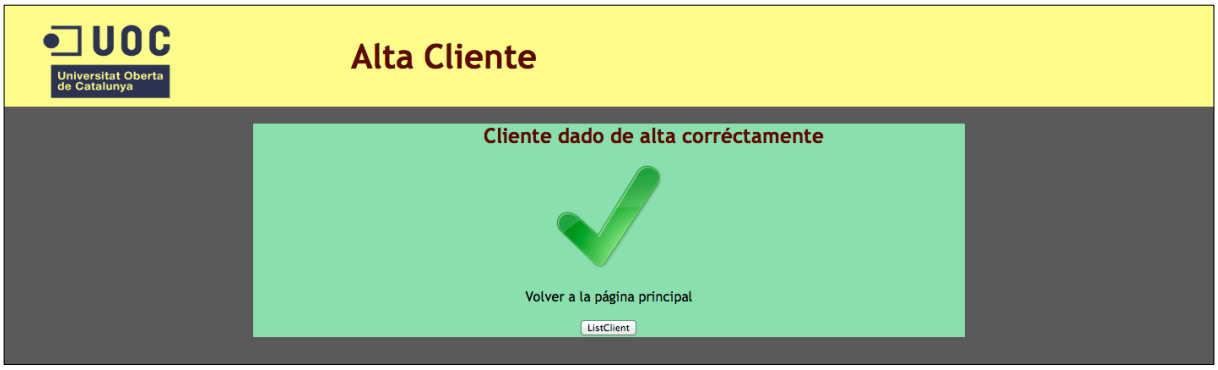
4.9.1 Pantalla inicial (clientList.jsp)

Aquesta pantalla mostra el llistat de Clients i a més, una finestra on es pot donat d'alta un nou client. Es mostra en els dos idiomes que s'han configurat (català i castellà).

En aquesta captura de pantalla, es pot veure el client nou donat d'alta :

4.9.2 Pantalla confirmació alta de client (clientsOK.jsp)

Aquesta pantalla, mostra un missatge de confirmació, indicant que el client s'ha donat d'alta de forma satisfactòria. Es mostra en els dos idiomes que s'han configurat (català i castellà).



4.9.3 Pantalla d'error (errorTomir.jsp)

Aquesta pantalla, mostra un missatge d'error, indicant que el client no s'ha pogut donat d'alta, per exemple perquè estigui ja donat d'alta, com es mostra en les captures. Es mostra en els dos idiomes que s'han configurat (català i castellà).

Capítol 5. Conclusions

Com a conclusió d'aquest projecte, es pot dir que ha sigut tot un repte de disseny i programació. Per a una persona amb poca experiència amb *J2EE, és* un projecte molt complicat i en el que s'ha de fer un gran treball d'investigació i d'ampliació de coneixements.

Per un altra banda, l'experiència i coneixements adquirits han estat enormes, i han servit per entendre el concepte i funcionalitat dels *frameworks J2EE* de la capa de presentació i la gran utilitat que representen pels desenvolupadors web, facilitant molt la seva tasca.

Un altra aspecte molt interessant en el que també s'adquireixen molts de coneixements, és en el disseny de patrons. En el present projecte, s'han utilitzat alguns patrons de disseny, com per exemple *Service To Worker*, que està compost d'altres patrons. La seva utilitat en el disseny web és molt gran, i és una de les bones pràctiques de la programació *J2EE*.

En el disseny i implementació de l'aplicació web de prova, encara que no sigui molt extensa, s'ha pogut comprovar la facilitat de disseny que ha resultat d'utilitzar el *framework TOMIR*. La implementació i dificultat ha quedat molt reduïda, ja que gran part de la feina està implementada en el *framework*.

En definitiva, un projecte molt interessant i uns coneixements adquirits molt grans, en una matèria molt utilitzada en el disseny web d'aplicacions distribuïdes en el mercat actual, com és la programació *J2EE*.

Annex A. Glossari

Apache Software Foundation : Fundació sense ànim de lucre, creada per donar suport als projectes *Apache*, incloent el popular servidor *HTTP Apache*. La *Apache Software Foundation* és una comunitat descentralitzada de programadors que treballen en projectes de codi obert. Els projectes *Apache* es caracteritzen per ser programari lliure.

API (Application Programming Interface) és el conjunt de funcions i procediments (o mètodes, en la programació orientada a objectes) que ofereix certa biblioteca per ser utilitzat per un altre programari com una capa d'abstracció . Són usades generalment en les biblioteques.

CGI (Common Gateway Interface) : és una tecnologia de la *World Wide Web* que permet a un client (navegador web) sol·licitar dades d'un programa executat en un servidor web. *CGI* especifica un estàndard per a transferir dades entre el client i el programa. És un mecanisme de comunicació entre el servidor web i una aplicació externa.

Core J2EE Patterns: Catàleg complet de patrons de disseny i arquitectònics específics per als problemes comuns en *J2EE*. També s'identifiquen males pràctiques que han evitar-se. Està organitzat en patrons de la capa de presentació, patrons a la capa de negoci i patrons de la capa d'integració.

Custom tag: etiquetes a mesura que es poden construir per utilitzar-se en pàgines *JSP*.

Dispatcher: Típicament, un controlador es coordina amb un component despatxador (*dispatcher*). Els *dispatcher* són responsables del maneig de les vistes i de la navegació. Així, un *dispatcher* escull la pròxima vista per l'usuari i els vectors de control per al recurs. Poden estar encapsulats dins el controlador directament o poden ser col·locats en un component diferent.

Framework: Estructura conceptual usada per a solucionar o conduir un problema complex. En programari, generalment consisteix en un disseny reutilitzable o subsistema que proporciona ajudes o implementacions a certs casos d'ús comuns en els programes.

HTTP (Hypertext Transfer Protocol): és el protocol usat en cada transacció de la *World Wide Web*. Defineix la sintaxi i la semàntica que utilitzen els elements de programari de l'arquitectura web (clients, servidors, *proxies*) per comunicar-se. És un protocol orientat a transaccions i segueix l'esquema petició-resposta entre un client i un servidor.

JavaBean: model de components creat per *Sun Microsystems* per a la construcció d'aplicacions en *Java*.

JDBC (Java Database Connectivity): és una *API* que permet l'execució d'operacions sobre bases de dades des del llenguatge de programació *Java*, independentment del sistema operatiu on s'executi o de la base de dades a la qual s'accedeix , utilitzant el dialecte *SQL* del model de base de dades que s'utilitzi.

J2EE: plataforma de programació per desenvolupar i executar programari de aplicacions en llenguatge de programació *Java* amb arquitectura de N capes distribuïdes.

JSP (Java Server Pages): tecnologia *Java* que permet generar contingut dinàmic per a *web*.

Locale: atribut del sistema operatiu que defineix certs comportaments relacionats amb el idioma. La configuració regional, defineix la pàgina de codis, o combinacions de bits, que s'utilitza per emmagatzemar dades de caràcters i l'ordre en què aquests es classifiquen. També defineix elements específics de l'idioma, com el format utilitzat per a dates i hores i el caràcter utilitzat per separar els decimals en els números.

Model Vista Controlador o MVC: patró d'arquitectura de programari que separa les dades d'una aplicació, la interfície d'usuari, i la lògica de control en tres capes diferents.

MVC Model 1: aproximació de la primera generació d'arquitectura a la capa de presentació dins *J2EE* on la funcionalitat de controlador i vista es realitzaven dins d'una *JSP* amb el que es trenca el paradigma *MVC*.

MVC Model 2: terme inventat per *Sun Microystems,* per descriure una arquitectura web basada en peticions *HTTP,* al que es passa la petició d'un client a un *Servlet* controlador que actualitza el model i invoca a una vista apropiada.

Plugin: conjunt de components programari que afegeixen característiques específiques a un programari existent.

POJO (Plain Old Java Object): sigla utilitzada per programadors *Java,* per emfatitzar l'ús de classes simples i que no depenen d'un *framework* en especial. Aquest acrònim sorgeix com una reacció en el món *Java* als *frameworks* cada vegada més complexos, i que requereixen un complicat muntatge que amaga el problema que realment s'està modelant. En particular sorgeix en oposició al model plantejat pels estàndards *EJB* anteriors al 3.0, en els que els "*Enterprise JavaBeans*" havien d'implementar interfícies especials.

Request: petició *HttpServletRequest*.

Response: resposta *HttpServletResponse*.

Servlet: és una classe Java en Java EE que s'ajusta a l'API *Java Servlet*, protocol pel qual una classe *Java* pot respondre a les peticions *HTTP*.

SQL (Structured Query Language): és un llenguatge declaratiu d'accés a bases de dades relacionals, que permet especificar diversos tipus d'operacions en elles. Una de les seves característiques, és el maneig de l'àlgebra i el càlcul relacional, que permeten efectuar consultes amb la finalitat de recuperar de manera senzilla informació d'interès de bases de dades, així com fer canvis.

TOMIR: nom del *framework* desenvolupat en aquest projecte.

XML (eXtensible Markup Language): és un llenguatge de marques, desenvolupat pel *World Wide Web Consortium* (*W3C),* i permet definir la gramàtica de llenguatges específics per estructurar documents grans. A diferència d'altres llenguatges, *XML* dóna suport a bases de dades, sent útil quan diverses aplicacions s'han de comunicar entre si o integrar informació.

XSLT (XSL Transformations): és un estàndard de l'organització *W3C,* que presenta una forma de transformar documents *XML* en altres i fins i tot a formats que no són *XML*. Els fulls d'estil *XSLT,* realitzen la transformació del document utilitzant una o diverses regles de plantilla. Aquestes regles de plantilla, unides al document font a transformar, alimenten un processador de *XSLT*, el qual realitza les transformacions desitjades posant el resultat en un fitxer de sortida, o, com en el cas d'una pàgina web, les fa directament en un dispositiu de presentació tal com el monitor de l'usuari.

Annex B. Bibliografia

Apache Software Foundation. **Struts Framework**. [en línia]. [Data de consulta: 10 d'abril de 2013].
http://struts.apache.org/1.x/index.html

Apache Software Foundation. **Struts2 Framework**. [en línia]. [Data de consulta: 10 d'abril 2013].
http://struts.apache.org/2.x/index.html

Johnson, Mark; Singh, Inderjeet; Stearns, Beth (2002). Designing Enterprise Applications with the J2EE Platform (2ª ed.). Palo Alto: Addison - Wesley**.**

Oracle. JavaServer Faces Technology - Documentation. [en línia]. [Data de consulta: 12 d'abril de 2013].
http://www.oracle.com/technetwork/java/javaee/documentation/index-137726.html

Spring Source Community. Spring Framework. [en línia]. [Data de consulta: 20 d'abril de 2013].
http://static.springsource.org/

DZone. Understanding Front Controller. [En línea]. [Data de consulta: 5 de Maig de 2013].
http://java.dzone.com/articles/understanding-front-controller

Wikipedia. "Comparison of web Application Frameworks". Wikipedia, the free encyclopedia. [en línia], [Data de consulta: 22 d'abril de 2013].
http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks