Disseny i implementació d'un framework de presentació per al desenvolupament d'aplicacions J2EE

Salvador Morlà Murillo
Enginyeria en Informàtica

Josep Maria Camps Riba

17/06/2013

1. Agraïments

Vull expressar el meu agraïment, i molt especialment, a la meva dona Mar i a la meva filla Elsa que durant aquests darrers anys d'estudis m'han donat tot el suport i, endemés han suportat que el poc temps lliure que he tingut l'hagi dedicat a estudiar, en especial durant el desenvolupament d'aquest projecte de fi de carrera. A partir d'ara lis dedicaré tota la meva atenció i el temps que se mereixen.

També vull expressar el meu agraïment a la UOC en el sèu conjunt, i en especial als consultors i tutors que m'han ajudat sempre que ho he necessitat.

2. Resum

El present projecte de fi de carrera se centra en el disseny d'un marc de treball per la capa de presentació pel desenvolupament d'aplicacions en tecnologies J2EE. L'ús de frameworks accelera el desenvolupament i propicia que aquests siguin més fiables i amb un manteniment més senzill.

En primer lloc se presenta un breu estudi de tres frameworks existents al mercat on se fa un resum de les seves característiques més rellevants i se descriu breument com efectuen aquests frameworks les peticions d'usuari. Els frameworks seleccionats per l'estudi seran Struts, Spring MVC i JavaServer Faces. Les conclusions extretes d'aquest estudi pretenen assentar les bases pel disseny d'un framework de la capa de presentació.

Finalment se presenta el disseny del nostre marc de treball de capa de presentació accompanyat de les principals instruccions que s'han de seguir per poder emprar-lo, i finalment, un petit tutorial de com desenvolupar una aplicació amb l'eina dissenyada.

3. Índex de continguts

4. Índex de figures

5. Cos de la memòria

5.1. Introducció

5.1.1 Justificació del PFC i context en el qual es desenvolupa: punt de partida i aportació del PFC

Les tecnologies de la informació juguen un paper molt important tant en el món empresarial com als entorns domèstics o particulars. La necessitat de les empreses i particulars per oferir i consumir serveis a través de la xarxa Internet condueix a una alta demanda de feina pels sectors de la informàtica i telecomunicacions, i en especial per als desenvolupadors d'aplicacions per entorns Web. A tall d'exemple podem anomenar els serveis oferits per les xarxes socials, serveis de correu electrònic, tendes virtuals de infinitat de productes, serveis de compra de música i vídeo, reserves de viatges i hotels, i així, un llarg etc, i que en definitiva no hem d'oblidar que són oferits per programari complexe. No obstant això, aquests serveis i aplicacions no són exclusius de la xarxa Internet ni tampoc dels usuaris particulars, ja que les empreses adopten i aposten fortament per les tecnologies Web com a arquitectura en la que basar els seus sistemes d'informació i desplegar aquest tipus de sistemes a les seves intranets.

Per la construcció d'aquests sistemes d'informació, una dels llenguatges de programació més populars és el llenguatge Java. El llenguatge Java va ser creat originalment per Sun Microsystems i el va presentar a l'any 1995. Va ser concebut com un llenguatge orientat a objectes, de propòsit general i amb un entorn de execució, la Java Virtual Machine, lleuger, gratuït i disponible per les plataformes d'explotació més populars. Això i el suport de la seva tecnologia del que se coneix com *applets*, pels principals navegadors Web de l'època (com Netscape Navigator), li va reportar gran popularitat i un gran acolliment per la comunitat de desenvolupadors. El llenguatge a experimentat nombrosos canvis des de la seva primera versió així com un notable augment en el nombre de paquets i classes. Endemés, al llarg dels anys els desenvolupadors han anat aportant les seves contribucions, de forma desinteressada, en forma d'APIs o llibreries, i en conseqüència això ha derivat en que la plataforma sigui molt extensa i se pugui trobar, en la majoria dels casos, una llibreria que faciliti la construcció de programari per la resolució de qualsevol tipus de problema. No hem de passar per alt tampoc que el fet de que el *kit* de desenvolupament sigui gratuït i que existeixin una gran varietat de IDEs i eines de suport també gratuïtes, i de qualitat, l'ha ajudat a aconseguir aquest nivell de popularitat.

Com a part de la plataforma Java trobam l'especificació J2EE, (o *Java Entrerprise Edition*), que consisteix en una especificació que permet emprar arquitectures de N capes distribuïdes recolzant-se en components de software modulars i que s'executen sobre un programari anomenat servidor d'aplicacions. Els servidors

d'aplicacions són complexes aplicacions Java que gestionen transaccions, seguretat, concurrència i en definitiva possibiliten als desenvolupadors a centrar-se en en resoldre els problemes de negoci en lloc de haver de solucionar problemes tècnics de més baix nivell. Tot això sumat al fet que és totalment possible desenvolupar un sistema empresarial amb cost zero, per l'absència de costs de llicències (del llenguatge, dels entorns de desenvolupament i de servidors d'aplicacions), li ha permès situar-se com a lider en el món del desenvolupament de sistemes informàtics per a empreses i organismes públics.

La majoria de desenvolupaments als que s'enfronten els equips de programadors segueixen la mateixa estructura i principis, de manera que sembla evident que és desitjable tenir una estructura de base estandarditzada, robusta i provada sobre la que encarar nous projectes. Aquest principi és l'argument que origina l'aparició dels frameworks. Aquests, proveeixen d'estructura i funcionalitats per alliberar als desenvolupadors de programar solucions per problemes que ja han estat resolts múltiples vegades amb anterioritat i els permeten centrar-se en donar solucions als requisits específics de les aplicacions que desenvolupen.

Com ja hem avançat, l'especificació J2EE permet estructurar les aplicacions en un nombre determinat de capes, on l'estructura de capes més representativa és la conjunció de les capes d'integració, de negoci i la de presentació. De manera molt resumida definim quina és la missió de cada una d'aquestes capes: a la capa d'integració és resolen els accessos a les dades amb les que treballa l'aplicació, la capa de negoci rep les peticions i les resol mitjançant l'execució de procediments que reprodueixen les regles del negoci concret, finalment la capa de presentació processa les peticions dels usuaris determinant les accions a executar, i presentant els resultats sol·licitats. I és aquesta capa de presentació, les seves característiques i la seva problemàtica en la que se centra el present projecte de fi de carrera, i per la que se dissenya una solució en forma framework per al desenvolupament d'aplicacions Web.

5.1.2 Objectius del PFC

Els objectius finals d'aquest projecte de fi de carrera és l'aprofundiment en els coneixements de J2EE i en especial de la capa de presentació, mitjançant la creació d'un *framework* que resolgui la problemàtica que se presenta en aquesta capa. Conseqüentment també s'espera aprofundir en l'ús de patrons de disseny.

Per assolir aquest objectius, en una primera fase s'avaluarà l'estat actual dels frameworks existents al món J2EE per la capa de presentació. Per això, ens centrarem en l'estudi intensiu dels frameworks més estesos i que gaudeixen de més acceptació a la comunitat de desenvolupadors, avaluant la manera en que resolen els problemes, i determinant els seus punts forts i febles. Fruit d'aquesta experiència i amb els coneixements adquirits determinarem les funcionalitats i serveis que oferirà el nostre *framework* de capa de presentació. Els frameworks existents que estudiarem i avaluarem seran Struts, Spring MVC i JavaServer Faces.

En una segona fase proposarem un disseny per marc de treball de presentació i realitzarem la seva implementació. Aquesta segona fase reportarà doncs el resultat de tots el coneixements adquirits fins al moment. El *framework* desenvolupat proveirà d'estructura i funcionalitats i serveis de base per la implementació de aplicacions empresarials basades en Web i te per objectiu principal principal ser una eina senzilla i pràctica amb la mira posada a la seva usabilitat.

Finalment, en la tercera i última fase del projecte ens posarem en la pell d'un desenvolupador que decideix donar-li una oportunitat al nostre framework per desenvolupar una petita aplicació d'exemple que faci ús de totes les funcionalitats i serveis que aporta. Conseqüentment, podrem analitzar quins punts febles té el nostre framework i enunciar les línies de millora futures que hauríem d'incorporar futures de cara a a la seva evolució.

5.1.3 Enfocament i mètode seguit

Per aconseguir els objectius que ens marcam s'estableix un calendari de fites, cada una de les quals se desglossa en unitats de treball més petites amb prou entitat per ser per elles mateixes un element lliurable al llarg del projecte.

Tota la responsabilitat del projecte recau damunt l'autor, el qual constitueix l'únic recurs assignat, per lo que lo que se pretén és aconseguir una cadena de lliuraments seqüencial amb que implica que el producte d'un lliurament suposi l'entrada per les fases següents.

En tot moment s'ha pretès donar al projecte un enfocament pràctic, tractant la teoria només en els casos que s'estima necessari.

5.1.4 Planificació

L'escenari en el que se planteja el projecte, que correspon a un entorn acadèmic i amb un únic recurs, i amb un temps de realització d'un semestre requereix d'una planificació ajustada. Segons els objectius del projecte i el calendari de lliuraments de productes parcials requerits, s'estableix una seqüenciació de tasques per la consecució d'objectius intermedis que se detallen a continuació:

En el primer lliurament corresponent a la PAC1 i lliurat dia 11 de març de 2013 s'inclou la introducció del projecte, una relació d'objectius a assolir i el pla de treball que marca les fites a assolir per completar el projecte amb èxit.

El segon lliurament correspondrà a l'estudi i avaluació dels frameworks existents. El lliurament consistirà en una documentació a on se posaran de manifest l'estructura, funcionalitats i relació de punts forts i febles de cada un dels marcs de treball d'estudi. La dedicació a l'estudi de cada framework serà la mateixa sense, a priori, atorgar prioritat a cap d'ells en especial. Aquest lliurament se realitzarà el 15 d'abril de 2013, com a producte d'avaluació per la PAC2.

Per al tercer lliurament, corresponent a la PAC3 i planificat pel 3 de juny de 2013 haurem treballat l'anàlisi, el disseny i la implementació del marc de treball propi de la capa de presentació.

La darrera fase del projecte tindrà com a objectiu finalitzar les tasques de documentació, (tasca que serà realitzada al llarg de tot el projecte amb l'objectiu de la confecció la memòria del projecte) i la realització de l'aplicació d'exemple.

En resum, les fites del projecte seran:

PAC1: 11 de març de 2013

- Introducció, objectius i planificació.

PAC2: 15 d'abril de 2013

- Estudi dels frameworks Struts, Spring MVC i JavaServer Faces.
- Documentar les conclusions i els coneixements adquirits.

PAC3: 3 de juny de 2013

- Anàlisi disseny i implementació del framework de la capa de presentació

Lliurament final: 17 de juny de 2013

- Implementació d'un cas pràctic en forma d'aplicació d'exemple mitjançant el frameworks desenvolupat.
- Conclusió de la memòria del projecte

El diagrama de Gantt que il·lustra la planificació i la relació de fites és el següent:

Nombre de tarea	Duración	Comienzo	Fin
PCF	**71 días**	**lun 11/03/13**	**lun 17/06/13**
Estudi de frameworks de presentació	**26 días**	**lun 11/03/13**	**lun 15/04/13**
JavaServer Faces	**8 días**	**lun 11/03/13**	**mié 20/03/13**
Investigació i estudi del framework	5 días	lun 11/03/13	vie 15/03/13
Documentació sobre el framework	3 días	lun 18/03/13	mié 20/03/13
Struts	**8 días**	**jue 21/03/13**	**lun 01/04/13**
Investigació i estudi del framework	5 días	jue 21/03/13	mié 27/03/13
Documentació sobre el framework	3 días	jue 28/03/13	lun 01/04/13
Spring MVC	**8 días**	**mar 02/04/13**	**jue 11/04/13**
Investigació i estudi del framework	5 días	mar 02/04/13	lun 08/04/13
Documentació sobre el framework	3 días	mar 09/04/13	jue 11/04/13
Documentació final del lliurament	2 días	vie 12/04/13	lun 15/04/13
Fi d'estudi de frameworks de presentació	0 días	lun 15/04/13	lun 15/04/13
Framework J2EE de capa de presentació	**35 días**	**mar 16/04/13**	**lun 03/06/13**
Anàlisi	6 días	mar 16/04/13	mar 23/04/13
Disseny	6 días	mié 24/04/13	mié 01/05/13
Implementació	21 días	jue 02/05/13	jue 30/05/13
Proves	2 días	vie 31/05/13	lun 03/06/13
Lliurament del framework	0 días	lun 03/06/13	lun 03/06/13
Aplicació d'exemple	3 días	mar 04/06/13	jue 06/06/13
Memòria	7 días	vie 07/06/13	lun 17/06/13
Lliurament final	0 días	lun 17/06/13	lun 17/06/13

Figura 1. Relació de tasques planificades

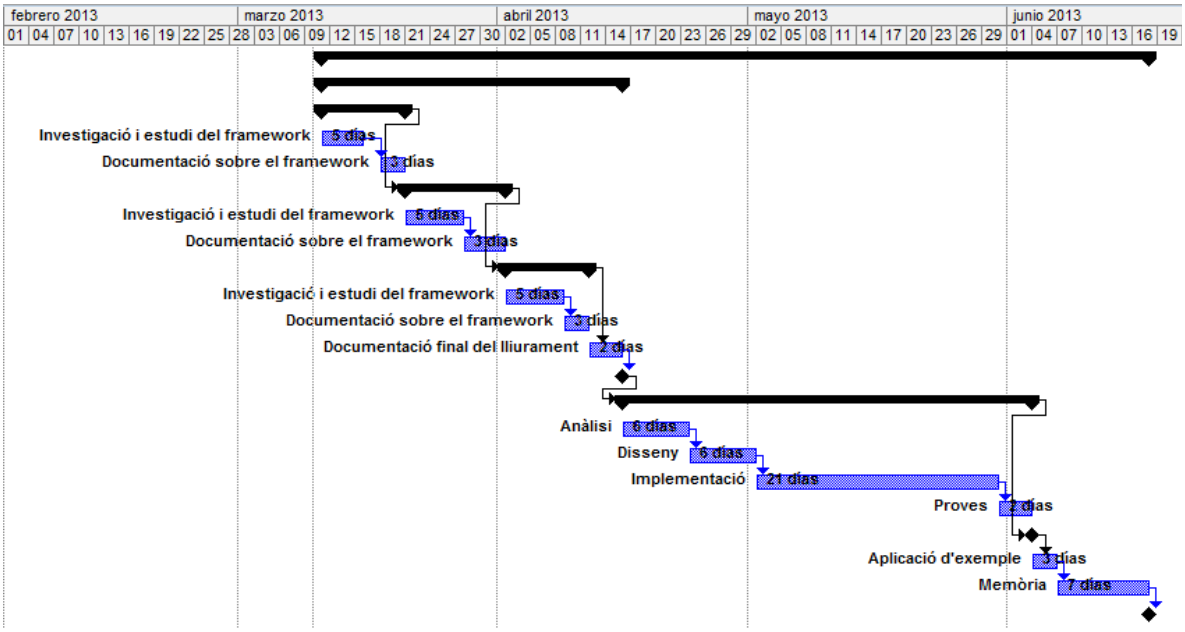

Figura 2. Diagrama de Gantt

5.1.5 Productes obtinguts

La realització d'aquest projecte produeix tres productes, un dels quals és aquesta memòria. Els altres productes seran:

SMFramework:

Constitueix la nostra proposta de marc de presentació J2EE. Se presenta en un fitxer comprimit que conté:

- Arxiu .jar amb els binaris del framework, preparats per ser inclòs com a llibreria d'una aplicació Web J2EE.

- Documentació JavaDoc per a tots els paquets i classes que composen el codi font del framework, amb descripcions tant del seu propòsit general com dels mètodes individuals.

- Codi font del framework, conservant l'estructura de paquets original i incloent la definició de classes, interfícies i anotacions.

Cas pràctic (aplicació de prova):

Com a punt final i com a tasca de la darrera la darrera etapa del projecte, s'adjunta l'aplicació web desenvolupada tenint en ment dos motius: comprovar el funcionament del SMFramework i poder determinar les possibles millores que podriem realitzar sobre aquest per extreure conclusions.

Presentació

Presentació en format OpenDocument Impress que sintetitza tota la feina realitzada per la consecució dels objectius que ens hem marcat en aquest projecte de fi de carrera.

5.1.6 Breu descripció d'altres capítols de la memòria

Estudi i comparativa de frameworks

Al capítol *5.2 Estudi i comparativa de frameworks existents*, se pretén l'estudi d'una serie de frameworks popular a la comunitat de desenvolupadors. Com a casos d'estudi s'han seleccionat els marc Struts, SpringMVC i JavaServer Faces. Antes d'abordar les descripció del mode de funcionament de cada un dels frameworks, s'introdueixen dos patrons de disseny considerats clau a tots ells, com són el patró d'arquitectura de Model-Vista-Controlador i el patró FrontController.

Per finalitzar l'estudi se presenten les conclusions, que al mateix temps conformen les bases que se seguiran per al disseny i construcció del marc de treball objecte del present projecte de fi de carrera.

Marc de treball de capa de presentació SMFramework

Al capítol 5.3 se presentarà la proposta desenvolupada com a projecte de marc de treball J2EE. En primer lloc el capítol ofereix una introducció d'una serie de tecnologies clau que se tindran en ment de cara al disseny del marc de treball.

Una vegada introduïdes les tecnologies se presenta una anàlisis de requeriments i un disseny per al *framework* que se desenvoluparà com a producte del projecte. S'inclouen un seguit de diagrames que faciliten la comprensió de certes fases clau del comportament del marc, com són la configuració inicial i el processat de les peticions. Posteriorment s'enumeren les anotacions Java que serveixen per configurar el comportament del tractament de les peticions. Finalment s'explica com se realitza la validació de dades per part del framework, com realitza el controlador la conversió de dades i com se gestionen els errors que se produeixen en el processament de les peticions.

Finalment s'aporta un petit manual d'ús, que explica com configurar el marc i com crear una classe de component de controlador.

Aplicació de prova

S'inclou en el capítol 5.4 una descripció del disseny i sensacions de la realització d'un desenvolupament d'una aplicació Web fent servir fent servir FMFramework

Millores a futures per al marc de treball

Al capítol 5.5 trobarem una primera extracte de conclusions que hem pogut extreure de l'experiència que hem tingut com a usuaris del nostre propi marc de treball de capa de presentació. En aquest capítol se proposen les millores que podríem aplicar al marc en les seves futures línies d'evolució.

Conclusió

Finalment, al capítol 5.6 s'exposen les conclusions finals del projecte que posen de manifest l'experiència viscuda durant el seu desenvolupament, els coneixements adquirits i una serie de reflexions finals.

5.2. Estudi i comparativa de frameworks existents

En el present capítol s'esposaran les característiques més importants d'alguns dels frameworks de la capa de presentació que s'han avaluat i dels que endemés, hem realitzat una comparativa per extreure les característiques més interessants. Els frameworks seleccionats han estat Struts, Spring MVC i JavaServer Faces, i la seva selecció respon a la seva popularitat i ús per la comunitat de desenvolupadors, a més de certes característiques funcionals, i de principis de disseny que esdevenen interessants per tenir-les en compte al nostre framework.

Abans de entrar en matèria, comentarem que existeixen una sèrie de característiques que comparteixen els frameworks d'estudi seleccionats. La principal és que donen suport al patró de disseny de Model Vista Controlador. Aquest patró és un patró d'arquitectura que separa les dades d'una aplicació, la interfície d'usuari i la lògica de control en tres components o capes diferenciades. Aquest patró ha esdevingut pràcticament en l'estàndard per al desenvolupament d'aplicacions Web. A continuació presentam un breu descripció dels components de MVC:

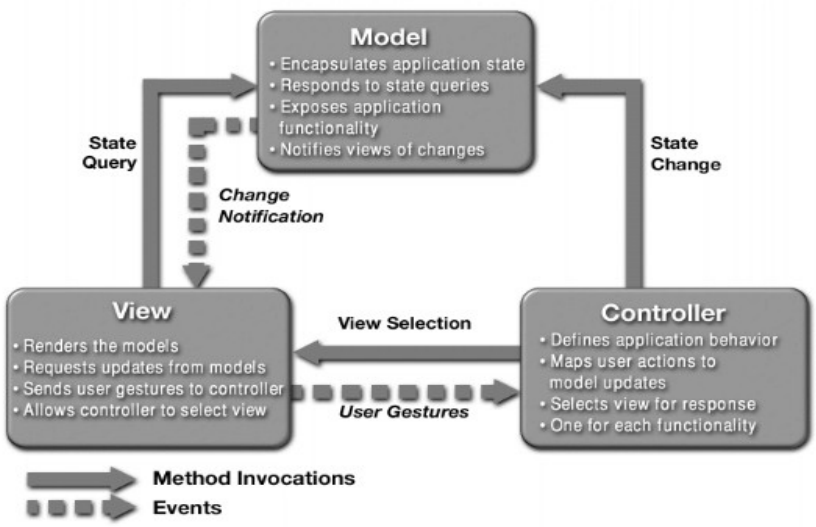

Figura 3. Esquema simple del patró d'arquitectura MVC

Model: és la representació específica de la informació i les regles de negoci que governa l'accés i l'actualització d'aquesta informació. El model no és conscient de com s'han de presentar les dades (a un navegador per exemple).

Vista: representa la capa de presentació de l'aplicació. Els objectes de la vista fan referència al model. Empra els procediments del model per obtenir els continguts i

els representa. La vista no té dependència de la lògica de l'aplicació i no se veu alterada si la lògica de negoci canvia.

Controlador: quan un usuari envia una petició aquesta se realitza a través del controlador. El controlador té la responsabilitat d'interceptar les peticions que se realitzen des de la vista i passar-les al model per que executi l'acció corresponent que obté la resposta adequada a la petició. Una vegada s'ha processat la petició pel model, el controlador ha de presentar la vista apropiada a l'usuari que va sol·licitar la petició.

Un altre característica comú dels frameworks avaluats és la implementació del patró de disseny Front Controller com a estratègia per gestionar les peticions dels clients.

El patró soluciona la problemàtica següent: quan un usuari accedeix a una vista directament sense passar per un mecanisme centralitzat que ho gestioni, cada una de les vistes tindrà la responsabilitat de proveir dels seus propis servicis de sistema incloent la generació/obtenció de continguts, la gestió de la vista i la gestió de la navegació. La conseqüència serà la duplicitat de codi per resoldre el mateix cada vegada i una organització de la navegació caòtica que implica que els sistemes esdevenguin altament complexos de mantenir. Aquesta problemàtica se pot resoldre resoldre centralitzant les peticions en un controlador com punt d'entrada pel seu procés.

Així, el FrontController proveeix d'un controlador centralitzat per gestionar les peticions que realitza l'usuari a l'aplicació des de la vista (una pàgina Web al nostre context). Un controlador frontal rep totes les peticions entrants dels clients (peticions d'usuari), remetent-les a un gestor adequat en funció de la petició, anomenat *dispatcher*, el qual s'encarregarà de gestionar la construcció d'una resposta adequada al client. En definitiva, són components de programari ideals per delegar serveis de seguretat, tractament d'errors de processament i la gestió del control per la generació dels continguts de les respostes.

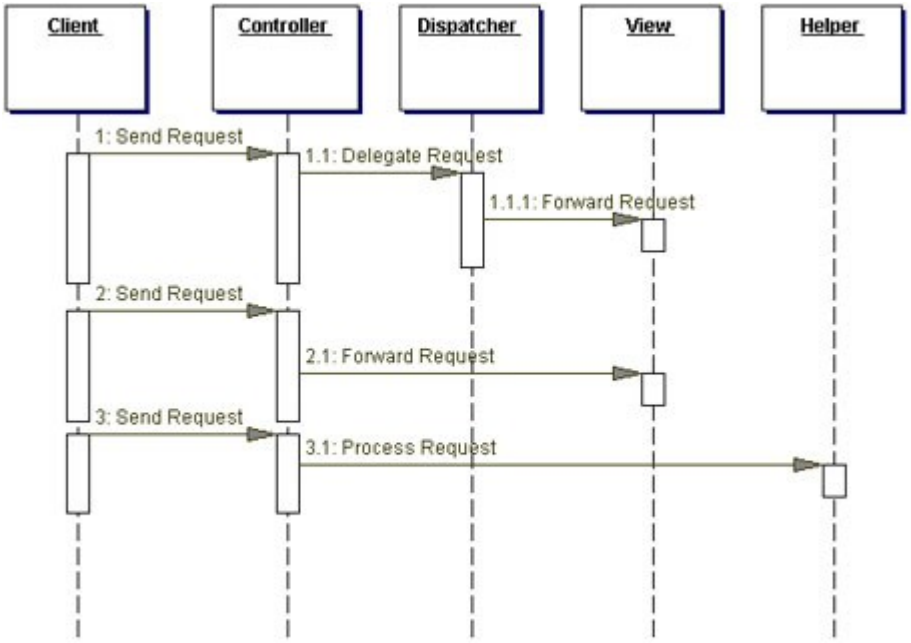

Figura 4. Diagrama de seqüencia de FrontController

Conèixer els patrons de disseny anteriorment introduïts i, tenint-los en ment, ens ajudarà a comprendre millor el funcionament dels frameworks que a continuació avaluem.

5.2.1 Struts

El marc va ser creat per Craig R. McClanahan amb l'objectiu de proporcionar un marc de treball MVC a la comunitat de desenvolupadors del llenguatge Java i va ser donat a la Apache Foundation al Maig del 2000. Tretze anys després el projecte es manté vigent i constitueix un dels marcs de treball més estesos per la creació d'aplicacions Web amb tecnologia Java.

L'objectiu principal d'Struts és separar el model (lògica de l'aplicació que interacciona amb la base de dades) de la vista (pàgines HTML que se presenten al client) i del controlador (programari que passa informació entre la vista i el model).

Disseny i mode de funcionament

Struts s'encarrega de proveir d'un controlador, implementat mitjançant un *servlet* i anomenat ActionServlet, i facilita la creació de *templates* (plantilles) per la capa de presentació (JSP típicament, encara que també se suporten altres tecnologies de presentació de continguts com XML/XSLT). El desenvolupador ha de construir el fitxer de configuració principal del framework, que és un fitxer XML i s'anomena *struts-config.xml*. Aquest fitxer essencialment s'especifica el lligam existent entre les capes de model i vista del model MVC que implementa el framework. Respecte a la capa de Model, el framework no és molt intrusiu i delega la responsabilitat sobre el programador.

Se tracta d'un framework orientat a peticions, on les peticions del client s'envien en forma de 'Actions', que s'han d'especificar al al fitxer de configuració. Quan el controlador rep la petició crida la corresponent classe de 'Action' que realitza les corresponents crides a procediments de negoci codificats a la capa de model. El codi de model retorna una 'ActionForward', que és en essència una cadena de text a on s'indica una URL la pàgina resultat que s'ha d'enviar al client. La informació viatja entre el model i la vista en la forma d'uns JavaBeans especials que per al seu tractament i representació de la informació que gestionen se recolza en una llibreria de *tags* pròpia que permet llegir i escriure el contingut d'aquests *beans* a la capa de presentació sense haver d'utilitzar codi Java embegut. Per conèixer un poc més en detall el funcionament del framework veurem a continuació la descripció del fluxe que segueixen les peticions d'usuari:

- Davant una sol·licitud d'un usuari des d'un formulari HTML, aquesta se redirigeix a una URL de la forma 'nom_accio.do'.

- Aquesta direcció se mapeja, gracies a la informació indicada a struts-config.xml en una classe Action. Aquesta classe Action implementa un mètode 'execute' que és invocat automàticament. Un dels arguments o

paràmetres que rep el mètode *execute* és un *bean* que se construeix automàticament amb les seves propietats emplenades amb els valors del formulari de la plana web des de on se iniciar la petició.

- L'objecte Action dins el seu mètode *execute* invoca els mètodes de negoci que resolen la petició.

- Finalment el resultat retornat pels processos de negoci invocats se posen a la sessió encapsulat a un *bean*.

- L'objecte Action empra el mapeig definit a struts-config.xml i mitjançant aquest mapeig, Struts redirigeix la petició al JSP adequat. Se permet definir varis *forwards* per acció per efectuar la redirecció condicionada.

Struts proveeix de la part controladora de l'aplicació. El controlador rep les peticions del usuari des del seu navegador, i decideix què lògica de negoci s'ha d'executar per després delegar la presentació al component de Vista apropiat.

El controlador Struts el composen els següents components:

- *Action Servlet*: proporcionat pel frameworks

- *Request Processor*: proporcionat pel framework

- *Classes Action*: les implementa el desenvolupador per processar cada una de les peticions que s'ha de processar a l'aplicació.

A la figura següent s'il·lustra el funcionament que hem descrit:

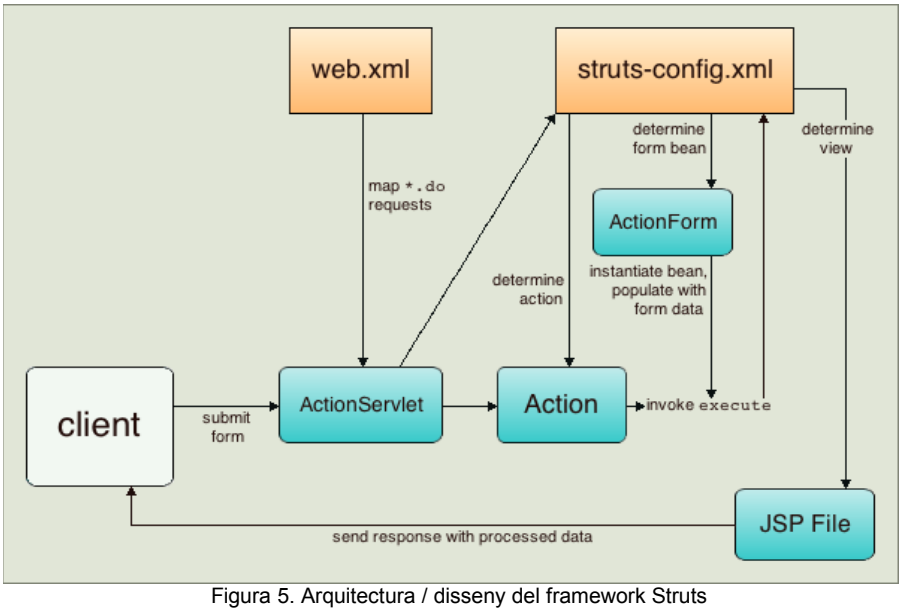

Figura 5. Arquitectura / disseny del framework Struts

Struts no és un framework dissenyat per proporcionar una solució total per al desenvolupament web, si no que el que proposa és una solució per la comunicació entre les capes de presentació i negoci. De totes maneres el seu disseny permet poder complementar-lo fàcilment amb extensions i és senzill d'integrar amb altres frameworks com solucions de persistència (Hibernate per exemple). De les funcions que proveu el framework trobam interessants les solucions per a les següents problemàtiques:

Internacionalització i localització: suport intern de la configuració de les vistes en funció de l'idioma del navegador. Els missatges s'especifiquen a un fitxer de propietats per cada idioma que se vol suportar.

Validació de dades: el framework ofereix suport per al validació del format de les dades enviades des de els formularis i encapsulades als JavaBeans ActionForms. Les dades de validació d'un formulari se defineixen dins un fitxer XML especific, a on per cada element de formulari definit al fitxer de configuració struts-config.xml s'especifica la seva regla de validació.

Conclusions

En definitiva podem dir que Struts és un framework amb un alt grau de maduresa al mercat, del que se pot trobar infinitat de documentació i exemples i tutorials a la xarxa (tant oficial como no oficial), la qual cosa fa que el seu aprenentatge sigui bastant assequible.

Un dels punts més característics del framework és que basa tota la configuració en fitxers XML. Com que quasi tota la configuració s'especifica al fitxer principal struts-config.xml a sistemes grans implica mantenir un fitxer de configuració molt extens i complexe.

Finalment, no podem deixar de mencionar que existeix també al mercat un framework conegut com Struts2. Aquest, tot i que és també un projecte de la *Apache Foundation* no prové d'una evolució d'Struts (re-anomenat com Struts1 des de l'aparició de Struts2), si no que parteix del projecte WebWork2.

5.2.2 SpringMVC

SpringMVC és un mòdul independent que complementa el framework *Spring,* i que com podem imaginar pel seu nom, aporta els components necessaris per construir programari web robust basant-nos en el patró d'arquitectura MVC. El framework va nàixer després i gràcies a la necessitats dels programadors usuaris d'*Spring* de tenir una alternativa a *Struts* per la construcció d'aplicacions Web i millora a aquest en molt d'aspectes.

Disseny i mode de funcionament

Com Struts, SpringMVC és un framework orientat a peticions i dissenyat en voltat d'un *servlet* central que despatxa les peticions als controladors, que és allà on els desenvolupadors implementen les seves accions i amb ajut de petits afegits en forma de regles de mapeig. A Spring MVC aquest *despatxador* central rep el nom de DispatcherServlet i igual que Struts, també segueix el patró de FrontController. A continuació se presenta el cicle de vida de les peticions:

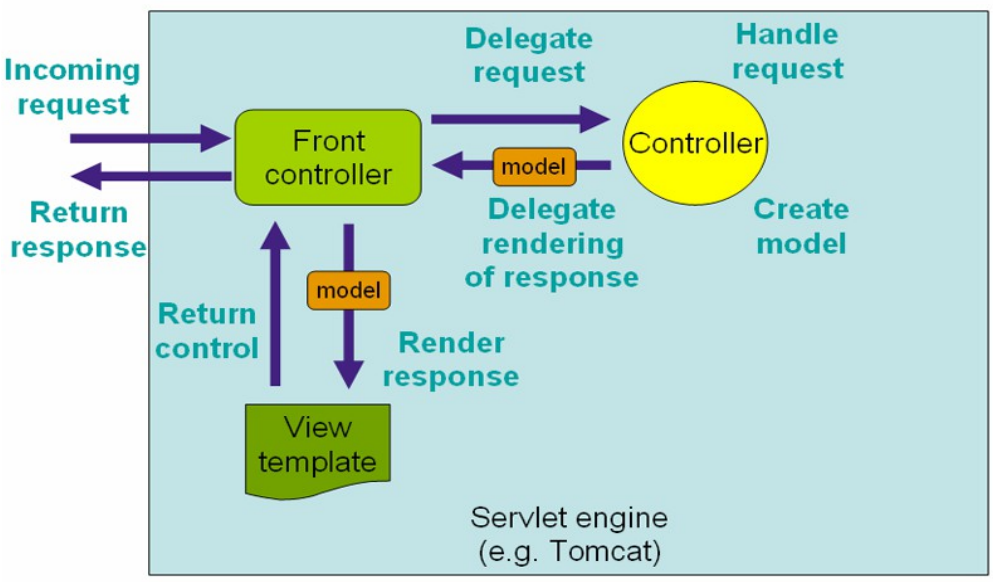

Figura 6. Fluxe del proces de peticions a Spring MVC (a alt nivell)

- El client realitza la petició de cert recurs en la aplicació Web.
- El FrontController de Spring, que està implementat com un *servlet*, interceptarà les peticions i intentarà trobar per elles els *HandlerMappings* apropiats.

- Els *HandleMappings* s'empren per mapejar una petició del client cap el seu objecte Controlador cercant per tots els Controladors definits al fitxer de configuració.

- Amb l'ajuda dels *HandleAdapters*, el *DispatcherServlet* despatxarà la petició al Controller.

- Els controladors processen la petició (request) del client i retorna l'objecte *ModelAndView* de volta cap el FrontController.

- El FrontController intenta resoldre la vista (que pot ser JSP, FreeMarker, etc.) consultant a l'objecte *ViewResolver*.

- Finalment la vista corresponent és generada i presentada al client com a resultat de la petició.

En la breu explicació del procés de resolució de les peticions s'han anomenat una serie de components (que hem ressaltat subratllant-los), que a continuació passem a descriure:

El DispatcherServlet és un *servlet* que segueix el patró de disseny FrontController i que gestiona les peticions dels clients interceptant-les i redirigint-les al controlador adequat. Un concepte associat amb el DispatcherServlet és el que anomenam *ApplicationContext.* Aquest representa normalment un conjunt de fitxers de configuració que aporten la informació de configuració a l'aplicació, i a SpringMVC el trobam en forma de fitxer XML que per defecte tindrà per nom <nom-dispatcher-servlet>-servlet.xml.

Els HandlerMappings aporten una manera abstracta de com s'han de mapejar les URL dels clients i per processar la informació que ve des de i va fins a ells una vegada processades les respostes. Dit d'una altra manera, permeten trobar el Controller que correspon segons el que s'especifica a l'arxiu de configuració.

Els HandlerAdapters defineixen a quin components s'ha de delegar les peticions perquè el DispatcherServlet trobi el HandlerMapping apropiat.

Els Controller són els components als que crida el DispatcherServlet per l'execució de qualsevol lògica de negoci.

ModelAndView és l'objecte de retorn que retornen els Controller a *Spring* al DispatcherServlet com a resultat de la petició. Aquesta classe és simplement una classe contenidora (Container) que guarda la informació del model i la vista. Qualsevol tecnologia per implementar la vista.

Els ViewResolver resolen el mapeig entre la definició de la vista (vista lògica) i la ubicació real de la vista (vista física). És a dir, s'encarreguen de, donat un identificador de vista, analitzar si aquest coincideix amb una vista implementada en una de les tecnologies de representació suportades pel framework.

A continuació presentem una figura que il·lustra la jerarquia del context al processament de les peticions a Spring.

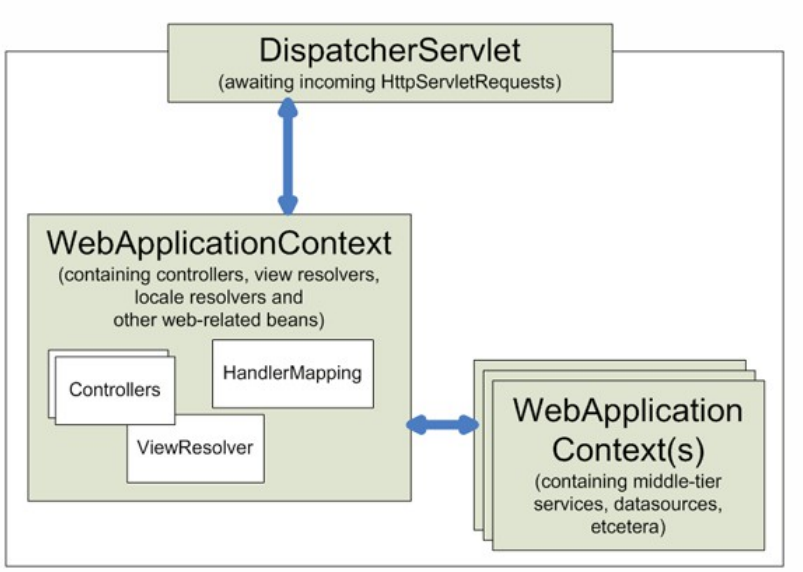

Figura 7. Jerarquia de context a Spring MVC

Altres característiques destacables del framework SpringMVC són:

Suport d'injecció de dependències per a la configuració de l'aplicació. Aquesta característica se suporta des de la versió 2.5 del marc 'pare' Spring i esdevé un gran avantatge, ja que podem substituir els llargs fitxers de configuració XML incloent les anotacions necessàries als controladors. Si volem emprar injecció de dependències o la configuració en XML és decisió del desenvolupador. Cal apuntar però, que la injecció de dependència pot realitzar-se fora fer servir anotacions, però aquestes dependències s'hauran d'indicar com a propietats dins el fitxer de configuració del ServletDispatcher, pel que aquest pot tornar-se complicat de mantenir.

Al igual que Struts també dona suport a la validació de dades dels formularis i mecanismes per la internacionalització i localització de les aplicacions.

Conclusions

SpringMVC és un marc de treball madur i àmpliament documentat amb multitud de tutorials. Malgrat això és pensam que és complexe i el seu domini requereix un domini extens de J2EE i les seves especificacions (Servlets, JSP, etc.). La quantitat de conceptes que introdueix és gran i la corba d'aprenentatge se pot considerar alta. Destaquem la possibilitat de configuració mitjançant anotacions i la flexibilitat que ofereix per integrar-se amb tot mena de frameworks.

5.2.3 JavaServer Faces

JavaServer Faces (JSF en endavant), apareix en escena amb la presentació de la seva primera versió a l'any 2004. Va ser creat per experts en el desenvolupament de la tecnologia Java i J2EE entre els que se trobava el creador de Struts, Craig McClanahan. Bàsicament, se pot definir com un marc de treball d'interfícies d'usuari del costat de la part servidora per aplicacions web desenvolupades amb tecnologia Java.

A diferencia dels altres marcs de treball que hem analitzat, JSF no és essencialment una implementació concreta de cap framework, sinó una especificació desenvolupada per la Java Comunity Process[1], determinant així els requeriments que han de complir les implementacions. Així doncs, existeixen varies implementacions de l'especificació entre les que destaca la pròpia implementació de referència (JSF Reference Implementation) creada per Sun MicroSystems, encara que existeixen implementacions amb gran popularitat com MyFaces introduïda per la Apache Software Foundation i altres com RichFaces de la Jboss Comunity, que dona suport a tecnologies AJAX.

Disseny i mode de funcionament

JSF introdueix una substancial diferencia en front als marcs de treball prèviament analitzats: està dissenyat de manera que se considera orientat a components i no a peticions. El seu funcionament se basa en un sistema basat en un controlador (dissenyat seguint el patró de FrontController) i implementat com un servlet (FacesServlet) que s'ajuda d'una sèrie de components per capturar els esdeveniments que se generen a la interfície d'usuari, processar-los i tornar la resposta adequada. Aquest processament d'esdeveniments d'interfície d'usuari s'executa a la part servidora, en lloc d'executar-se a la part client, pel que se sol considerar un framework bastant exigent i pesat per al servidor.

El cicle de vida del processament dels esdeveniments a JavaServer Faces és bastant similar al d'una pàgina JSP, encara que se completa amb processaments addicionals. A la següent figura s'il·lustra el procés:

1 Procés (o organisme) establert a l'any 1998 que recull les especificacions per la definició de les futures versions i característiques de la plataforma Java. http://www.jcp.org

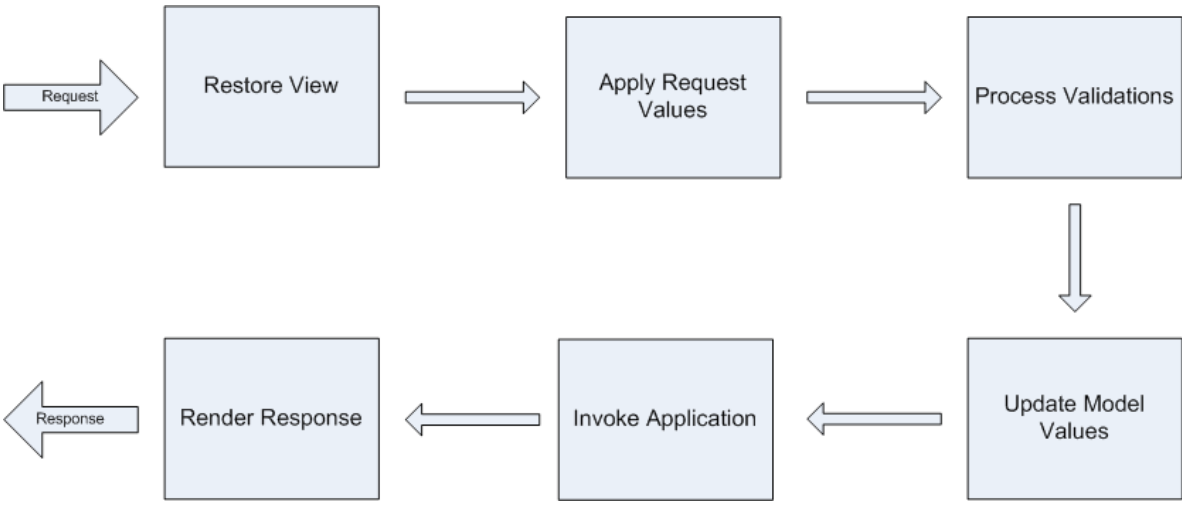

Figura 8. Cicle de vida de JavaServer Faces

El processament que té lloc a cada etapa és el següent:

- *Restore View*: comença en el moment que se realitza una petició. El seu objectiu és la creació d'un arbre que manté tots els components de la pàgina on s'origina la petició (és per això que se coneix també com Reconstitute Component Tree).

- *Apply Request Values*: cada un dels components de l'arbre fabricat a l'etapa anterior obté el valor que li correspon i s'emmagatzema.

- *Process Validations*: se validen els valors segons les regles definides per a ells. No se continua a l'etapa següent si no se satisfan totes les validacions.

- *Update Model Values*: els valors locals dels components són emprats per actualitzar els beans lligats als components.

- *Invoke Application*: s'executen les operacions que corresponen a la resolució del esdeveniment.

- *Render Response*: se representa la resposta.

A més del controlador FacesServlet el framework se completa amb unes llibreries de *tags* per facilitar la presentació de la informació del model, complementades també amb components d'interfície d'usuari (UI Components) llestos per ser emprats i que solucionen problemàtiques com la validació i la internalització de la interfície. Molts d'aquests components d'interfície són facilitats pel propi framework ja que estan continguts a l'especificació, però és possible la implementació de nous components per solucionar problemàtiques específiques.

Altres peces fonamentals del marc són el que se coneixen com ManagedBeans. Aquest són components de programari lligats als UI Components i que implementen el processament dels esdeveniments.

Conclusions

JSF és un marc de treball molt diferent en el mode de treballar que han d'afrontar els desenvolupadors a l'hora de fer feina amb ell, si el comparam amb les altres propostes analitzades. La seva orientació cap els esdeveniments i la gran intrusió que realitza sobre la vista són factors a tenir en compte.

Trobam molt positiva la baixa necessitat de configuracions basades en descriptors (XML) gràcies al suport en l'ús d'injecció de dependències amb anotacions.

5.2.4 Conclusions de l'estudi

Els textos que acompanyen les descripcions dels frameworks estudiats no són només la conseqüència de l'estudi de la documentació dels mateixos en forma d'exposició de les seves característiques més rellevants, sinó també de la realització de diversos casos pràctics de cada un d'ells en forme de tutorials o petits desenvolupaments experimentals, i que no s'han inclòs a aquesta memòria, ja que estan fora del seu abast. Endemés, l'estudi dels marc de treball ha suposat un punt d'entrada en les tecnologies clau que hem de conèixer necessàriament de cara a afrontar la constricció d'un framework per la capa de presentació J2EE, com són les tecnologies de Servlets, Filters, TagLibs, Servlet Requests i Response, etc.

Hem aprés les bases del marc d'arquitectura MVC i la seva influència en el camp del desenvolupament de les aplicacions Web i també ens ha servit per poder-mos

familiaritzar amb l'ús de patrons de disseny estudiant més en detall l'arquitectura del patró FrontController, emprat a tots els marcs revisats.

Com a conclusió de l'estudi s'enumeren les característiques que hem observat més interessants per al disseny del nostre framework:

És interessant basar el controlador del marc basat en un Servlet i seguint el patró de disseny FrontController, que intercepti les peticions de l'usuari i sigui capaç de retornar la resposta adequada a aquestes peticions. La configuració necessària perquè aquest controlador pugui realitzar la seva labor pot estar indicada mitjançant descriptors XML o mitjançant injecció de dependències mitjançant anotacions.

Tots ells introdueixen un conjunt d'interfícies per la definició dels objectes encarregats de recollir les dades de les interfícies d'usuari (formularis) per que arribin al controlador i puguin ser processades correctament i retornades a l'usuari.

Incorporen solucions per la validació de les dades provinents dels formularis, així com solucions per la internacionalització i localització de les vistes.

Faciliten la construcció de les vistes proveint de llibreries de *tags*, encara que són oberts en quant a la facilitar d'incorporar una gran varietat de tecnologies per a la representació d'aquestes.

Amb aquests ingredients, estam en condicions d'enunciar la nostra proposta de disseny per a la construcció del marc SMFramework.

5.3. Marc de treball de capa de presentació SMFramework

5.3.1 Objectius perseguits

Com hem comentat ja en aquesta memòria, un dels punts clau que suposa l'èxit d'un marc de treball és la seva corba d'aprenentatge. Si aquesta és massa alta, el nivell d'adopció del marc pot arribar a ser molt baix o ser relegat a usos molt específics, si la resta de les seves característiques i funcionalitats fa que valgui la pena el seu ús. Per tant, el que perseguim amb el SMFramework és la senzillesa, tant de configuració com d'ús.

Com a desenvolupador, el primer que ens pot donar una idea de la senzillesa del marc que estam emprant és la quantitat de fitxers de configuració que hem de parametritzar per fer funcionar una senzilla aplicació. Endemés cada un d'aquests fitxers a configurar té les seves pròpies directives que s'han d'aprendre i dominar per ser productius en el desenvolupament. Si no aconseguim ser productius amb l'ús d'un determinat framework podem concloure que aquest no s'adapta a les nostres necessitats.

El SMFramework serà un marc de treball senzill però potent que oferirà en essència un component controlador que doni suport complet al flux MVC de les peticions d'usuari i que addicionalment donarà suport per:

- Validació de dades provinents de la vista
- Control de la seguretat d'accés a accions i els seus mètodes

La senzillesa que perseguirem al SMFramework vindrà donada per la baixa presència de fitxers de configuració. Les principals tecnologies que facilitaran aconseguir aquest principi són l'ús d'Anotacions Java i Java Reflection.

5.3.2 Introducció a les tecnologies clau emprades pel desenvolupament del marc de treball

Anotacions Java

Farem ús d'anotacions Java per tal de simplificar les tasques de configuració dels diferents components del SMFramework. Les anotacions Java són interfícies especials introduïdes desde la versió 1.5 de Java, que es declaren amb el prefix @, i permeten acompanyar a paquets, classes, atributs, mètodes o paràmetres per tal d'aportar informació addicional sobre aquests.

El seu propòsit és en essència donar

- Instruccións per al compilador
- Instruccions per tenir en compte en temps de construcció dels byte-codes
- Instruccions a tenir en compte en temps d'execució.

En el nostre cas, l'ús que farem serà el darrer. Un exemple d'anotació Java seria:

```
@LaMevaAnotacio (nom=”qualqueNom”, valor = ”Hola Mon”)
public class LaMevaClasse {}
```

La classe podem observar que té indicada l'anotació LaMevaAnotacio. Mostram a continuació com se definiria l'anotació LaMevaAnotacio:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface LaMevaAnotacio {
    public String nom();
    public String valor() default “”;
}
```

Com veim, una anotació és una interfície que se marca amb el caràcter especial @. Aquesta té un nom i una sèrie d'atributs, que poden ser obligatoris o opcionals. La forma d'expressar la opcionalitat s'indica si en definir un atribut expressam el seu valor per defecte. A l'exemple, el camp 'valor' de l'anotació serà opcional, en canvi en indicar l'anotació sempre haurem d'especificar un valor per la propietat 'nom'.

Les dues directives que trobam a la definició de l'anotació @Retention (RetencionPolicy.RUNTIME) i @Target(ElementType.TYPE) especifiquen com han de ser emprades les anotacions.

L'anotació @Retention permet configurar que les anotacions es mantinguin després de la compilació del codi, ja que el seu comportament per defecte és no sobreviure una vegada generats els byte-codes en acabar la compilació. En concret, en especificar que la política de retenció a seguir és RUNTIME (temps d'execució) que l'anotació serà accessible mitjançant Reflection en temps d'execució.

Per altra banda l'anotació @Target especifica els tipus de components que se poden anotar amb la nostra anotació. En el cas concret de l'exemple, s'ha especificat @Target(ElementType.TYPE), i per tant aquesta anotació només pot ser aplicada a classes i interfaces. Se pot especificar també METHOD o FIELD que expresen que l'antació només pot ser aplicada a mètodes i variables respectivament.

Java Reflection

Reflection és un paquet paquet inclòs de sèrie en la plataforma Java que ens permet analitzar, accedir a atributs i executar mètodes de classes en temps d'execució. Gràcies a aquesta llibreria és possible avaluar, inicialitzar i utilitzar instàncies de classes que desconeixem en el moment de desenvolupar el nostre codi.

Les característiques de les funcionalitats que ofereix aquest paquet fa que sigui molt habitual el seu ús al desenvolupament de frameworks, ja que aquests han de resoldre situacions que se desconeixen en el moment del seu desenvolupament i que se donen quan aquest se fan servir com a eina per al desenvolupament d'aplicacions.

SMFramework farà servir las classes del paquet java.lang.reflect per detectar anotacions existents en les classes definides a les aplicacions que el fan servir.

5.3.3 Anàlisis de requeriments funcionals mitjançant casos d'ús

En aquest apartat expressarem l'abast funcional del SMFramework mitjançant l'anàlisi de casos d'ús. El marc de treball és un sistema que té un únic actor i que és el desenvolupador que empra el marc de treball per desenvolupar aplicacions Web, i que anomenarem, per simplificar 'Usuari'.

Així doncs segons l'abast determinat pel producte SMFramework se detecten els següents casos d'us:

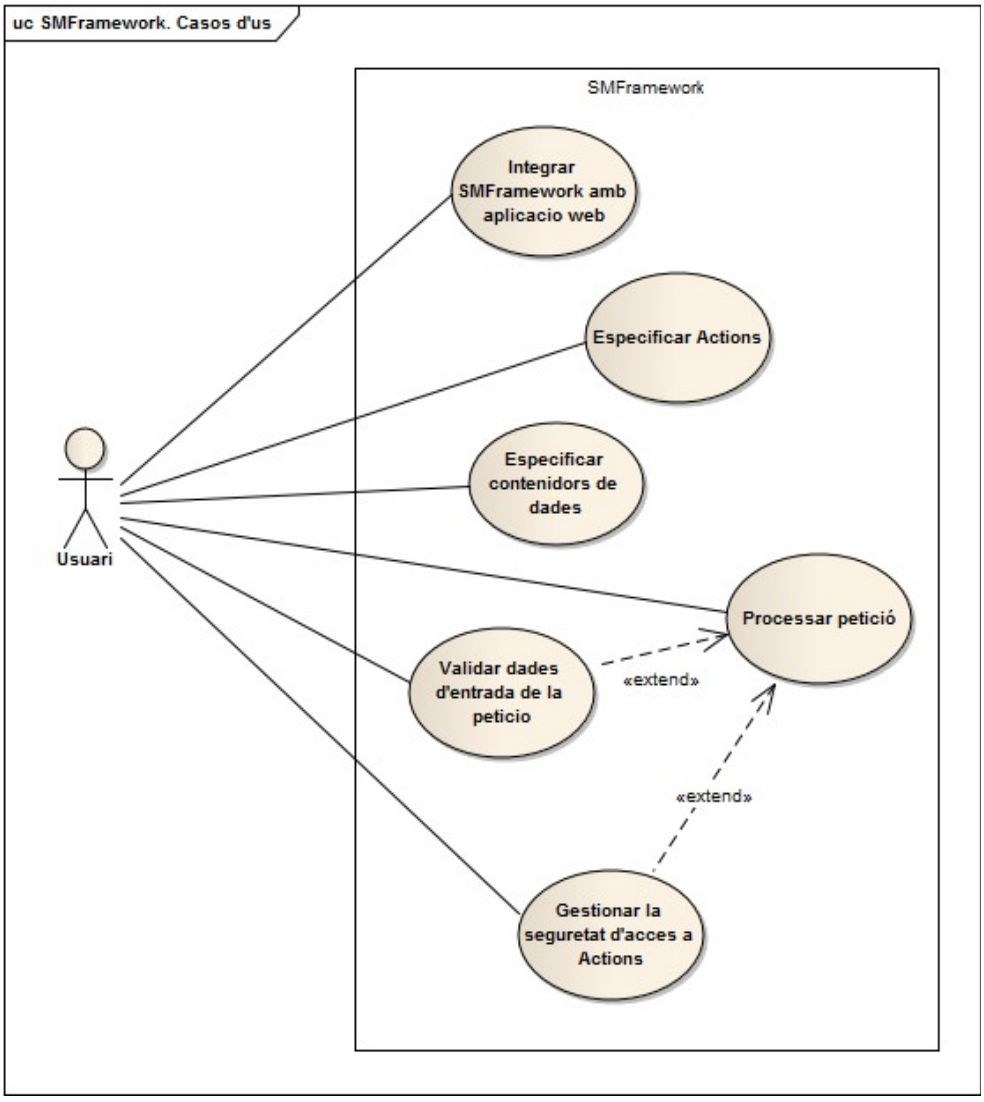

Figura 9. Identificació dels casos d'ús de SMFramework

Passem ara a especificar cada un dels casos d'ús detectats:

Cas d'us "Integrar SMFramework amb aplicació web"

Descripció general del cas d'us:

Un usuari que vol desenvolupar una aplicació web vol emprar SMFramework. Per això ha d'integrar les el producte SMFramework al seu projecte Web.

Pre-condició:

L'usuari ha de desenvolupar un projecte d'aplicació Web amb tecnologia Java J2EE

Post-condició:

El marc SMFramework queda integrat en el projecte d'aplicació Web i configurat per funcionar.

Descripció del processament a realitzar:

- L'usuari integra el producte SMFramework.jar i les seves dependències (jars que SMFramework necessita per funcionar) al conjunt de llibreries del seu projecte.
- L'usuari afegeix una entrada en el descriptor web.xml per indicar que vol emprar el controlador que proporciona SMFramework.
- L'usuari indica els paràmetres de configuració que indiquen a SMFramework les característiques de l'aplicació web i les rutes que ha de tenir en compte per capturar les peticions i carregar les vistes de l'aplicació.
- En desplegar l'aplicació en el servidor d'aplicacions, s'executa l'inicialització del controlador en funció dels paràmetres especificats.

Cas d'us "Especificar un component del controlador Action"

Descripció general del cas d'us:

Un usuari que empra SMFramework per desenvolupar la seva aplicació especifica una Action on s'especifica el processament de dades del Model.

Pre-condició:

L'usuari ha integrat SMFramework al seu projecte d'aplicació Web J2EE.

Post-condició:

L'usuari ha afegit al projecte web una classe per processar les dades del model i que, una vegada realitzat el processament especifica a quina vista s'han de retornar.

Descripció del processament a realitzar:

- L'usuari integra una classe Action com a component del controlador per a processar les peticions que se realitzaran des de l'aplicació Web que està desenvolupant.
- L'usuari especifica i implementa a la classe Action els mètodes que suporta i en el seu defecte especifica i implementa un mètode d'execució per defecte.
- L'usuari indica la vista a la que ha de dirigirà el controlador la navegació una vegada finalitzat el processament.
- L'usuari especifica la vista a la que s'ha de redirigirà el controlador la navegació en cas de que se produeixi un error.
- L'usuari especifica per a cada mètode que se defineix a la classe si com a pas previ a la seva execució el controlador ha d'ordenar la validació de les dades d'entrada.
- L'usuari especifica per cada classe i addicionalment per cada mètode d'aquesta els rols d'usuari que tenen accés a la classe i/o als mètodes.
- L'usuari especifica les dades d'entrada i sortida de la classe component del controlador.

Cas d'us "Especificar un contenidor de dades"

Descripció general del cas d'us:

Un usuari que empra SMFramework per desenvolupar la seva aplicació especifica un contenidor de dades. Aquest contenidor de dades defineix les dades que se mostren a les Vistes i que són processades a la capa de Model.

Pre-condició:

L'usuari ha integrat SMFramework al seu projecte d'aplicació Web J2EE.

Post-condició:

L'usuari ha afegit al projecte web una classe que representa les dades del model que el controlador és capaç d'oferir a les vistes.

Descripció del processament a realitzar:

- L'usuari integra un contenidor de dades com a component del controlador per a gestionar les dades de les vistes i del model per l'aplicació Web que està desenvolupant.
- L'usuari especifica i implementa a una classe, que és en essència una abstracció de les dades que se gestionen a la seva aplicació, indicant les propietats que la defineixen i els mètodes de tipus *getter* i *setter*.
- L'usuari indica per a cada propietat definida a la classe les directrius que permetran al controlador de SMFramework realitzar la validació de dades.

Cas d'us "Validar dades d'entrada d'una petició"

Descripció general del cas d'us:

Un usuari que empra SMFramework per desenvolupar la seva aplicació obté la validació de les dades d'entrada del processament provinents de la vista.

Pre-condició:

- L'usuari ha integrat SMFramework al seu projecte d'aplicació Web J2EE.
- L'usuari ha especificat a les propietats del contenidor de dades de la vista les directrius necessàries per que se validin.
- L'usuari ha especificat al mètode de la classe Action les directiva que indica que s'ha de realitzar la validació de les dades.

Post-condició:

L'usuari obté una validació de les dades efectuada pel controlador com a pas previ a l'execució dels mètodes definits a la classe Action.

Descripció del processament a realitzar:

- El controlador si estima els condicionants expressats en l'especificació de les pre-condicions realitza el processament de la validació de les dades que provenen de la vista i que constitueixen la entrada per als mètodes de les classes de component de controlador Action.
- Si durant la validació se troben errors el controlador dirigeix la vista a la plana d'error especificada. Si totes les dades són correctes, el processament segueix segons el procés ordinari.

Cas d'us "Validar seguretat en l'accés a components del controlador Action"

Descripció general del cas d'us:

Un usuari que empra SMFramework per desenvolupar la seva aplicació obté una validació d'accés en funció de la seguretat definida per l'aplicació i el control d'accés definit per un component del controlador Action o per un dels seus mètodes.

Pre-condició:

- L'usuari ha integrat SMFramework al seu projecte d'aplicació Web J2EE.
- L'usuari ha especificat a les propietats del component de controlador Action o als seus mètodes les directrius necessàries per què el controlador realitzi la validació.

Post-condició:

L'usuari obté una validació de la seguretat d'accés al component del controlador Action, efectuada pel controlador i com a pas previ a l'execució dels mètodes definits a la classe Action.

Descripció del processament a realitzar:

- El controlador si estima els condicionants expressats en l'especificació de les pre-condicions realitza el processament de la validació de la seguretat d'accés a la classe de component del controlador i/o als seus mètodes.
- Si durant la validació se determina que no s'hi té accés el controlador dirigeix la navegació a la vista corresponent informant de l'error. Si se te accés, el processament segueix segons el procés ordinari.

Cas d'us "Especificar un nou tipus de validació"

Descripció general del cas d'us:

Un usuari que empra SMFramework per desenvolupar la seva aplicació afegeix una nova classe de validació per a validar dades provinents de la vista.

Pre-condició:

- L'usuari ha integrat SMFramework al seu projecte d'aplicació Web J2EE.

Post-condició:

- L'usuari integra un nou tipus de validació de dades que serà gestionat pel controlador del SMFramework per validar les dades que s'hagin especificat del tipus definit en les propietats dels contenidors de dades.

Descripció del processament a realitzar:

- El controlador en el procés de validació descrit al cas d'ús "Validar dades d'entrada d'una petició" empra l'especificació de validació indicada per l'usuari per validar les dades marcades amb el nou tipus.
- Si el controlador troba el tipus especificat realitza la validació, sinó, avorta el procés de validació pel tipus especificat.

Cas d'us "Processar una peticio"

Descripció general del cas d'us:

Un usuari que empra SMFramework per desenvolupar la seva aplicació des de una vista sol·licita una petició de processament a la aplicació (consulta de dades, transacció de dades pel seu emmagatzemant, etc.). El controlador intercepta la petició, la resol obté la resposta adequada i la retorna a l'usuari de l'aplicació en la vista corresponent.

Es a dir, se gestiona tot el flux MVC des de l'origen de la petició fins el retorn de la resposta en la vista especificada al component de controlador Action executat.

Pre-condició:

- L'usuari ha integrat SMFramework al seu projecte d'aplicació Web J2EE.

– L'usuari ha definit els components de controlador Action que executen els processament necessaris per obtenir les dades del model dessitjades.
– L'usuari ha definit els contenidors de dades necessaris tant per solicitar el processament com el contenidor de dades de resposta.

Post-condició:

– L'usuari origina una petició que és processada correctament pel controlador de l'SMFramework i després qual. aquest redirigeix la navegació a la vista que pertoca, segons s'ha definit al component de controlador Action que s'executa.

Descripció del processament a realitzar:

– L'usuari realitza una petició mitjançant la submissió d'un formulari.
– El controlador la intercepta y carrega un contenidor de dades a partir de les dades contingudes en el formulari des de on s'origina la petició.
– El controlador determina el component de controlador Action que ha d'instanciar i quin mètode d'aquest ha d'executar per satisfer la petició.
– Una vegada processada l'acció el controlador determina la vista a mostrar com a resultat de la petició
– El controlador redirigeix la navegació a la vista determinada o alternativament a una vista on se mostra un missatge d'error en cas de que aquest se produeixi.

Alternativament, abans d'executar els procediments que satisfaran la petició i si així ha estat indicat per l'usuari en configurar el component del controlador Action, el controlador pot efectuar una validació de les dades provinents del formulari i comprovar la seguretat d'accés segons s'ha descrit en els casos d'ús anteriors.

5.3.1 Disseny de classes

Per il·lustrar una primera aproximació del marc que s'implementa se inclou el següent diagrama de classes, després del qual passarem a donar una descripció de cada una de les classes que hi apareixen.

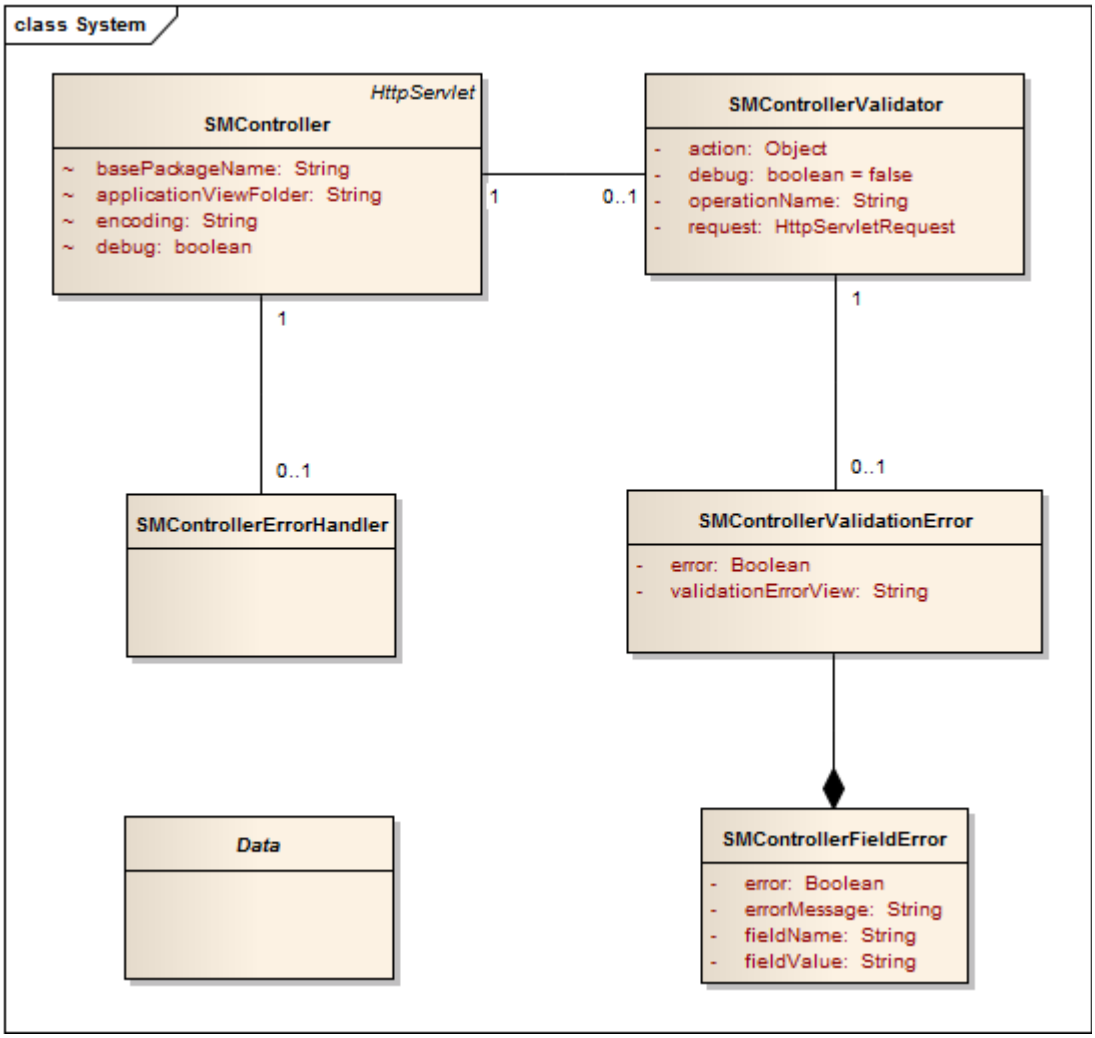

Figura 10. Disseny de classes del SMFramework

SMController

Aquesta classe representa el component principal del sistema i al·lora és la classe que implementa el patró FrontController. El seu propòsit és el de rebre i processar les peticions de l'usuari per generar una resposta adequada. En definitiva, aquesta classe gestiona tot el flux de la petició segons el que descriu el patró d'arquitectura MVC. El controlador s'implementarà com un *Servlet*.

SMControllerValidator

Aquesta classe recolza a la classe SMController i ho fa realitzant la validació de dades prèvies a l'execució dels mètodes d'un component de controlador Action. El procés de validació que executa té efecte directe en la resposta que entrega el controlador a l'usuari com a resultat de la petició ja que en funció de l'èxit o no de la validació el controlador mostrarà una vista o una altra.

SMControllerErrorHandler

Aquesta és la classe que s'encarrega de processar una excepció generada durant el processament d'una petició d'usuari. Bàsicamennt s'encarrega de rebre l'excepció i de determinar la vista encarregada de donar mostrar a l'usuari la resposta del error en funció de la definició que se fa al component de controlador Action.

SMControllerValidationError

Representa un error produït durant el procés de validació que efectua la classe SMControllerValidator per al controlador SMController. Se tracta amb una classe específica per poder informar dels errors de validació de manera personalitzada en funció del error de validació detectat.

SMControllerFieldError

Representa un error de validació particular d'una propietat o camp definit a un contenidor de dades de SMController. És una classe que està relacionada amb la classe SMControllerValidationError, que actua de contenidor d'aquesta.

Data

Representa un contenidor de dades del framework. En essència serà una classe abstracta de la que estendran tots els contenidors de dades definits per l'usuaris del framework. Entenem con una classe contenidora del nostre framework aquelles representacions d'una entitat del sistema.

5.3.2 SMController, particularitats i inicialització

Com ja hem introduït anteriorment SMController és la peça més important del marc de treball, ja que implementa un controlador seguint el patró de disseny FrontController i endemés, constitueix la part controladora en el patró d'arquitectura MVC.

Una de les decisions de disseny preses en quant al controlador és la tecnologia a emprar per la seva implementació: s'implementarà mitjançant la tecnologia de Servlets.

El SMController precisa de una sèrie d'atributs que han de ser inicialitzats en el moment de la seva creació per part del servidor d'aplicacions que l'executarà. Aquest atributs seran:

- *basePackageName*: determinarà el paquet base de l'aplicació que fa servir el SMFramework.
- *applicationViewFolder*: determina el directori que conté les vistes de l'aplicació que farà servir a SMFramework.
- *debug*: paràmetre opcional que determina si SMController extraurà per la sortida estàndard informació de debug. Serà útil a l'usuari en temps de desenvolupament.

Aquests paràmetres han de ser proporcionats a SMController i és l'única informació que necessitarà el marc de treball que sigui configurada mitjançant XML, a part de la pròpia definició del *Servlet* i el seu mapeig.

A continuació s'il·lustra el procés de configuració mitjançant un senzill diagrama de seqüencia:

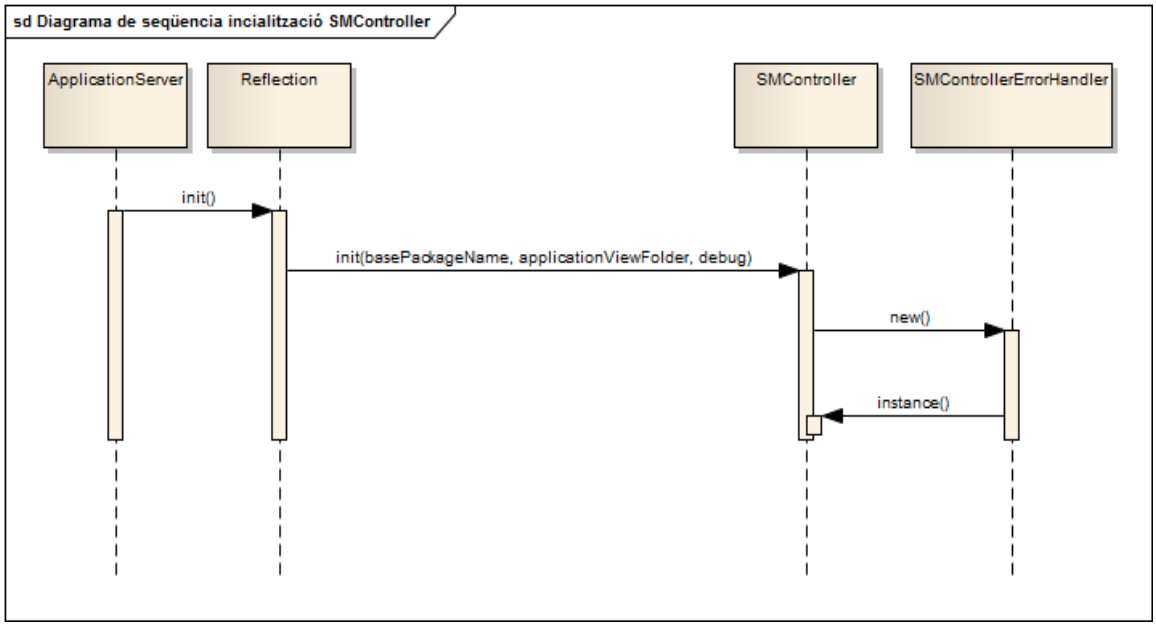

Figura 11. Inicialització de SMController

5.3.3 Processament de peticions per part de SMController

A continuació passem a descriure gràficament mitjançant un diagrama d'activitat el processament que haurà de dur a terme el controlador SMController per satisfer una petició d'usuari.

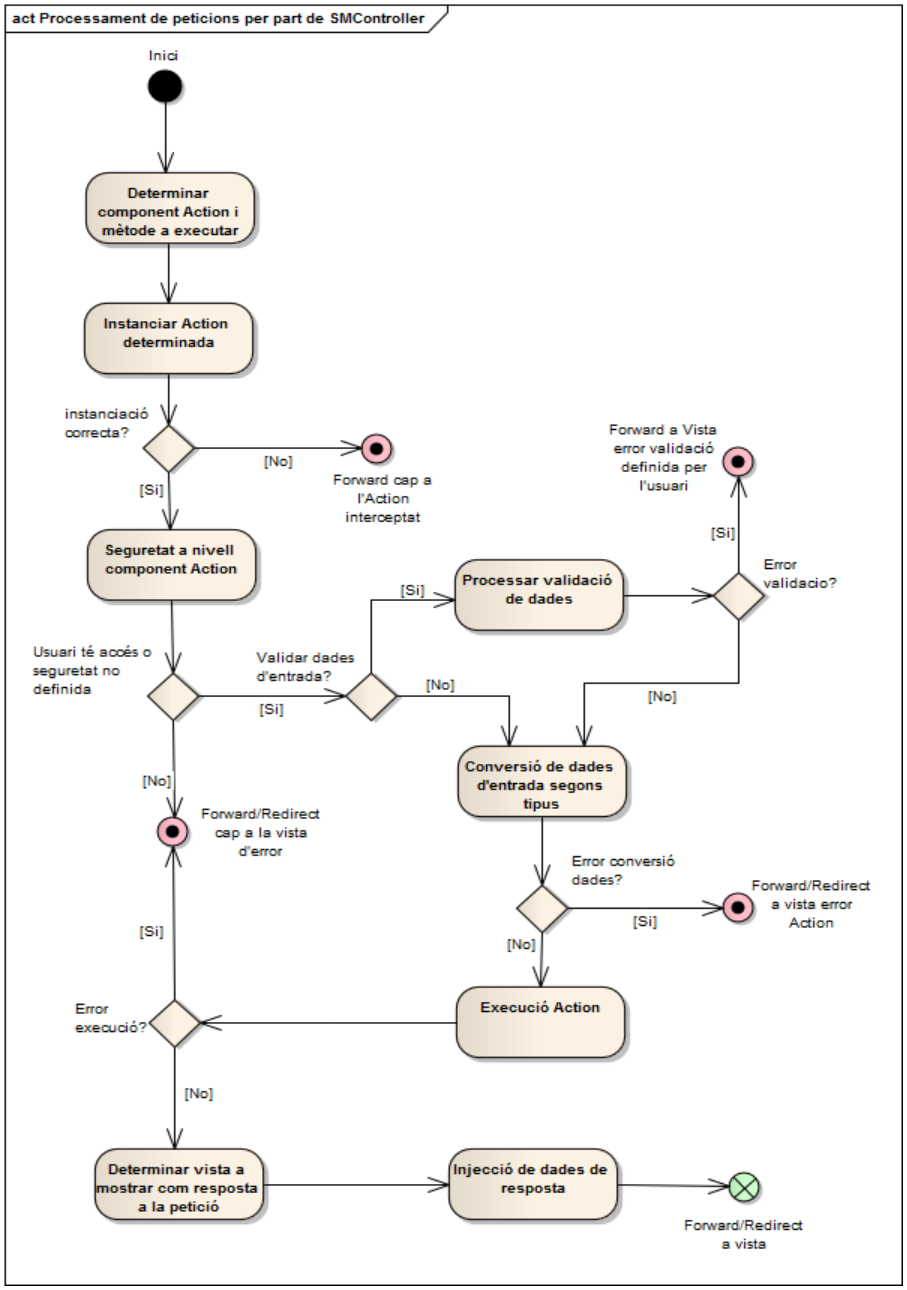

Figura 12. Diagrama d'activitat del processament de peticions de SMController

5.3.4 Configuració de components mitjançant anotacions

Como hem comentat anteriorment l'única configuració que haurem de realitzar al nostre marc de treball mitjançant XML ja la hem introduït, i hem fet només referència a la especificació del servlet del SMController, a la especificació dels seus paràmetres d'inicialització i a la definició del seu mapeig. És per tant que arribats a aquest punt introduirem els elements que ens permetran configurar els components de controlador Action, i que determinaran el comportament que realitza el controlador en el processament de les peticions.

Aquesta configuració dels components la realitzarem mitjançant anotacions Java 5, introduïdes en apartats anteriors en aquesta memòria. A continuació presentam les anotacions que podem fer servir al nostre framework descrivint el comportament que desencadenen i els seus paràmetres:

@SMControllerBean

Aquesta anotació s'emprarà per anotar atributs o camps (variables) definides a les classes de component de controlador Action. L'anotació sobre la variable implica un comportament per defecte que consisteix en que el controlador la instanciarà i intentarà mapejar tots els possibles paràmetres del request en la variable i finalment injectar-la al request com atribut.

Aquest comportament pot ser modificat amb l'especificació (opcional dels següents paràmetres):

- *create*: de tipus *boolean*, el seu valor per defecte s'estableix a *true*. Si el seu valor s'estableix a false implica que serà el mètode oportú de la classe Action el que haurà crear l'instància de l'objecte. Per exemple, si a la vista a la que el controlador ha de retornar el resultat del processament d'una petició hem de retornar altres objectes que no se poden obtenir de la petició (valors per omplir llistes desplegables, per exemple).
- *requestGetParameter*: de tipus *boolean*, amb valor per defecte *true*. En aquest cas el controlador repassa per *reflection* el contingut de la classe instanciada i tracta de donar valor als seus atributs a partir de la informació que contenen els paràmetres del request. S'indicarà el seu valor a false quan no tengui sentit incialitzar l'objecte amb la informació del request.
- *requestGetParameter*: de tipus *boolean*, el seu valor s'estableix a *true* per defecte. En aquest cas el controlador injecta al request, (i conseqüentment el posarà a disposició de qualsevol vista a la que se redirigeixi la navegació) la instància de la variable. S'indicarà amb valor false quan s'empri per

anotar dades auxiliars que no tenen sentit ni utilitat a la vista de destí en acabar el processament de la petició.

Amb això veim que els objectes anotats amb aquesta anotació constitueixen els components per passar informació entre la vista i el model.

@Forward i @Redirect

Aquestes anotacions se poden aplicar tant a nivell de classe de component Action o individualment en la declaració dels seus mètodes. Com els seus propis noms indiquen, el seu esperit és el d'indicar quina tècnica volem que empri el controlador quan ha de redirigir la navegació cap a la vista i endemés a quina d'aquestes vistes ens hem de dirigir. Ambdues tindran un paràmetre d'obligada especificació per part de l'usuari.

És a dir, si s'anota una classe de component de controlador Action amb l'anotació @Forward, estarem indicant al controlador que quan acabi l'execució que processa la petició s'ha de redirigir la navegació a la vista indicada per paràmetre mitjançant *forward.*

Si en lloc de @Forward s'anota la classe Action amb @Redirect, el controlador emprarà la tècnica *redirect* per redirigir la navegació a la vista indicada com a resposta del processament, però en aquest cas el component de vista de destí no disposarà de la informació injectada al *request*, ja que aquesta tècnica implica una petició completament nova.

Aquestes anotacions se poden indicar específicament a nivell de mètode de la classe Action. En aquest cas, prevaleix l'anotat pel mètode sobre el que s'hagi anotat a nivell de classe quan s'executi el mètode. Si no s'anota res a nivell de mètode, tots els mètodes de la classe comparteixen l'anotació de la classe.

És doncs amb aquestes anotacions com indica l'usuari al controlador a quina vista s'ha de dirigir la navegació com a resposta del processament d'una petició.

Exemple:

```
@Forward(“/jsp/empresa/LlistaEmpreses.jsp”)
```

o

```
@Redirect(“/jsp/empresa/LlistaEmpreses.jsp”)
```

@NoForwardNoRedirect

Aquesta anotació s'emprarà per indicar al controlador que no ha de redirigir la navegació a cap vista una vegada processada una petició. És una anotació que només se podrà aplicar a nivell de mètode d'una classe de component de controlador Action.

@ErrorDestination

Aquesta anotació, al igual que les anotacions de @Forward i @Redirect se pot emprar tant a nivell de classe de component de controlador Action o a nivell del seus mètodes. Com el seu nom ens suggereix indicarà al controlador a quina vista s'ha de redirigir la navegació quan se produeixi un error en el processament de la petició, per tal de que se pugui efectuar el tractament.

La anotació té dos propietats:

- *value*: obligatòria i indica la vista a la que redirigir en cas d'error.
- *redirect*: opcional i de tipus *boolean*, amb valor per defecte indicat a false. S'empra per modificar el comportament de la redirecció per especificar al controlador que volem ser redirigits a la vista que gestionarà l'error via *redirect*.

Exemple d'ús:

```
@ErrorDestination(“/jsp/error/error.jsp”)
```

o

```
@ErrorDestination(“/jsp/error/error.jsp”, redirect=true)
```

@RolesAllowed

Aquesta anotació permet establir quins rols de l'aplicació poden executar un mètode d'un component de controlador Action. També se pot emprar aquesta anotació a nivell de classe, indicant que només els rols anotats tenen accés a l'Action.

La seguretat del mòdul web no haurà de ser mai substituïda per la combinació de l'ús d'aquesta anotació, sinó que és s'ha d'emprar preferiblement la seguretat estàndard que se defineix al fitxer web.xml. La finalitat d'aquesta anotació és la de recollir casos excepcionals en els que dins un componen al que hi tenen accés diversos rols volem restringir la seguretat només a certs rols. Si un rol no té accés a l'Action, ja no tendrà accés als seus mètodes.

Exemple d'ús:

```
@RolesAllowed({“ROL_ADMIN”, “ROL_CONSULTA”})
```

@Request i @Response

Aquestes anotacions poden ser emprades per anotar aquells camps o variables de les classes de component de controlador Action si aquestes efectuen qualque processament en el que aquests components han de ser accedits.

Un cas d'ús pot ser el tractament d'un processament de paràmetres que arriben al request que no està directament suportat pel framework: en aquest cas anotaríem el mètode de l'Action amb @Request, i dintre del mètode faríem el que estiméssim convenient per processar la informació del request.

Sempres han d'anar declarades per instancies de HttpServletRequest o HttpServletResponse (segons pertoqui) i no accepten cap configuració de les seves propietats.

Exemple d'ús per una propietat d'un contenidor de dades:

```
@Request
HttpServletRequest theRequest;
```

@ValidateBean i @ValidateMethod

L'usuari haurà de combinar aquestes dues anotacions per indicar que la informació que conté un objecte declarat a un component de controlador Action i anotat com @ValidateBean s'ha de validar abans d'executar el mètode de la classe anotat amb @ValidateMethod.

En la l'anotació de @ValidateMethod s'haurà d'especificar el paràmetre següent de manera obligatòria:

– errorView: de tipus String. Especifica la vista a mostrar quan se produeix un error de validació.

Per disseny, la validació de dades que realitza el controlador s'especifica a nivell de mètode, ja que és la manera que aporta major flexibilitat, ja que dins un mateix component de controlador Action hi pot haver mètodes que precisen de la validació de dades i d'altres que no.

@ValidateField

Aquesta anotació és l'única anotació disponible a SMFramework que no s'empra per anotar elements d'una classe de component de controlador Action, sinó que la farem servir per anotar les propietats de les classes de contenidor de dades que estendran de Data. És per tant una anotació només aplicable a nivell de camp i per la que l'usuari ha de configurar las següents propietats:

– *type*: de tipus String i de especificació obligatòria. Especifica el tipus de validació que se farà sobre el camp i dependrà del tipus de camp (numèric, text, data, etc.). El *framework* ofereix una serie de tipus que se detallen més endavant en aquesta memòria. L'usuari pot definir tipus propis per completar la llista de tipus oferts per SMFramework.

– pattern: opcional i de tipus *String*. Serveix per indicar una expressió regular mitjançant la qual volem validar un camp.

– maxLength: opcional i de tipus *int*. Serveix per indicar la llargària màxima que s'accepta per a camps de tipus text.

– minLength: opcional i de tipus *int*. Serveix per indicar la llargària mínima que s'accepta per a camps de tipus text.

– maxValue: opcional i de tipus *int*. Serveix per indicar el valor màxim que s'accepta per un camp numèric.

– minValue: opcional i de tipus *int*. Serveix per indicar el valor mimin que s'accepta per un camp numèric.

– notNull: opcional i de tipus *boolean*. Serveix per indicar si el camp accepta valors no nuls o és obligatori. Per defecte la propietat pren valor 'false'.

Exemple d'ús per una propietat d'un contenidor de dades:

```
@FieldValidation (type="TextString", notNull=true, maxLength=25)
private String firstName;
```

5.3.5 Component de validació de dades de SMController

SMFramework ofereix un validador de dades d'entrada. El mode per habilitar la validació s'ha introduït a l'apartat anterior en l'explicació de les anotacions @ValidadateBean, @ValidateMethod i @FieldValidation. Segons el disseny, per efectuar aquesta labor el controlador SMController se recolza al component SMControllerValidator. Aquest component escaneja les anotacions del component de controlador Action i els contenidors de dades per determinar les validacions a realitzar i li retornarà al controlador un objecte de tipus SMControllerValidationError.

A SMFramework s'inclouran els següents tipus de validadors per indicar-los a la anotació @FieldValidation:

- AlfanumericIndentifier: cadena de text per se emprada com a identificador
- Date: data en format dd/mm/yyyy
- DateTime: data completa amb hores i minuts segons el format dd/mm/yyy hh:mm.
- EmailAddress: cadena de text que representa una direcció de e-mail.
- Numeric: valor numèric, incloent valors en punt flotant.
- NumericInteger: valors numèrics sencers.
- PositiveInteger: valor numèric sencer i positiu.
- PositiveNumber: valor numèric positiu.
- TextString: cadena de text estàndard.
- Time: cadena de text que expressa hores i minuts segons el format hh:mm.

L'usuari del marc de treball té però la llibertat de completar els tipus oferts definint nous tipus. Per això només ha de definir noves classes que implementin l'interface SMControllerDataFieldValidator i estenguin la classe SMControllerValidationType.

Un vegada definits quan se vulguin emprar, s'ha d'indicar el seu nom de classe com a valor de la propietat type de la anotació @FieldValidation.

Per exemple:

```
@FieldValidation (type="com.xxx.yyy.ElMeuTipus", maxLength=25)
```

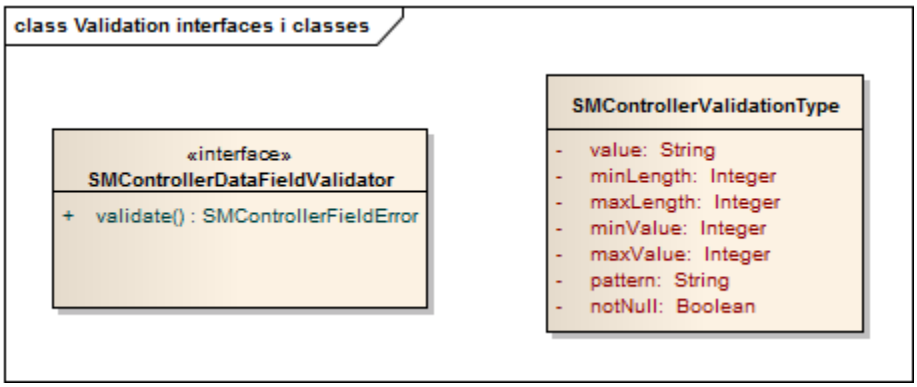

Figura 13. Interface SMControllerDataFieldValidator i classe abstracta SMControllerValidationType

5.3.6 Conversió de dades per part de SMController

En apartats anteriors hem introduït els SMControllerBean. Aquests són objectes marcats mitjançant l'anotació @SMControllerBean i constitueixen el vincle entre la capa de presentació i la capa de model (o negoci) i permeten el traspàs de dades entre capes.

A una aplicació Web, hem de tenir en compte que a la capa de vista totes les dades estan representades com cadenes de text, independentment del que vulguin representar (quantitats, dates, etc.). Això implica que perquè puguin ser processades correctament a la capa de model, el controlador ha de realitzar una conversió de dades. SMController contemplarà la conversió per als següents tipus de dades:

- Numerics: Integer, Long, Float, Double, BigDecimal.
- Dates: Date i Timestamp.

Per realitzar aquesta funció se farà servir la llibreria de BeanUtils de ApacheCommons. Les BeanUtils aporten les funcions per tal de poder construir objectes contenidors de dades (Data) del marc i donar valor a les seves propietats. Aquestes funcions són recolzades per mètodes de suport propis de SMController.

En aquesta primera versió de SMFramework, els tipus de dades que és capaç de convertir el *framework* no pot ser estès per l'usuari.

5.3.7 Gestió d'errors de SMController

Per defecte el marc SMFramework és capaç de gestionar de forma senzilla els errors que se poden produir al llarg del processament de les peticions d'usuari per part de l'SMController.

Els errors produïts són capturats per l'SMController i gestionats gràcies a un senzill component, anomenat SMControllerErrorHandler que se s'encarregarà de redirigir la navegació cap una vista on presentar el missatge d'error que s'inclou a l'excepció.

5.3.8 Classes de component de controlador Action

Segons hem descrit al llarg d'aquest capítol de la memòria les classes de component de controlador, Action, són aquelles classes que processen, o des de on se ordena el processament de la informació del sistema per satisfer una petició d'usuari des de una vista.

Segons el disseny de SMFramework, aquestes classes seran de tipus POJO, (Plain Old Java Object), que vol dir que no tenen cap dependència d'altres classes de les que hereten o han d'implementar cap interfície. Per tant, una classe Action, se declararà com una classe pública, sense més indicacions.

Les classes de component de controlador Action tenen però, la següent restricció: han de residir a un paquet específic dins l'aplicació Web i només a un. Dit d'una altra manera, el *framework* no suporta l'especificació de dos paquets diferents per indicar un paquet base des de on trobar les classes, encara que si que suporta la organització de les classes a partir d'aquest paquet base. Ho veim amb un exemple:

- org.xxx.yyy.actions i org.xxx.zzz.actions: no suportat.
- org.xxx.yyy.actions.paquet1, org.xxx.yyy.actions.paquet2, etc.: suportat.

Una classe Action pot declarar tants mètodes com l'usuari estimi necessari, però amb el condicionant de que si només en declara un, aquest ha de tenir per nom **execute().**

A continuació presentam una classe Action d'exemple a on s'especificaran també les anotacions que configurarien el seu tractament per part del controlador:

```
package org.xxx.yyy.actions.paquet1

@SMControllerBean (requestGetParameter=false, create=false)
List<UnaClasse> llista


@Forward (“/vistes/LlistaUnaClasse.jsp”)
@ErrorDestination (“/error/jsp”)
public void execute() {
    llista = new ArrayList<UnaClasse>();
    UnaClasseDAO unaclasseDao  = new  UnaClasseDAO();
    llista = unaclasseDao.list();
}
```

Aquest fragment de codi tan senzill que hem presentat correspondria a la definició d'una classe de component de controlador Action que obté del model la llista d'elements d'UnaClasse del sistema i els deixa a la variable llista.

La variable llista de la classe està anotada com a SMControllerBean, i implica que aquesta llista és la llista que se retornarà a la vista. També podem observar que s'han configurat amb la propietat `requestGetParameter=false` i `create=false`, pel que el controlador sap que no ha de crear l'instanciar ni tampoc ha d'intentar inicialitzar-la amb el contingut del request abans d'executar l'acció (ja que és la pròpia acció la que ho farà).

Veim que la vista destí a la que el controlador ha de redirigir la navegació per presentar la llista s'ha indicat amb `@Forward` i que també s'ha indicat la vista de destí en el cas de que se produeixí qualque error en l'execució amb `@ErrorDestination`.

5.3.9 Guia d'usuari

A continuació exposam una breu guia d'usuari que detalla les passes que ha de dur a terme un desenvolupador que basa la seva aplicació Web en SMFramework ha de conèixer. En primer lloc s'especifica com instal·lar i configurar el controlador del framework SMController, com ha d'organitzar les vistes, i finalment de les millors estratègies a seguir per definir les classes de component de controlador

Instal·lació i configuració del framework

Seguint el principi de màxima senzillesa de configuració que hem intentat aconseguir en el disseny de SMFramework, instal·lar i configurar el marc per que faci 'funcionar' la nostra aplicació Web consta de una serie de passes senzilles:

1. Afegir al classpath de la nostra aplicació el binari SMFramework.jar, sense oblidar incloure també les llibreries d'Apache de les que el marc en té dependència. Si estam fent servir un IDE per desenvolupar la nostra aplicació ho farem indicant els arxius *jar* com a llibreries del projecte Web que estam desenvolupant.

2. Com que estam desenvolupant una aplicació Web haurem de disposar d'un descriptor XML anomenat *web.xml.* Serà dins aquest fitxer on s'ha de definir la peça clau que vincula el projecte d'aplicació Web amb SMFramework, i que en definitiva és la declaració del servlet SMController i del seu mapeig. Un exemple de declaració se pot consultar a l'annex 8.1 d'aquesta memòria. En la declaració del servlet és important declarar correctament els paràmetres que aquest necessita per processar les peticions correctament i que corresponen al paquet base on se troben les classes 'Action' i el directori on se situaran les vistes de l'aplicació Web.

3. Opcionalment dins el descriptor, l'usuari pot declarar la seguretat.

Creació dels components de l'aplicació: vistes i classes Action

Vistes:

A continuació l'usuari ha de desenvolupar les vistes de l'aplicació en funció de la problemàtica que vol resoldre amb l'aplicació Web. En aquesta primera versió de SMFramework no s'ha inclòs cap facilitat la generació de vistes, pel que l'usuari és lliure d'emprar la que s'estimi més convenient.

Les vistes únicament tenen una restricció per al *framework*, i és que s'han d'allotjar a un directori base definit en la configuració del controlador: per exemple “/jsp”. A partir d'aquest directori l'usuari pot organitzar les vistes com millor cregui convenient en funció del disseny de la seva aplicació Web.

Action:

Els components a on se defineix el comportament de l'aplicació Web de l'usuari són les classes de component de controlador Action, i de les que ja hem fet una descripció a l'apartat 5.3.8. L'usuari tindrà la responsabilitat de definir les actions que consideri necessàries segons el disseny de la seva aplicació Web.

A continuació indicam el mode en que una vista provoca d'invocació d'una acció per part del controlador en resposta a una petició HTTP:

```
/context_aplicació/<actions>/<acció>.<operació>[?paràmetre_1[&parametre_n]]
```

on

- context_aplicació: fa referencia al context definit per l'aplicació que se desenvolupa amb SMFramework.
- actions: especificació, segons el mapeig de 'url pattern' que s'ha definit al fitxer web.xml en definir el mapeig per al servlet SMController (veure annex 8.1).
- acció: nom de la classe action que ha de resoldre la petició segons el disseny de l'aplicació Web.
- operació: operació o mètode dins la classe action. Si no s'especifica cap, el controlador executarà el mètode 'execute' de la classe *Action*.
- paràmetres: paràmetres opcionals que anirà inclosos a la petició.

5.4. Aplicació de prova.

Passem a continuació a donar-li una oportunitat a SMFramework, amb el que desenvoluparem una senzilla aplicació Web per gestionar una agenda de contactes. Mitjançant l'experiència que obtindrem al llarg d'aquesta del desenvolupament de l'aplicació explotant les possibilitats que ofereix el marc, estarem en condicions de treure conclusions de les bondats i de les millores o canvis que s'haurien d'introduir en el futur.

5.4.1 Descripció de l'aplicació

L'aplicació de proves que s'ha dissenyat és una senzilla base de dades de contactes per la gestió dels números de telèfon i comptes de correu electrònic de cada un dels contactes.

El sistema presenta a l'usuari dues vistes principals: un llistat per localitzar els contactes en el sistema i una fitxa o formulari de dades de contactes des d'on mantenir les dades de telèfons i comptes de correu electrònic. Així el sistema permetrà donar d'alta contactes i mantenir les seves dades i també especificar, modificar i eliminar números de telèfon i comptes de correu electrònic per a cada un d'ells. Finalment, des de la vista que representa aquesta fitxa de contacte podrem establir dins quins cercles classifiquem els contactes.

Cal dir que no s'han inclòs directives de seguretat d'accés a les funcionalitats de l'aplicació descrita ja que això implica que, aquesta s'hauria de acabar de configurar en funció del servidor d'aplicacions a emprar i no ens volem centrar en la gestió de que fa SMFramework de les peticions d'usuari.

En estar fora de l'abast d'aquest projecte, disposarem d'una sèrie de classes Java que ja implementen tant el model com l'accés a "dades", tot i que és només una solució senzilla que manté les dades a memòria, però que simula l'ús d'una base de dades, i que per aquesta aplicació d'exemple consideram suficient.

5.4.2 Disseny de l'aplicació

Per poder implementar l'aplicació descrita amb SMFramework cal dissenyar-la tenint en consideració les classes de component de controlador Action i els contenidors de dades. Per les vistes s'han de crear una vista de llista de contactes des de on l'usuari realitzarà les peticions de obtenir la pròpia llista, crear un nou contacte i la petició de consulta d'un contacte existent. Endemés necessitam vistes per contenir el formulari de dades d'un contacte, una vista per la gestió de les dades dels números de telèfon i una vista per la gestió dels comptes de correu electrònic del contacte.

L'estructura del projecte per l'aplicació queda com se mostra a la figura següent:

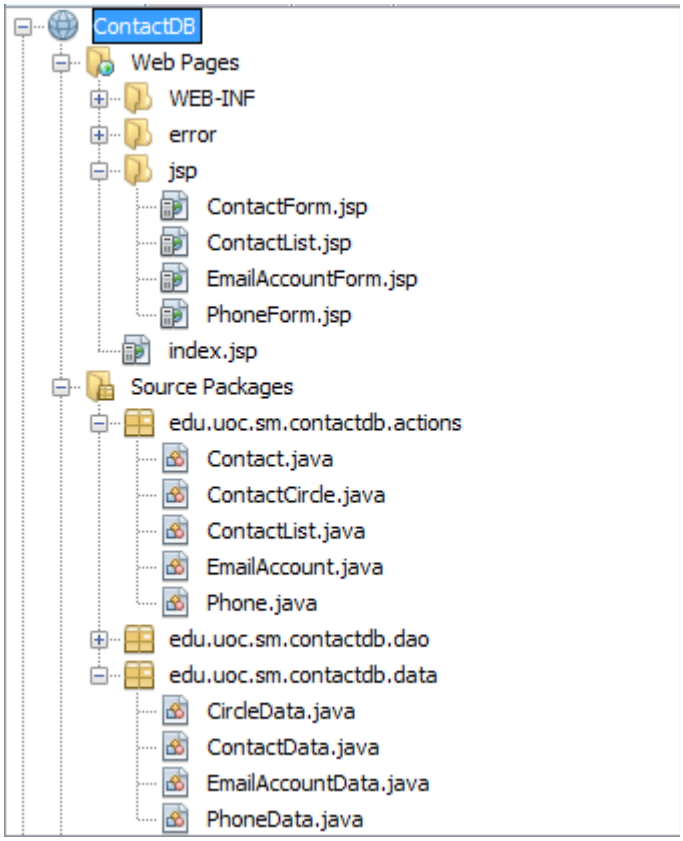

Figura 14. Disseny de l'aplicació de proves ContactDB

A la figura podem observar les classes de component de controlador Action dissenyades: ContactList, Contact, ContactCircle, Phone i EmailAccount i els contenidors de dades Data.

5.4.3 Elements clau del marc de treball

En el disseny de l'aplicació Web de proves se posa de manifest que tot gira en torn als @SMControllerBean que se defineixen a les classes de component de controlador Action. És mitjançant aquests elements que podem portar de manera transparent, i gràcies al controlador que implementa el marc de treball, les dades de la capa de vista cap a la capa de model i viceversa.

En quant al disseny de classes de component de controlador Action, veim que aquestes donen molta flexibilitat al desenvolupador, que pot optar per disposar d'una classe per cada petició o agrupar el processament de les peticions d'una 'entitat' de l'aplicació en una mateixa classe. Un exemple d'això ho veim a les classes ContactList i Contact. S'ha de destacar la senzillesa d'aquestes classes des de on se realitzen les cridades als procediments de negoci simulats per les classes d'accés a dades que s'han inclòs.

L'aplicació inclou varies mostres de com se realitza la validació dels camps de les diferents classes de contenidor de dades del sistema ContactData, PhoneData i EmailAccountData. És aquest un dels punts que manco flexibilitat aporta el marc de treball al desenvolupador. El principal punt que introdueix dificultat és la forma que en que el mac realitza el tractament dels errors del validació i la manera que té per comunicar-los a l'usuari. El sistema fa que l'usuari perdi el context (perd de vista el formulari) on se troba i que si té errors perd la informació. És llavors responsabilitat de l'usuari la de idear un sistema per evitar aquest fet.

5.4.4 Resum de l'experiència

Podríem resumir l'experiència de desenvolupar una aplicació Web amb SMFramework com satisfactòria, ja que hem pogut resoldre totes les funcionalitats que ens havíem marcat, però com és normal i amb una primera versió del marc, com a eina és millorable: el que fa ho fa bé, però podria fer més coses.

El fet de posar-se en la pell d'un desenvolupador totalment aliè al projecte de desenvolupament del marc de treball ha estat en primer terme un tant difícil, però una vegada acomodats en el nostre paper d'usuaris hem pogut verificar un dels aspectes que hem considerats com a prioritaris: la mínima necessitat de configuració i la facilitat d'ús. A banda de petites errades tècniques ja reparades, experimentar aquest nou rol ens ha obert tota una sèrie de portes en forma de possibles millores i que detallarem en el capítol següent.

5.5. Millores per futures versions de SMFramework

Les hores de possible de dedicació d'un únic recurs a un projecte de caire tan complexe i amb un abast potencial tan ampli fa que el producte obtingut sigui millorable i evolucionable en molts aspectes. El producte obtingut implementa diverses funcionalitats que ajuden al desenvolupament de projectes Web J2EE però existeixen millores i noves funcionalitats que farien del marc un producte més atractiu.

A continuació exposam les millores que hem considerat que podrien ser incloses a futures versions del marc de treball i que serveixen també d'una primera treta de conclusions per aquest projecte:

5.5.1 Millora del sistema de validació de dades

Com hem introduït al capítol en el que descrivim el desenvolupament de l'aplicació de proves, uns dels aspectes millorables és la forma de processar la validació de dades. El sistema, tal i com està dissenyat redirigeix la navegació a una vista específica des de on comunicar a l'usuari que realitza la petició si s'han produït errors de validació. Aquest fet fa que l'usuari que realitza la petició perdi el context en el que es troba (per de vista la informació sobre la que està treballant).

És doncs responsabilitat del desenvolupador que empra el marc, la tasca d'idear un sistema per minimitzar aquest efecte: una possible solució al problema anterior pot ser la de fer el *submit* de les peticions que requeriran de validació en una nova finestra emergent del navegador. Si se produeix un error de navegació el marc comunica els errors en aquesta nova finestra i l'usuari no perd el context de l'acció que estava processant. Si no se produeixen errors, el desenvolupador pot configurar una vista per representar la resposta de l'acció informant de l'èxit en el processament de la petició i provocar una recarrega de la informació de la vista (finestra pare) des d'on se va originar la petició. Cal dir que això implicaria tenir un sistema de finestres modals per controlar les possibles accions no desitjables d'un usuari.

Malgrat això, el disseny de com s'especifica la validació és millorable: una millora que suposaria una simplificació tant en el processament per part del framework com per a l'usuari desenvolupador seria la de poder indicar en l'anotació als mètodes que indica que se requereix validació (@ValidateMethod) quins elements anotats com @SMControllerBean s'han de validar. Aquesta millora soluciona un problema de la versió actual del framework: abans d'executar el mètode marcat com @ValidateMethod el controlador processa la validació de tots els @SMControllerBeans definits en la classe de component de controlador Action i

anotats amb @ValidateBean, encara que aquests @SMControllerBeans no hagin de ser processats pel mètode ni que a lo millor tengui sentit la seva validació (depèn del disseny de la classe).

5.5.2 Millora del disseny del *framework* per la conversió de dades

La conversió de les dades de la petició que provenen de la vista en objectes Java és la part del marc de treball que més dificultat ens ha suposat. Per donar solució a aquest fet ens hem recolzat a les llibreries externes d'Apache Foundation de BeanUtils.

La solució de la conversió de dades provinents de la vista és processada pel mateix controlador SMController com a part del processament de la petició, i en el que només és possible la conversió dels tipus definits (veure capítol de disseny) condiciona l'evolució de l'actual disseny, pel que la millora hauria introduir al marc és la de comptar amb un component específic per a la realització d'aquesta tasca del conversió i que permetés la conversió de tipus de dades més complexes. A mode d'exemple, en aquesta primera versió del marc només se poden convertir col·leccions de tipus bàsics. Aquesta se realitza a partir del processament del mateix paràmetre Http (paràmetre el mateix nom) per al que s'han assignat diferents valors, però actualment no podem realitzar la conversió de un Contact amb la seva llista de EmailAccount de la nostra aplicació de proves.

5.5.3 Millora del tractament d'excepcions

En el disseny i implementació que s'ha fet d'aquesta primera versió del *framework* s'ha incorporat un component per al tractament d'excepcions. Quan se produeix qualque error en el processament d'una petició d'usuari el controlador captura la aquesta excepció i redirigeix la navegació a una vista específica a on mostrar l'error, però l'error que se retorna a l'usuari només compren el missatge d'error compost pel missatge de l'excepció llançada i addicionalment el contingut de la causa de l'excepció que inclou.

Una millora seria un nou sistema de retorn de les excepcions que permetés l'extracció de totes les excepcions contingudes i que són causa de l'excepció que se retorna.

5.5.4 Suport per aplicacions multi-idioma

De l'estudi previ que se va realitzar sobre les solucions de *framework* de capa de presentació J2EE existents en el mercat varem observar com totes les solucions revisades incorporaven qualque solució per a suportar el desenvolupament d'aplicacions multi-idioma. Dit d'una altra manera, aporten mecanismes per poder particularitzar la interfície i els missatges de l'aplicació en l'idioma de l'usuari que l'empra. És doncs interessant incloure una solució a futures versions de SMFramework que permeti especificar la localització dels elements d'interfície d'usuari i de missatges.

5.5.5 Incorporar facilitats per la generació semi-automatitzada de les vistes

Una de les característiques que no s'han pogut incorporar a aquesta primera versió del marc és la d'incorporar qualque element que faciliti la generació de les vistes on se presenta la informació del model com a resposta a una petició d'usuari. Per tant una millora que un usuari agrairia seria la de comptar amb un conjunt de components en forma de llibreria de *tags* que fossin capaços de proporcionar a l'usuari automatismes per representar les dades dels contenidors de dades *Data* i de llistes de dades dins una pàgina JSP de forma senzilla, i seguint amb la filosofia de configuració amb anotacions.

5.5.6 Eina de generació de codi

Una millora notable que facilitaria molt la tasca del desenvolupador és el poder comptar amb una eina de generació de codi a partir de l'especificació de meta-informació de les entitats que se volen gestionar a l'aplicació a desenvolupar.

Una eina d'aquest estil, a partir de la definició d'una entitat i les accions que volem realitzar amb ella podria generar el codi de les classes de component de controlador Action i les classes contenidores Data i que donés així un esquelet bàsic sobre el que el desenvolupador només hagi de completar els mètodes per a l'obtenció i processament de les dades del model.

Una eina així seria un complement paral·lel al marc, però altament atractiu per als usuaris.

5.6. Conclusió

Al llarg del desenvolupament d'aquest projecte de fi de carrera, hem hagut de realitzar un exercici de canvi de papers del que estam acostumats a exercir com a desenvolupadors. Això és perquè és habitual que tant en l'experiència adquirida com a estudiants com en l'experiència professional sempre hem actuat com a usuaris d'eines per aconseguir un producte que resol una certa problemàtica que ha vengut especificada a un enunciat o bé en forma de requeriments d'un client. En aquest cas però, hem hagut d'adoptar el rol d'un creador d'eines per a que els desenvolupadors les facin servir per a construir les seves solucions, i cal dir que una vegada passada l'experiència la primera conclusió és que el desenvolupament d'un marc de treball de capa de presentació J2EE és un problema complexe.

Tot i que, com hem introduït, el desenvolupament de marcs és un tema dificultós, cal dir que en el desenvolupament d'aplicacions, donar solució a la capa de presentació és un dels punts més complexes i costosos d'implementar al món del desenvolupament de solucions Web en J2EE i pels següents motius:

- Es la part en la que l'usuari final interacciona amb l'aplicació i per tant ha de ser intuïtiva i adaptable als perfils d'aquests i endemés ser capaç de resoldre totes les situacions que aquests puguin generar en el seu ús.

- Si no se té una estructura ben definida acabam amb una lògica d'aplicació dispersa i altament replicada a tota la capa de presentació.

És per aquests motiu que l'ús d'un marc de treball esdevé gairebé imprescindible si volem construir aplicacions sòlides, estables i fàcilment mantenibles.

Una conclusió evident de la feina feta és que el desenvolupament d'una eina com un marc de treball exigeix un coneixement de la tecnologia sobre la que s'està treballant a un nivell altament superior que el nivell que cal tenir per desenvolupar una aplicació d'usuari. En el recorregut que hem realitzat sobre els marcs existents ens ha permès entrar poc a poc en matèria i anar descobrint els aspectes més interns de les tècniques i patrons de disseny que són imprescindibles d'absorbir i aprendre abans d'intentar intentar abordar el disseny d'un controlador per a aplicacions basades en el patró de disseny de model-vista-controlador.

També al llarg de l'estudi de marcs existents hem pogut experimentar amb les diferents alternatives que existeixen per abordar la manera d'establir la configuració de les aplicacions desenvolupades amb els diferents *frameworks*.

Hem vist com a opció majoritària l'ús de fitxers d'especificació de configuracions basats en XML i ens han paregut costoses de mantenir per al desenvolupament de sistemes complexes i hem aportat una solució, al nostre parer, innovadora en aquest sentit: mínim ús d'XML i configuració del comportament de l'aplicació basada en anotacions Java 1.5 i basada en un fort ús de Java Reflection.

Ja en el disseny i implementació de SMFramework ens hem hagut d'enfrontar a la resolució de temes complexos, i en aquest sentit ha de rebre una especial menció la conversió dels paràmetres provinents de la vista via peticions HTTP a classes Java. La solució donada és plenament funcional però en aquest punt ens hagués agradat aprofundir en una solució i extensible per l'usuari, tal i com hem introduït al capítol de millores.

En definitiva el desenvolupament del projecte de construcció de SMFramework ens ha proporcionat l'adquisició del molts coneixements que de ben segur seran molt beneficiosos a l'hora de poder aplicar-los en l'àmbit professional: tant pel coneixement adquirit en les diferents parts de l'arquitectura J2EE com de les característiques i funcionament de les eines existents, l'ús de patrons de disseny i finalment en les tècniques de programació en llenguatge Java relacionades amb l'exploració de classes Java per extreure la seva informació de mètodes, propietats i anotacions.

Finalment, podem concloure aquesta memòria afirmant, que dintre de l'abast que s'ha pogut assolir, hem aconseguit un marc de treball senzill però capaç de donar suport per al desenvolupament d'aplicacions de cert grau de complexitat i endemés suficientment lleuger com per complementar-lo amb altres tecnologies de capa de presentació.

6. Glossari

API: Agrupació de programari disposada en forma de mòdul independent que permet oferir serveis específics a una aplicació externa.

Applet: és un component d'una aplicació que s'executa en el context d'un altre programa, per exemple un navegador web. L'applet s'executa en un contenidor, que li proporciona un programa amfitrió, mitjançant un *plugin*.

Intranet: És una xarxa de comunicacions privada desplegada dintre d'una empresa. Pot també estar formada per varies xarxes locals interconnectades mitjançant línies de comunicació contractades d'una xarxa d'àrea estesa (wide area network).

J2EE: Especificació de plataforma de programació per a aplicacions Java amb una arquitectura de capes i executades sobre un servidor d'aplicacions.

Java: Llenguatge de programació orientat a objectes desenvolupat per Sun Microsystems als anys 90 i recolzat per una àmplia de comunitat d'usuaris. Actualment en propietat de Oracle Corporation.

Framework: Solució de base (habitualment un programari) que permet establir certes regles i automatismes per posteriorment desenvolupar una solució específica un nivell per sobre.

MVC: Patró d'arquitectura que estableix una separació entre les dades, la presentació i la lògica de negoci, parts diferenciades i comunicades entre elles.

Patrò: Un patró és una solució de disseny de software a un problema, acceptada com a correcta i que pot ser aplicada en altres contexts com una solució vàlida per un problema similar.

Vista: Component d'una aplicació corresponent a les interfícies d'usuari. També sol referir-se al conjunt de vistes com a la capa de presentació.

Meta-informació: Dades sobre una dada o informació específica d'una dada.

7. Referències bibliogràfiques

Josep Maria Camps i Riba
J2EE Una plataforma de components distribuïda
UOC Ref: P06/81059/01155

The Apache Software Foundation (2005-2011).
Struts Documentation: Key Technologies Primer.
<http://struts.apache.org/primer.html>

The Apache Software Foundation (darrera publicació 14/12/2008)
Struts. User Guide
<http://struts.apache.org/release/1.3.x/userGuide/index.html>

Deepak Alur, John Crupi, Dan Malks. (2006). Core J2EE Patterns. Best practices and Design Strategies. Second Edition.
Sun Microsystems Press.
ISBN 0-13-142246-4

Sun Microsystems (2001-2002)
Core J2EE Patterns. Front-Controller.
<http://www.oracle.com/technetwork/java/frontcontroller-135648.html>

Rod Johnson, Juergen Hoeller, Alef Arendsen, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Rick Evans
The Spring Framework Reference Documentation
Chapter 13. Web MVC framework
http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html

Moisés Daniel Díaz. (1998-2013)
Diseño de aplicaciones Internet usando los Patrones de diseño J2EE.
<http://www.programacion.com/articulo/diseno_de_aplicaciones_internet_usando_los_patrones_de_diseno_j2ee_229>

David Geary [JavaWorld.com] (2002, novembre).
A first look at JavaServer Faces, Part I & II.
<http://www.javaworld.com/javaworld/jw-11-2002/jw-1129-jsf.html>
<http://www.javaworld.com/javaworld/jw-12-2002/jw-1227-jsf2.html>

Jakob Jenkov (sense data de publicació)
Java Tutorial Language. Java Annotations
<http://tutorials.jenkov.com/java/annotations.html>

Jakob Jenkov (sense data de publicació)
Java Reflection. Java Reflection: Annotations
<http://tutorials.jenkov.com/java-reflection/annotations.html>

Arpit Shah (21/10/2012)
Undestanding Java Annotation – Annotation Examples
<http://tutorials.jenkov.com/java/annotations.html>

Cristóbal González Almirón (26/03/2009)
Introducción a JavaServer Faces
<http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=IntroduccionJSFJava>

JavaBeat (nom autor desconegut) (23/06/2007)
SpringMVCFramework with Example
<http://www.javabeat.net/2007/06/introduction-to-spring-mvc-web-framework-web-tier/>

8. Annexos.

8.1. Exemple de declaració del Servlet SMController a un descriptor web.xml

```
 <servlet>
    <servlet-name>SMController</servlet-name>
    <servlet-class>edu.uoc.pfc.sm.servlet.SMController</servlet-class>
    <init-param>
        <param-name>basePackageName</param-name>
        <param-value>edu.uoc.sm.contactdb</param-value>
    </init-param>
    <init-param>
        <param-name>applicationViewFolder</param-name>
        <param-value>jsp</param-value>
    </init-param>
    <init-param>
        <param-name>debug</param-name>
        <param-value>true</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>SMController</servlet-name>
    <url-pattern>/actions/*</url-pattern>
</servlet-mapping>
```

El descriptor mostrat és el corresponent a l'aplicació de proves de l'agenda de contactes.

8.2. Relació de lliurables

- Binari de SMFramework: smframework/dist/SMFramework.jar
- Codi font de SMFramework: smframework/src
- Documentació Javadoc de SMFramework: smframework/jdoc/jdoc.zip
- Aplicació de prova: ContactDB app_exemple_contactdb/dist/ContactDB.war (context path: /contactdb)
- Llibreries externes de dependència: smframework/extlib
- Memòria del PFC (present document): smorla_memoria.pdf
- Presentació del PFC: smorla_presentacio.ppt

8.3. Plataforma de desenvolupament emprada i entorn d'explotació.

El programari que se lliura com a producte resultant d'aquest projecte de fi de carrera ha estat desenvolupant fent servir:

– IDE Oracle NetBeans versió 7.2.1

L'aplicació ContactDB d'exemple ha de ser desplegada a un servidor d'aplicacions compatible amb l'especificació J2EE. Per comoditat s'ha fet servir el servidor d'aplicacions Glassfish versió 3.1.2, que ve integrat al IDE NetBeans versió 7.2.1.

Nota: és necessari que el servidor d'aplicacions a on se desplegui l'aplicació d'exemple s'inclogui la llibreria de J2EE.