Universitat Oberta de Catalunya

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

PROJECTE FI DE CARRERA

Daniel Clemente Marcè

Enginyeria d'Informàtica (2on cicle)

Josep Maria Camps Riba

gener de 2011

SOME RIGHTS RESERVED

Aquest treball està subjecte - excepte que s'indiqui el contrari- en una llicència de Reconeixement-NoComercial-SenseObraDerivada 2.5 Espanya de Creative Commons. Podeu copiar-lo, distribuir-los i transmetre'ls públicament sempre que citeu l'autor i l'obra, no es faci un ús comercial i no es faci còpia derivada. La llicència completa es pot consultar en http://creativecommons.org/licenses/by-nc-nd/2.5/es/deed.es.

Apartat
1

RESUM

Aquest document correspon a la PAC2 del PFC titulat “Disseny i implementació d’un marc de treball (framework) de presentació per aplicacions J2EE” emmarcat dins de l’àrea J2EE. En aquesta segona PAC, tal i com vaig indicar en la primer PAC o pla de treball, he realitzat un estudi i avaluació dels frameworks existents.

En aquest estudi, he començat per introduir i descriure el concepte de patró. Aquest concepte és imprescindible per entendre la funcionalitat que volen oferir els marcs de treball. Entre les descripcions de patró he volgut incloure casos particulars de patrons que més endavant trobem funcionant en productes estudiats.

Un cop conclòs el punt sobre patrons, he volgut seguir amb els conceptes necessaris per comprendre els marcs de treball. De la mateixa manera que volem trobar semblances entre marcs de treballs, tenim que podem definir aquestes semblances i diferències dins de l’ús de frameworks i les podem definir, per només haver de fer referència més endavant.

He volgut descriure la tecnologia JEE per tal de poder definir conceptes que més endavant necessitaré citar. La tecnologia JEE té una sèrie de característiques que els frameworks exploten de forma diferent, donant lloc a diferents productes amb característiques els fan singulars.

Un punt és l’anàlisi i l’estudi dels diferents marcs de treball. Els frameworks estudiats són JSF, Spring MVC, Struts, Struts 2/WebWork. I finalment, en aquest darrer punt, faig una comparació entre Struts 1 i 2 i Struts amb Spring MVC.

A partir d'aquesta PAC, vull continuar en una següent PAC fent el disseny i implementació d'un framework que faci aquesta tasca. Aquesta tasca prèvia, que ha estat aquesta PAC, em servirà per saber que ha de fer un framework d'aquest tipus, que ha de tenir en compte a l'hora de dissenyar-ho i quines tècniques utilitzen per poder imitar-les.

Apartat
2

INDEX

2.1 Taula de continguts

1.RESUM 2

2.INDEX 4

2.1 Taula de continguts 4

2.2 Índex de figures 7

2.3 Índex de taules 8

3.INTRODUCCIÓ 9

3.1 Justificació del PFC i context en el qual es desenvolupa..... 9

3.2 Objectius del PFC 9

3.3 Enfocament i mètode seguit 10

3.4 Planificació del projecte 11

3.5 Productes obtinguts 12

3.6 Breu descripció dels altres capítols de la memòria 12

3.6.1 Estudi i avaluació de frameworks de presentació 12

3.6.2 Implementació d'un framework 13

4.ESTUDI I AVALUACIÓ DE FRAMEWORKS DE PRESENTACIÓ 14

4.1 Patrons 14

4.1.1 Introducció 14

4.1.2 Objectius patrons 14

4.1.3 Problemàtica patrons 15

4.1.4 Patró de disseny 16

4.1.5 El patró observador 17

4.1.6 El patró Model Vista Controlador 18

4.2 Marcs de treball 21

4.2.1 Introducció 21

4.2.2 Ús de frameworks 22

4.3 Arquitectura JEE 22

4.3.1 Introducció 22

4.3.2 Programació eficient 23

4.3.3 Extensibilitat 24

4.3.4 Integració 24

4.3.5 Contenidors 25

4.3.6 API 26

4.3.7 Servidor d'aplicacions JEE 26

4.3.8 Servlet, HTTP i el paradigma petició-resposta 26

4.4 Marcs de treballs de presentació per a aplicacions J2EE 29

4.4.1 JavaServer Faces JSF 29

4.4.2 Spring MVC 33

4.4.3 Struts 37

4.4.4 WebWork 2.2/ Struts 2.0 42

4.5 Comparació 45

4.5.1 Struts 1 i Struts 2 45

4.5.2 Comparativa entre Struts i Spring MVC 47

5.IMPLEMENTACIÓ D'UN FRAMEWORK 48

5.1 Descripció 48

5.2 Implementació MVC 48

5.2.1 Patró MVC 48

5.2.2 Fitxers de configuració 51

5.2.3 Fluxe 53

5.2.4 Analogia amb Struts 56

5.3 Funcionalitats 58

5.3.1 Simulador de login 58

5.3.2 Restricció Usuaris 58

5.3.3 Mode debug 59

5.4 Resultats 63

6.CONCLUSIONS 64

7.GLOSSARI 66

8.BIBLIOGRAFIA 71

8.1 Fonts utilitzades 71

9.ANNEXOS 73

9.1 Manual del desenvolupador 73

9.1.1 Preparació 73

9.1.2 Desenvolupament 74

2.2 Índex de figures

Figura 1: Diagrama de Gantt de les principals fites 12

Figura 2: Diagrama d'actualització de classe en el patró Observer 17

Figura 3: Diagrama de seqüències pel patró Observer 17

Figura 4: Diagrama UML del Patró Observador 18

Figura 5: Diagrama MVC on la línea puntejada és l'aplicació del patró Observador 19

Figura 6: Cas d'ús a MVC 20

Figura 7: Seqüència del patró MVC. 20

Figura 8: Estructura de l'arquitectura JEE 25

Figura 9: Paradigma petició-resposta amb un servlet 27

Figura 10: Cicle de vida d'un servlet 28

Figura 11: Seqüència JSF 32

Figura 12: Cicle de vida JSF 33

Figura 13: Funcionament del Spring MVC 35

Figura 14: Diagrama de classes del framework Struts 38

Figura 15: Estructura d'Struts 39

Figura 16: Capa de Control d'Struts 41

Figura 17: Diagrama de fluxe d'Struts 42

Figura 18: Esquema funcionament d'Struts2 45

Figura 19: Diagrama UML del servlet 50

Figura 20: Interacció Navegador Servlet 51

Figura 21: Flux del Framework desenvolupat 55

Figura 22: Flux amb aparició d'errors 60

Figura 23: Logotip NetBeans 73

Figura 24: Captura de pantalla NetBeans IDE 6.9.1 74

2.3 Índex de taules

Taula 1: Fites i temporització 11

Taula 2: Fitxers de configuració 53

Apartat
3

INTRODUCCIÓ

3.1 Justificació del PFC i context en el qual es desenvolupa

Avui en dia, per desenvolupar aplicacions web, estem obligats a que segueixin una sèrie de qualitats per tal de fer-les útils. Aquestes són eficiència, extensibilitat, reutilització i portabilitat. De no ser així, el nostre programari no tindrà lloc al mercat. Per tal d'aconseguir aquestes qualitats, cal seguir les regles establertes i utilitzar frameworks. Aquests frameworks implementen patrons de disseny. Aquest patrons foren creats al 1995 i avui en dia, són la norma pel desenvolupament. Per altra banda, Java és una plataforma molt estesa que ens proporciona l'avantatge de crear un programari amb les qualitats prèviament descrites.

El desenvolupament d'un framework és un tasca per la qual el desenvolupador ja no ha de pensar en el resultat final, sinó en el programador que l'utilitzarà, canviant el seu rol de desenvolupador per arquitecte. La realització d'un Projecte Final de Carrera és una fita dins de la carrera acadèmica que marca la fi d'un cicle i en ell s'ha de demostrar maduresa, experiència i criteri. La realització d'aquest projecte és una fita dins d'aquesta carrera i la finalització d'aquesta demostra aquestes qualitats que ha de tenir un professional de l'enginyeria informàtica.

3.2 Objectius del PFC

Amb la conclusió d'aquest projecte (i com a conclusió dels meus estudis en enginyeria informàtica) espero obtenir uns coneixements més profunds en la capa de presentació, no sols a nivell de desenvolupador, sinó d'arquitecte.

L'estudi d'una sèrie de frameworks, no sols vol dir, adquirir el coneixement de les diferents eines del mercat, sinó entendre com ho fan. Això, significa poder allunyar-me de la posició del desenvolupador per poder tenir una visió d'arquitecte.

L'ús d'un framework, si es fa seguint les recomanacions i criteris indicats, vol dir assegurar un èxit en el programari. Aquest èxit es produeix gràcies a la utilització de patrons, a més de la maduresa d'un producte àmpliament testat i que ha crescut amb la retroalimentació de molts professionals. Addicionalment, l'ús de patrons garanteix un futur d'aquest programari construït, proporcionant una estructura sòlida, a més de les múltiples avantatges derivades del seu ús.

Començar a treballar com un arquitecte és per a mi, l'objectiu intrínsec d'aquest projecte. Com he comentat anteriorment, dissenyar un framework és allunyar-me de la tasca del programador, per pensar com oferir-li una eina: què necessita, com fer-ho, quina opció triar,...

3.3 Enfocament i mètode seguit

El projecte ha seguit les pautes i recomanacions marcades per la realització de PFC a la UOC.

En primer lloc, emmarcat en la primera Prova d'Avaluació Continuada, s'ha establert un pla de treball. En aquest pla, s'ha establert la temporalització del projecte (coincidint amb les recomanacions de la UOC) i la definició del projecte.

En una segona part, s'ha realitzat un estudi de les alternatives que ja existeixen. Per tal de realitzar l'estudi, he hagut d'analitzar el context que marca la tecnologia, per tal de poder introduir els conceptes que s'han treballat. I com a conclusió d'aquesta segona part, he realitzat una comparativa entre els principals productes que implementen un framework MVC en Java.

A la tercera i darrera part, he dissenyat un nou framework aplicant els principis estudiats i triant les característiques que més m'han cridat l'atenció, posant èmfasi en els punts que més m'han agradat. A més de dissenyar el framework, he creat una aplicació de demostració que em servia per anar desenvolupant i depurant els errors en el procés de construcció.

3.4 Planificació del projecte

Per a la realització d'aquest projecte, he creat una taula amb les principals fites, completant-la amb un diagrama de Gantt. He creat les fites intentant dirigir-les a les entregues de les PAC, tal i com es recomana.

Tasca	Durada	Data inici	Data fi
Reunió de Kickoff	1 dia	Dijous 30/09/10	Dijous 30/09/10
Planificació i estudi de tasques	4 dies	Divendres 01/10/10	Dimecres 06/10/10
Estudi i avaluació dels frameworks existents	26 dies	Dijous 07/10/10	Dijous 11/11/10
Disseny i construcció d'un framework propi	27 dies	Divendres 12/11/10	Dilluns 20/12/10
Documentació i presentació	14 dies	Dimarts 21/12/10	Divendres 07/01/11
Entrega	1 dia	Dilluns 10/01/11	Dilluns 10/01/11

Taula 1: Fites i temporització

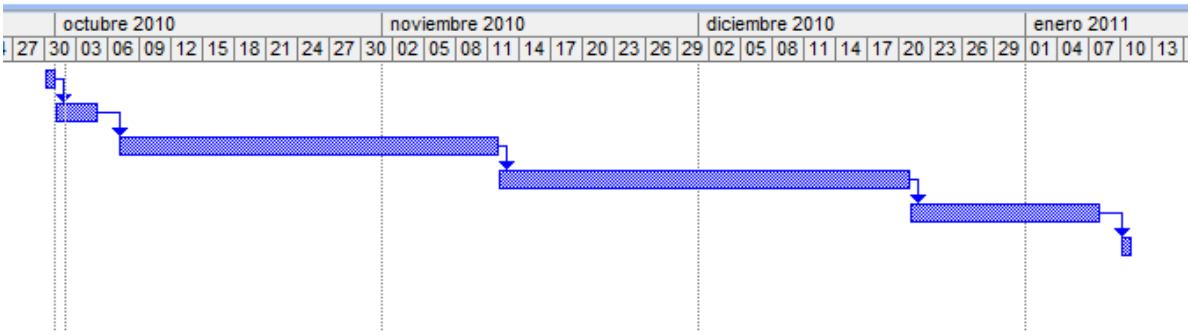


Figura 1: Diagrama de Gantt de les principals fites

3.5 Productes obtinguts

Els productes resultants, a més d'aquesta memòria, són el framework desenvolupat i una aplicació d'exemple. D'ambdós s'inclou, el codi font de les classes, interfícies, pàgines web i fitxers de configuració. També s'inclou la documentació javadoc.

3.6 Breu descripció dels altres capítols de la memòria

3.6.1 Estudi i avaluació de frameworks de presentació

Aquest és el capítol quart corresponent a la part realitzada a la PAC 2, i està dedicat a l'estudi i avaluació dels frameworks existents. En primer lloc he introduït i descrit el concepte de patró (punt 4.1). Aquest és un concepte imprescindible per continuar amb el següent, marcs de treball (punt 4.2). Entre les descripcions de patró he descrit els que són útils per la comprensió d'aquest projecte. També he tractat, en aquest punt, l'ús de marcs de treball (punt 4.2) i l'arquitectura Java (punt 4.3). A partir d'aquest dos punts podem entendre la tecnologia, la forma de treballar i la metodologia a seguir.

En el següent punt resumeixo les característiques i formes de treballar amb els diferents frameworks (punt 4.4) i finalment en el darrer punt d'aquest apartat trobem una comparativa entre diferents frameworks (punt 4.5).

3.6.2 Implementació d'un framework

Aquest és el capítol cinquè corresponent a la part realitzada en la PAC 3, i està dedicat a la implementació d'un framework i. Després d'estudiar la tecnologia i el paradigma que suposa treballar amb marcs de treball i Java, he desenvolupat un framework nou i és en aquest capítol on explico el seu disseny i funcionament. Finalment, aquest capítol queda complementat amb l'annex Manual del desenvolupador on realitzo una guia pel seu ús.

Apartat
4

ESTUDI I AVALUACIÓ DE FRAMEWORKS DE PRESENTACIÓ

4.1 Patrons

4.1.1 Introducció

Un patró (Miquel & Martos, 2005) és una plantilla que fem servir per a solucionar un problema durant el desenvolupament del programari per tal d'aconseguir que el sistema desenvolupat tingui les mateixes qualitats que tenen altres sistemes on anteriorment s'ha aplicat la mateixa solució amb èxit.

Segons els seu propòsit trobem tres tipus:

- Creació (Factory Method o mètode de fabricació): Fàbrica, Constructor, Propotipus i singleton.
- Estructural (Adapter o adaptador): Adaptador, pont, composició, decorador, façana i flyweight.
- De conducte (Intèrpret o plantilla): Cadena de responsabilitat, comanda (ordre), iterador, intermig, observador, estat, estratègia, visitant i memòria.

4.1.2 Objectius patrons

Els seus avantatges són nombrosos, aquests són fruit de l'experiència d'haver solucionat un problema moltes vegades i analitzar les conseqüències, per tant podem deduir que pretenen:

- Reutilització de solucions: aprofitant les experiències prèvies d'altres persones que han dedicat esforç a entendre els contextos, les solucions i conseqüències del problema.
- Benefici del coneixement: coneixement i experiència de les persones que han desenvolupat un solució mitjançant un enfocament metòdic.
- Transmissió de l'experiència: comunicació i transmissió de l'experiència o d'un nou coneixement.
- Vocabulari comú: establir un vocabulari comú per millorar la comunicació.
- Encapsulació del coneixement: encapsulació detallada sobre un tipus de problema i les seves solucions assignant un nom per tal que ens puguem fer referència fàcilment.
- No reintentar res: no cal tornar a analitzar el problema.

De la mateixa forma, no pretenen:

- Imposició: imposar certes alternatives de disseny front a altres.
- Eliminar creativitat: no és la intenció eliminar el procés creatiu del disseny.

4.1.3 Problemàtica patrons

Un problema que ens podem trobar en l'aplicació de patrons és l'ús incorrecte. El fet de disposar d'un catàleg de solucions a problemes generals, no evita que haguem de raonar sobre el nostre desenvolupament per entendre els problemes que se'ns plantegen durant el procés. Així, per exemple, durant el disseny, hem de saber quin és el nostre objectiu per tal de valorar si l'aplicació d'una solució concreta és més bona que una altra. És força habitual que, quan algú comença a treballar amb patrons, assumeixi que el fet d'aplicar un patró proporciona, automàticament, la millor solució possible, la qual cosa no sempre és certa. Un cop triat un patró que sembli adequat (és a dir, que sigui aplicable al context en què ens trobem), hem d'assegurar-nos d'haver entès correctament quin és el problema que es vol solucionar i quines són les conseqüències d'aplicar- lo. Un ús poc acurat del patró ens pot portar a una

solució final inadequada, sigui perquè el patró no resol el problema a què ens enfrontàvem o bé perquè no hem sabut valorar correctament les conseqüències de la seva aplicació. Llavors els inconvenients superen els avantatges.

4.1.4 Patró de disseny

Per aquest projecte em vull centrar en uns tipus de patrons determinats, els patrons de disseny. Els patrons de disseny són aquells que s'apliquen per resoldre problemes concrets de dissenys que no afecten el conjunt de l'arquitectura del sistema. Un patró de disseny (Gamma, Helm, Johnson, & Vlissides, 1995) dóna nom, motiva i explica de manera sistemàtica un disseny general, que permet solucionar un problema recurrent de disseny en sistemes orientats a objectes. El patró descriu el problema, la solució, quan s'ha d'aplicar la solució així com les seves conseqüències. També dóna algunes pistes sobre com implementar-lo i exemples. La solució és una distribució d'objectes i classes que solucionen el problema. La solució s'ha de personalitzar i implementar per tal de solucionar el problema en un context determinat.

Una arquitectura orientada a objectes (Grady Booch) ben estructurada està plena de patrons. La qualitat d'un sistema orientat a objectes es medeix per l'atenció que els dissenyadors han prestat a les col·laboracions entre els seus objectes. Els patrons condueixen a arquitectures més petites, més simples i més comprensibles.

Els patrons de disseny són descripcions de classes que tenen instàncies que col·laboren entre sí. Cada patró és adequat per ser adaptat a un cert tipus de problema. Per descobrir un cas, hem d'especificar:

- Nom.
- Propòsit o finalitat.
- Sinònim (conèixer totes les nomenclatures).
- Problema al que es pot aplicar.
- Estructura o diagrama de classe.

- Participants o responsabilitat de cada classe.
- Col·laboradors o diagrama d'interaccions.
- Implementació (amb tota la documentació).
- Altres patrons relacionats.

4.1.5 El patró observador

El patró observador (de vegades conegut com a publica/subscriu) és un patró de disseny utilitzat en programació d'ordinadors per a observar l'estat d'un objecte en un programa. Està relacionat amb el principi d'invocació implícita.

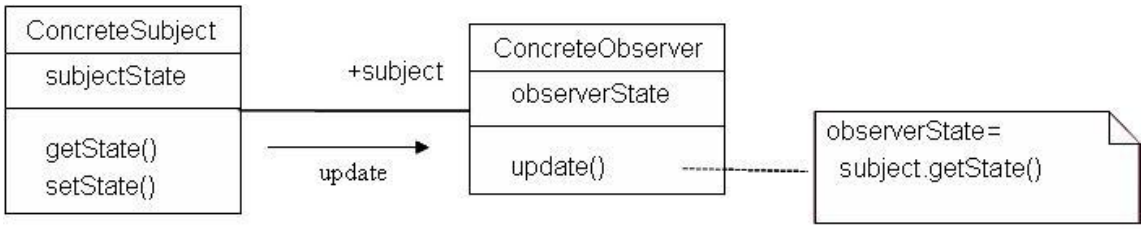


Figura 2: Diagrama d'actualització de classe en el patró Observer

Aquest patró es fa servir principalment per a implementar sistemes de tractament d'esdeveniments distribuïts. En alguns llenguatges de programació, els problemes tractats per aquest patró, són tractats a la sintaxi de tractament d'esdeveniments nativa. Aquesta és una funcionalitat molt interessant en termes de desenvolupament d'aplicacions en temps real.

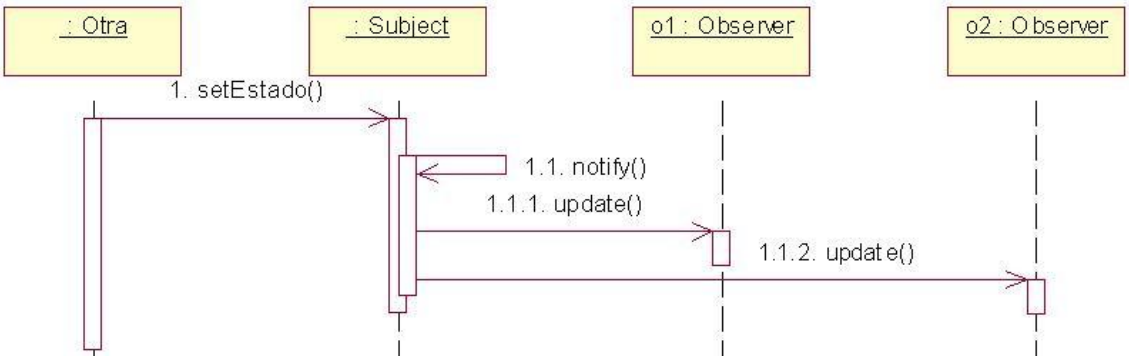


Figura 3: Diagrama de seqüències pel patró Observer

Defineix una dependència de 1_n, de forma que quan l'objecte 1 canviï l'estat, els n objectes són avisats automàticament. Això s'utilitza si tenim un o més

objectes depenent entre sí. L'observador no és un mediador entre els subjectes[1] i els objectes dependents, ja que ell mateix és un objecte depenent. L'observador rep l'ordre d'actualitzar-se per part de l'objecte dominant.

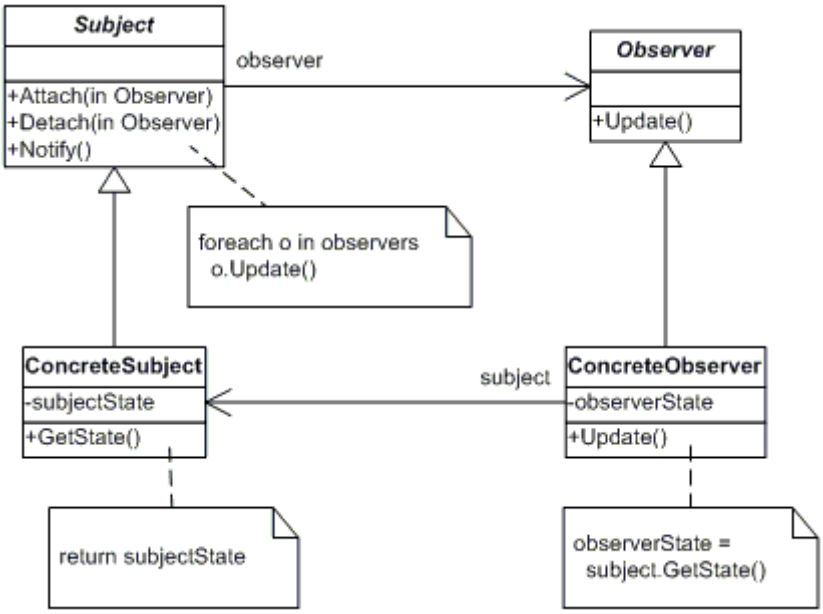


Figura 4: Diagrama UML del Patró Observador

El patró observador també és associat[2] molt sovint amb el paradigma model vista controlador (MVC). A MVC, el patró observador es fa servir per a crear una mena d'unió entre el model i la vista. Típicament, una modificació en el model llença la notificació dels observadors del model, que són, realment, les vistes.

4.1.6 El patró Model Vista Controlador

El patró Model Vista Controlador, anomenat MVC, és usat en el disseny d'aplicacions amb sofisticades interfícies. Si necessitem desacoblar les capes, aquest model separa les dades de l'aplicació, d'interfície de l'usuari i la lògica de control en tres components diferents. D'aquesta manera, les modificacions

[1] Objectes que canvien d'estat.
[2] Veure diagrama de la figura 4: Diagrama MVC on la línea puntejada és l'aplicació del patró Observador

que es duen a terme durant el disseny impacten de la menor forma possible.Els elements que trobem són els següents:

- Model:
  - Accedeix a la capa d'emmagatzemament de dades. Per ser correcte, aquest hauria de ser independent al sistema d'emmagatzematge.
  - Definir les regles de negoci: funcionalitat, flux de pantalles i lògica.
  - Registre de vistes i controladors.
  - Notificació a les vistes dels canvis produïts[3].
- Vista:
  - Rebre les dades del model i mostrar-les al usuari.
  - Tenir un registre del seu controlador.
  - Si el model notifica els canvis (patró observador) informa al controlador.
- Controlador:
  - Rebre dades del model i retornar a la vista.
  - Regles de gestió dels successos.

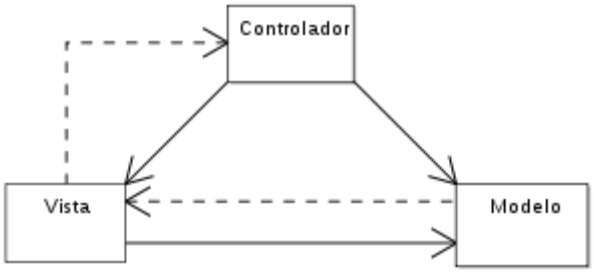


Figura 5: Diagrama MVC on la línea puntejada és l'aplicació del patró Observador

[3] Si existeix el patró observador.

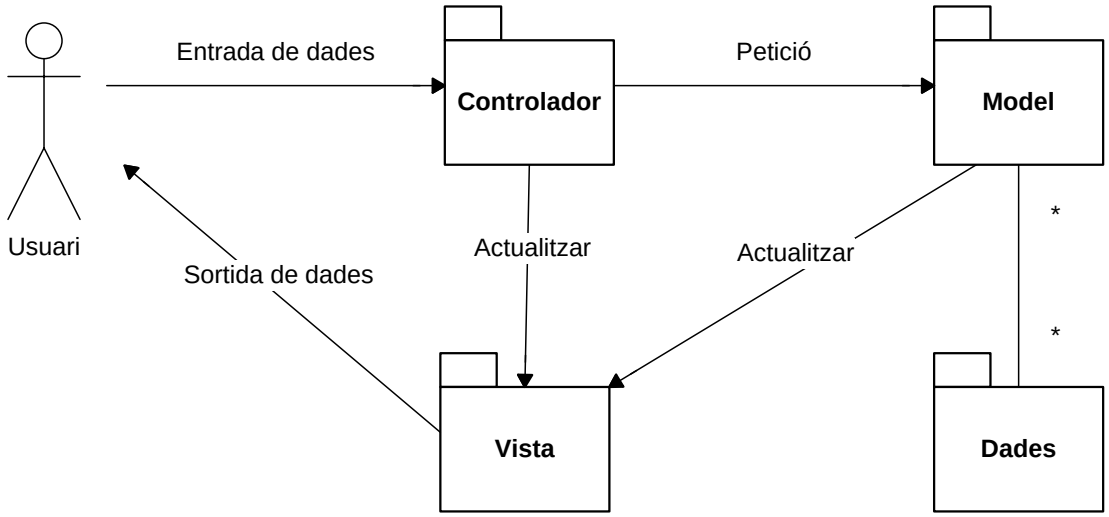


Figura 6: Cas d'ús a MVC

El funcionament correspon a la següent seqüència:

1. L'usuari introdueix el succés.
2. El controlador rep el succés i el tradueix a una petició al Model (o directament la vista).
3. El model (si cal) crida a la vista per la seva actualització.
4. Per complir amb l'actualització, la vista pot sol·licitar dades al model.
5. El controlador rep el control.

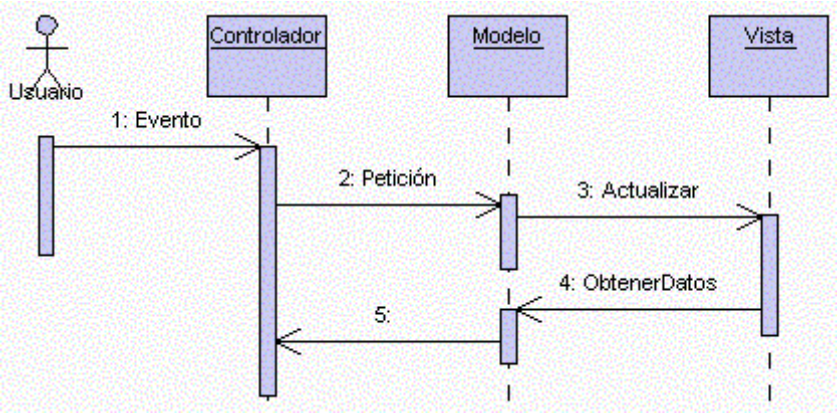


Figura 7: Seqüència del patró MVC.

Existeixen molts frameworks que utilitzen el patró MVC, dels quals, segons el llenguatge amb què estan desenvolupats podem fer la següent classificació:

- **Ruby**: Ruby on Rails, Merb, Ramaze
- **Java**: Grails, Framework Dinàmica, Struts, Beehive, Spring, Tapestry, Aurora, JavaServerFaces.
- **Perl**: Catayst, Gantry Framework, Jifty, Maypole, OpenInteract2, PageKit, Cyclone 3, Solstice, CGI::Builder.
- **PHP**: Self Framework (php5, MVC, ORM, Templates, I18N, Multiples DB), Tlalokes, SiaMVC, Agavi, Zend Framework, CakePHP, KumbiaPHP, Symfony, QCodo, CodeIgniter, Kohana, PHP4ECore, PRADO, FlavorPHP, Yupp PHP Framework, Yii PHP Framework.
- **Python**: Zope3, Turbogears, Web2py, Pylons, Django.
- **.NET**: MonoRail, Spring .NET, Maverick .NET, ASP.NET MVC, User Interface Process (UIP) Application Block.
- **AS3**: Cairngorm, CycleFramework.

4.2 Marcs de treball

4.2.1 Introducció

Anomenem framework a un marc de treball. Un framework (Foote, 1988) és un conjunt de classes que incorpora un disseny abstracte de solucions per a una família de problemes relacionats. Segons això, un framework està composat per un conjunt de classes i de fluxos de control i així, proveeix una estructura precisa per definir noves aplicacions que resolguin un problema donat dins d'un domini donat (família de problemes relacionats). Un framework orientat a objecte (Roa, Gutiérrez, & Stegmayer, 2008) és una arquitectura de software reutilitzable, que proveeix tant de codi com de disseny. Aquest representa una aplicació parcial del programari, tant en disseny com en codi, per a solucionar problemes d'un domini donat. Les tècniques de desenvolupament de programari orientat a objecte ha demostrat ser una eina poderosa en l'obtenció de components de programari reutilitzables, com són els patrons de disseny.

Aquests són l'exemple idoni de l'aplicació de disseny orientat a objectes, per a obtenir components de programari flexibles i reutilitzables.

4.2.2 Ús de frameworks

Existeixen dos tipus de frameworks, els de caixa blanca i els de caixa negre. Els frameworks de caixa negre, anomenats black-box, són els mecanismes de control ocults a l'usuari, i la forma d'utilització és a través de la composició d'objectes. Els de caixa blanca, anomenats white-box, s'utilitzen fent una extensió de les classes aparegudes en el framework. Aquestes classes esteses són els punts de flexibilització o hot-spots.

El seu ús es molt diferent al de les biblioteques de classes, encara que aquest les inclogui. Un framework té implementats fils de control i les crides a codi de les aplicacions de les quals són realitzades quan es necessiten. Així doncs, el framework proveeix un context pels components d'una biblioteca de classe que serà reutilitzada.

L'objectiu d'un framework és produir una solució parcial a un conjunt de problemes semblants. Tenint en compte aquest objectiu, el seu desenvolupament no és una tasca arbitrària. El framework resultant ha de proveir uns components de software que puguin ser adaptats i reutilitzats en la solució global d'una solució per un problema específic.

4.3 Arquitectura JEE

4.3.1 Introducció

La plataforma JEE implica una forma d'implementar i desplegar aplicacions empresarials. La plataforma s'ha obert a una gran varietat de fabricants de programari per aconseguir satisfer una amplia varietat de requisits empresarials. L'arquitectura JEE implica un model d'aplicacions distribuïdes en diverses capes o nivells (tier). La capa client admet diversos tipus de clients (com poden ser HTML, Applet, aplicacions Java,...), la capa intermèdia (middle

tier) conté subcapes (el contenidor web i el contenidor EJB) i la tercera capa està dintre d'aquesta visió sintètica, que és la d'aplicacions "backend" com poden ser ERP, EIS, bases de dades,... Un concepte clau de l'arquitectura és la del contenidor, que dit d'altre forma genèrica, no és més que un entorn d'execució estandarditzat que ofereix uns serveis per medi de components. Els components externs al contenidor tenen una forma estàndard d'accedir als serveis de dit contenidor, amb independència del fabricant.

JEE té tres àrees diferents:

- Java Micro Edition (JME): orientat a dispositius mòbils com PDA's i telèfons mòbils amb capacitat suficient com per executar Java. Té limitacions de memòria, capacitat, rendiment i ofereix solucions específiques per les característiques d'aquests dispositius.
- Java Estandar Edition (JSE) és la més coneguda i només necessita una màquina que sigui capaç d'executar la màquina virtual de Java.
- Java Enterprise Edition (JEE) és la destinada al desenvolupament i execució d'aplicacions empresarials que necessiten característiques especifiques per aquest entorn com seguretat, transaccionalitat, alta disponibilitat o la robustesa.

Anteriorment, les nomenclatures eren J2EE i ara s'ha eliminat per passar a ser JEE (així com la resta J2ME passa a JME i JJ2SE passa a JSE) ja que corresponia a 1.2 i ara ens referim sense .2 deixant la versió com al nom final.

4.3.2 Programació eficient

J2EE és actualment una plataforma de desenvolupament empresarial que defineix un estàndard pel desenvolupament d'aplicacions empresarials multicapa. Simplifica el desenvolupament d'aquestes aplicacions a partir de components modulars estandarditzats, proveint d'un conjunt molt complet de serveis a aquest components i gestionant automàticament moltes funcionalitats o característiques que requereix qualsevol aplicació empresarial sense

necessitat d'una programació complexa. Per aconseguir productivitat és important que els equips de desenvolupament tinguin una forma estàndard de construir múltiples aplicacions en diverses capes (client, servidor web,...). En cada capa necessitem diverses eines, per exemple, en la capa client hi han applets, aplicacions Java,... A la capa web hi han servlets, pàgines JSP, ... Amb JEE aconseguim una tecnologia estàndard, un model únic d'aplicacions que inclou diverses eines; en contraposició al desenvolupament tradicional amb HTML, Javascript, CGI, servidor web, ... que implica nombrosos models per la creació de continguts dinàmics, amb els seus inconvenients per la integració.

4.3.3 Extensibilitat

En un context de creixement de nombrosos usuaris es precisa la gestió de recursos, com connexions a bases de dades, transaccions i balanceig de càrrega. A més, els equips de desenvolupament han d'aplicar un estàndard que els proporcioni abstraure's de la implementació del servidor, amb aplicacions que puguin executar-se en múltiples servidors, des d'un simple servidor fins una arquitectura d'alta disponibilitat i balanceig de càrrega entre diverses màquines.

4.3.4 Integració

Els equips d'enginyeria precisen estàndards que afavoreixin la integració entre diverses capes de programari.

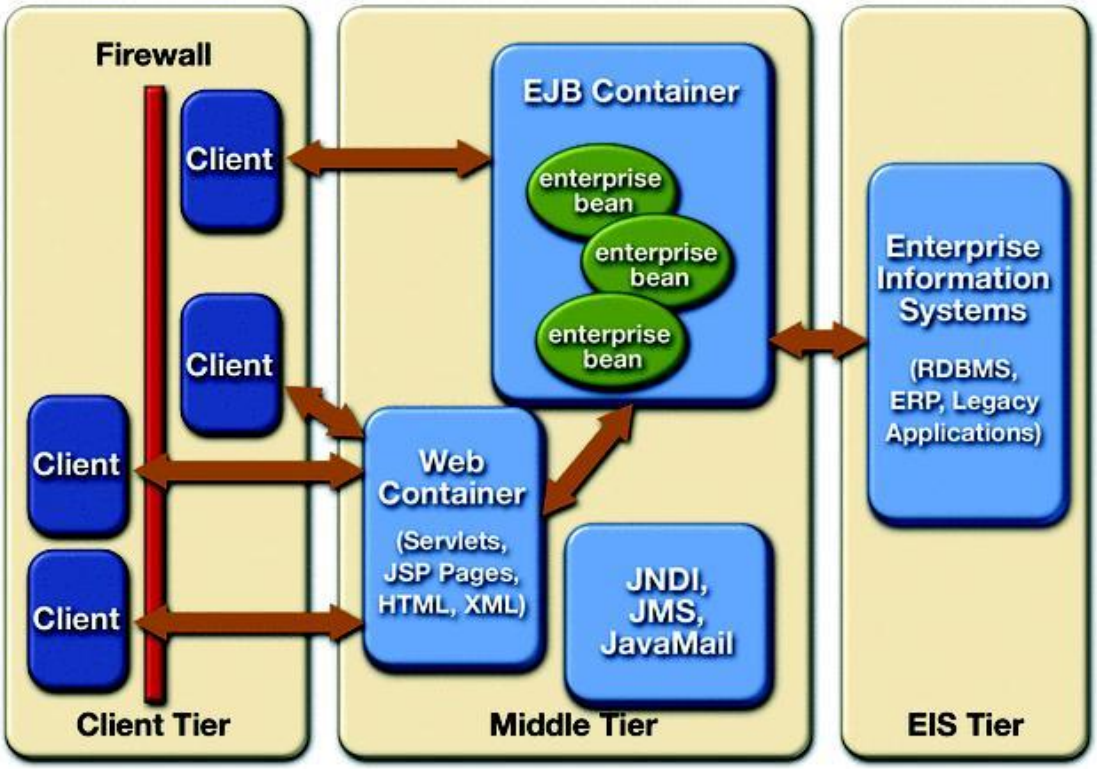


Figura 8: Estructura de l'arquitectura JEE

4.3.5 Contenidors

El contenidor Web és el Servlet/JSP que controla l'execució dels servlet i les pàgines JSP. Aquest components s'executen sobre un servidor Enterprise Edition.

El contenidor Enterprise JavaBeans gestiona l'execució dels EJB que requereix un servidor JEE.

Els contenidors inclouen descriptors de desplegament (deployment descriptors), que són fitxers XML. Aquest fitxers serveixen per configurar l'entorn d'execució: camins d'accés a aplicacions, control de transaccions, paràmetres d'inicialització,...

4.3.6 API

Una API és la interfície de programació d'aplicacions que és un conjunt de funcions i procediments que ofereix una biblioteca determinada, per ser utilitzada per altre software com a capa d'abstracció.

El APIs inclosos a JEE són:

- JDBC és l'API per l'accés a GBDR des de Java.
- Java Transaction API (JTA) és l'API per el control de transaccions mitjançant sistemes heterogenis.
- Java Naming and Directory Interface (JNDI) és l'API per l'accés a serveis de noms i directoris.
- Java Message Service (JMS) és l'API per l'enviament i recepció de missatges mitjançant sistemes de missatgeria empresarial com IBM MQ Series.
- JavaMail com a API per enviament/recepció de correu electrònic.
- Java IDL com a API per crides a serveis CORBA.

4.3.7 Servidor d'aplicacions JEE

Un servidor d'aplicacions està format per:

- Servidor HTTP: servidor web, per exemple Apache.
- Contenidor d'aplicacions o contenidor Servlet/JSP, per exemple Tomcat (Tomcat també és un Servidor HTTP).
- Contenidor Enterprise Java Beans, per exemple JBoss (JBoss també és un Servidor HTTP i Contenidor d'aplicacions).

4.3.8 Servlet, HTTP i el paradigma petició-resposta

La capa web d'una aplicació J2EE està desenvolupada sobre un protocol HTTP seguint el paradigma de petició-resposta. Els components web reben peticions HTTP del servidor web, les processen i generen una resposta a la petició.

Si les peticions dels clients rebudes pel HTTP van dirigides a un component web, aquest se les passa al contenidor web i aquest s'encarregarà de processar-les i retornar la resposta al client. Si no és el cas, enviarà la resposta al client sense passar pel contenidor web.

Els servlet són objectes de Java que estenen la funcionalitat d'un servidor web, rep peticions HTTP i genera contingut dinàmic com a resposta a les peticions.

El servlet resideix dins d'un contenidor web, carrega i executa dinàmicament com a resposta a les peticions que els clients fan a una URL determinada. Basen el seu funcionament en el paradigma de petició-resposta sota el protocol HTTP.

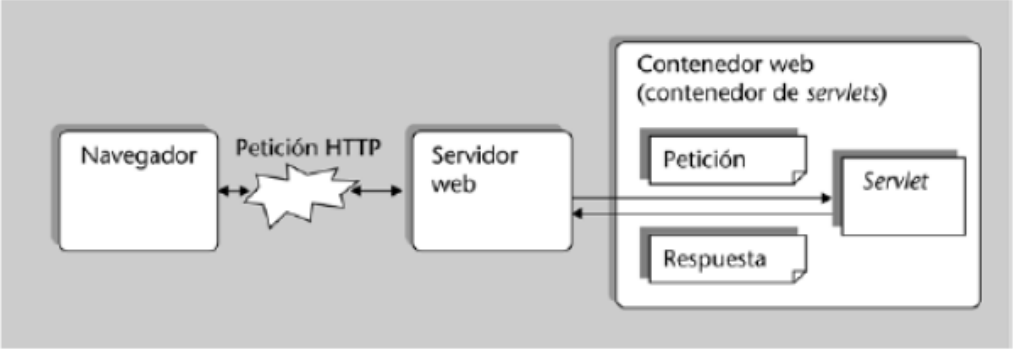


Figura 9: Paradigma petició-resposta amb un servlet

La seqüència és la següent:

1. El client fa una petició HTTP a una URL a la que té associat un servlet.
2. El servidor web rep aquesta petició, s'adona que va dirigit a un servlet i passa la petició al contenidor de servlet.
3. El contenidor de servlets crea un objecte Java que representa la petició i ho passa al servlet cridant-lo al mètode service() d'aquest.
4. El servlet processa la petició:
5. Analitza els paràmetres de la petició.
6. Executa les tasques que requereixen per completar la petició.
7. Crea una resposta (que s'acaba traduint en una pàgina HTML).
8. El servlet retorna la resposta al contenidor de servlet.
9. El contenidor de servlets envia la resposta al servidor web.
10. El servidor web l'envia al client.

El cicle de vida dels servlets és controlat pel contenidor[4] on s'ha desplegat. Quan el contenidor rep una petició que està mapejada amb un servlet, es produeix els següents passos:

- El contenidor crea els objectes que treballaran amb la petició i la resposta.
- Si no hi ha cap instància del component a memòria, el contenidor crea una instància i la inicialitza amb el mètode init() i després al mètode service().
- Si ja havia una instància del component, crida al mètode service() directament.
- El codi del mètode service() s'executa.

Les peticions fetes mitjançant HTTP més utilitzades son les POST i GET. El servlet, a més del mètode service() té els mètodes doGet() i doPost() per servir directament les peticions efectuades d'aquesta manera.

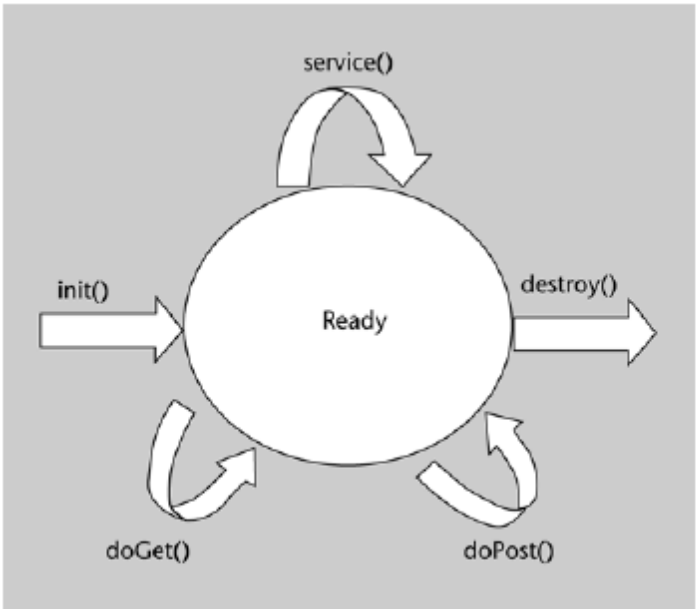


Figura 10: Cicle de vida d'un servlet

[4] La gestió del cicle de vida dels components és un servei que proporcionaven els contenidors als components que tenen desplegats.

4.4 Marcs de treballs de presentació per a aplicacions J2EE

4.4.1 JavaServer Faces JSF

Introducció

JavaServer Faces és un framework orientat a la interfície gràfica de l'usuari (GUI), facilitant el desenvolupament d'aquest i realitzant una separació entre comportament i presentació. A més, proporciona un servlet com a controlador, implementant així els principis del patró de disseny MVC. La característica principal és que és un model basat en components pel desenvolupament d'aplicacions web similar al GUI standalone com Swing que és un estàndard pel desenvolupament d'interfícies gràfiques.

JavaServer Faces (JSF) és un marc de treball per aplicacions web basades en Java que simplifica el desenvolupament d' interfícies d'usuari per a aplicacions Java EE. JSF utilitza JavaServer Pages (JSP) com a tecnologia per fer el desplegament de les pàgines, però també pot utilitzar altres tecnologies, com per exemple XUL. JSF inclou:

- Un conjunt d'APIs per representar components d'una interfície d'usuari i administrar el seu estat, controlar esdeveniments i validació d'entrada, definir un esquema de navegació de les pàgines i donar suport per a internacionalització i accessibilitat.
- Un conjunt per defecte de components per a la interfície d'usuari.
- Dues llibreries d'etiquetes personalitzades per a JavaServer Pages (JSP) que permeten representar una interfície JavaServer Faces dins d'una pàgina JSP.
- Un model d'esdeveniments en el costat del servidor.
- Administració d'estats.
- Managed Beans (JavaBeans creats amb injecció de dependència).

Aquests objectius de disseny representen el focus de desenvolupament de JSF:

- Definir un conjunt simple de classes base de Java per a components de la interfície d'usuari, estat dels components, i esdeveniments d'entrada. Aquestes classes tractaran aspectes del cicle de vida de la interfície d'usuari, controlant l'estat d'un component per al curs de la vida de la seva pàgina.
- Proporcionar un conjunt de components per la interfície d'usuari, incloent-hi els elements estàndards d'HTML per representar un formulari. Aquests components s'obtindran del conjunt bàsic de classes base que es poden utilitzar per definir components nous.
- Proporcionar un model de JavaBeans per a enviar esdeveniments des dels controls de la interfície d'usuari del costat del client a l'aplicació del costat del servidor.
- Definir unes APIs per a la validació d'entrada, incloent-hi suport per a la validació del costat del client.
- Especificar un model per a internacionalització i localització de la interfície d'usuari.
- Automatitzar la generació de sortides apropiades per l'objectiu del client, tenint en compte totes les dades de configuració disponibles del client, com la versió del navegador,...

Funcionament

Necessitem incorporar llibreries, aquestes són jsf-api.jar, jsf-impl.jar, commons-beanutils.jar, commons-collections.jar, commons-digester.jar, commons-logging.jar, standard.jar i jstl.jar. Després hem d'incorporar web.xml i faces-config.xml en WEB-INF. També hem de carregar els TLD[5]: html_basic.tld, jsf_core.tld i facesconfig_1_1.dtd. Per utilitzar-ho a la pàgina jsp hem de incorporar:

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

[5] Aquests són les definicions de les etiquetes JSF

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

Ara podem utilitzar els components amb h i validacions i events amb la f. Podem associar un component amb un Bean, així doncs a l'hora d'usar el bean podem utilitzar h:nomEtiqueta value="#{NomClasse.atribut}" i aconseguirem posar al camp de nomEtiqueta el valor de setatribut de la classe NomClasse. Per tant hem de definir tots els setters i getters dels atributs definits a les clases.

El fitxer faces-config és un fitxer XML que conté els fluxos. Aquí hem de definir quina vista ha d'anar segons el resultat. Podem indicar que si estem a la pàgina1.jsp anem a la pàgina2.jsp si l'acció del botó donat és "success", per exemple:

```
<navigation-rule>
    <from-view-id>/pagina1.jsp</from-view-id>
    <navigation-case>
        <description>
               Anem de la pàgina1 a la pagina2 amb l'accio succes
        </description>
      <from-outcome>success</from-outcome>
      <to-view-id>/pagina2.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
```

Cicle de vida

El controlador de JSF intercepta les peticions a les pàgines JSF. A més d'interceptar aquestes crides, el servlet prepara el context JSF abans d'enrutar a les pàgines corresponents. La invocació a la pàgina JSF ha de recórrer necessàriament el servlet, que ha de ser interceptat pel servlet controlador. La seqüència es pot apreciar a la figura 10.

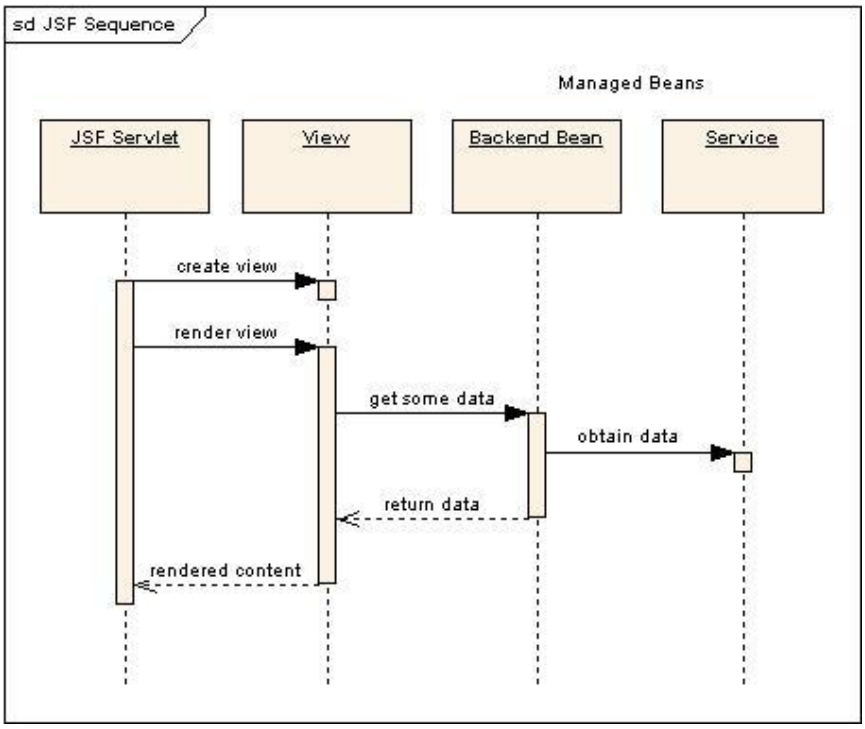


Figura 11: Seqüència JSF

Així d'aquesta forma podem distingir sis fases. Aquestes fases queden descrites a la figura 11 tenint en compte que els "Process Events" són l'execució d'events externs cridats durant el cicle.

- **Restore View**: és la primera etapa que es realitza i s'inicia al realitzar una petició. El seu objectiu és la creació d'un arbre amb tots els components de la pàgina en qüestió.
- **Apply Request Values**: cadascun dels components de l'arbre creat en la fase anterior obté el valor que li correspon a la petició realitzada i el magatzema.
- **Process Validations**: després d'emmagatzemar els valors de cada component, aquest són validats segons les regles que s'han declarat.
- **Update Model Values**: els valors locals dels components són utilitzats per actualitzar els beans que estan lligats als components. Aquesta fase només succeeix si les validades abans han estat passades correctament.

- **Invoke Application**: s'executa l'acció corresponent a l'event inicial que va donar començament al procés.
- **Render Response**: la resposta es renderitza i torna al client.

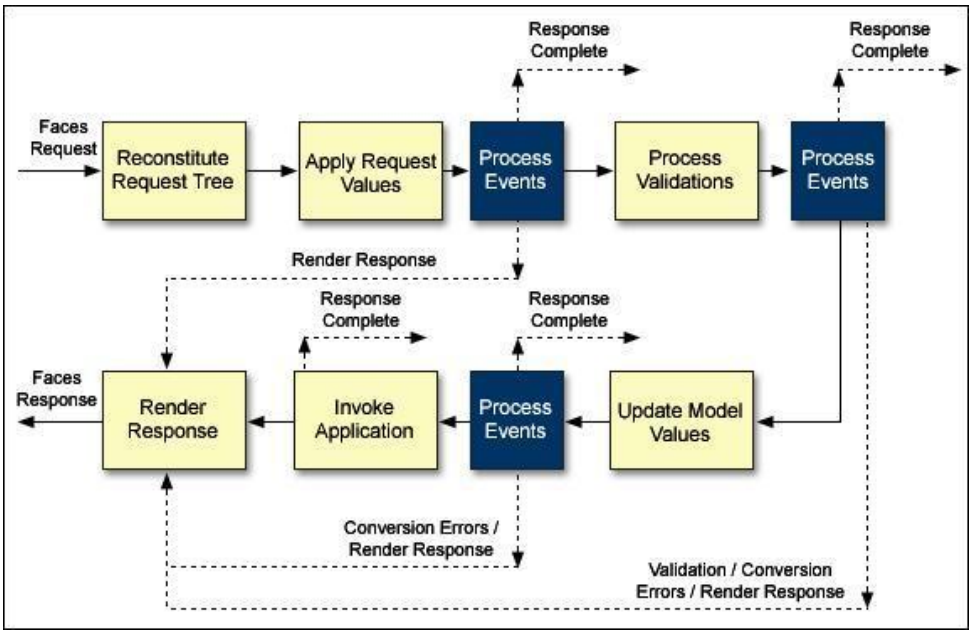


Figura 12: Cicle de vida JSF

4.4.2 Spring MVC

Introducció

Spring és un framework que es caracteritza principalment per la seva modularitat. No és un framework MVC però si té un mòdul MVC. La modularitat es pot dur a terme sense comprometre's a la resta. Alguns altres mòduls són:

- El Core Container (Inversion of Control, IoC) o Contenidor de Inversió de Control. És el responsable de la creació i configuració dels objectes.
- Aspect-Oriented Programming Framework, treballa amb solucions que són utilitzades en nombrosos llocs d'una aplicació, el que es coneix com a temes transversals (cross-cutting concerns).
- Data Access Framework, que facilita la feina d'utilització de l'API amb JDBC, Hibernate, etc.
- Transaction Management Framework.

- Remote Access framework. Facilita la existència d'objectes en el servidor que és exportat pel seu ús com a serveis remots.
- Spring Web Flow.
- Spring Web Service.

A més de la modularitat, Spring també conté altres característiques:

- Ús de patrons Factory, Abstract Factory, Builder, Decorator, Service Locator, entre altres.
- Codi obert.
- Objecte de negocis dins de l'arquitectura de capes.
- Funció de connector entre APIs (com JDBC i JNDI) i frameworks (com Struts i iBatis).

Arquitectura i classes

Per l'ús del patró MVC tenim:

- Capa de vista: jsp, html, css, etiquetes personalitzables,...
- Capa de model: composada per subcapes de serveis, persistència i domini.

La capa de control té les següents classes:

- **DispatcherServlet**: és el controlador frontal, que rep i gestiona totes les peticions (request). Resulta ocult al desenvolupador i és instanciat per Spring.
- **Interface HandlerMapping**: analitza cada petició i determina el controlador que gestiona. Podem comptar amb diversos controladors, en funció de les diverses estratègies de mapejar. Per defecte utilitzem BeanNameUrlHandleMapping.
- **Controladors**: s'encarreguen de les peticions de cada pàgina. Cada controlador rep les peticions de la seva pàgina corresponent, delegant en el domini i recollint els resultats. El que fa és tornar un model a la

vista que ha seleccionat mitjançant el controlador frontal. El que retorna cada controlador és un ModelAndView. Aquest es composa de:

- Una referència a la vista de destinació.
- El model, que és un conjunt d'objectes que s'utilitzen per composar (renderitzar) la vista de destinació.

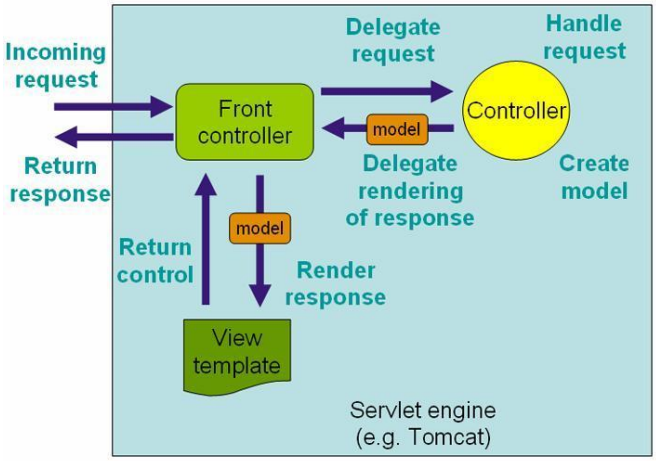


Figura 13: Funcionament del Spring MVC

Funcionament

Al web.xml hem d'incloure:

```
<listener>
        <listener-class>
org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>
<context-param>
        <param-name>
                contextConfigLocation
        </param-name>
        <param-value>
                WEB-INF/xyz.xml, WEB-INF/abc.xml
        </param-value>
</context-param>
<servlet>
        <servlet-name> spring21 </servlet-name>
        <servlet-class>
                org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
        <servlet-name>spring21</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Al web.xml que trobem a totes les aplicacions web java, definim el Listener amb l'event contextInicialitzed per carregar el context de l'aplicació. Ara hem de triar el fitxer de definició de context de l'aplicació amb el paràmetre contextConfigLocation (en aquest cas els xyz.xml i abc.xml dins de la ruta relativa /WEB-INF com a exemple). I finalment, indicar quin és el servlet (en aquest cas spring21). El url-pattern indica quines peticions accepta, en aquest exemple les extensions .do.

Aquest seria la configuració del servlet que abans hem anomenat spring21. En aquest cas hi ha dos controladors, el primer que fa les insercions, actualitzacions (indexControler) i consultes, i l'altre per l'esborrat (borrarControler). El viewResolver indica on són les vistes que són invocades pel controlador frontal (en el xml anterior són .jsp del directori relatiu /WEB-INF/jsp). El Bean urlMapping indica el mapeig de peticions i controladors. En aquest xml tenen .do i el controlador definit.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
        <!-- Controlador de index (insercions, actualitzacions) -->
        <bean id="indexController" class="com.controlador.IndexController">
                <property name="servicioCliente" ref="servicioCliente" />
        </bean>
        <!-- Controlador de index (esborrar) -->
        <bean id="borrarController" class="com.controlador.BorrarController">
                <property name="servicioCliente" ref="servicioCliente" />
        </bean>

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

            <bean id="urlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
                        <value>
                /inicio.do=indexController
                /borrar.do=borrarController
                        </value>
                </property>
        </bean>
</beans>
```

Tenint això, seguirem la seqüència:

1. El servidor d'aplicacions activa l'event ContextInitilitzed.
2. Aquest event invoca al ContextLoaderListener (definit en el web.xml) i crea el context de l'aplicació applicationContext.xml.
3. Inicialització del servlet frontal (Dispatcher Servlet) i creació del seu context (spring21-servlet.xml).
4. El controlador central cerca i inicialitza els components. Si no els troba, utilitza els que estan per defecte.

4.4.3 Struts

Introducció

Struts estava emmarcat dins del projecte Jakarta Struts. Ara ja no és així i actualment es coneix com Apache Struts. Va ser creat per Craig R. McClanahn al 2002 dins del Apache Software Foundation's (ASF) per convertir-se en un projecte independent. Aquests van desenvolupar el framework Struts creant una eina de treball per a desenvolupadors que vulguin crear una aplicació web utilitzant servlets i JavaServerPages JSP, encara que no és obligatori. És un dels frameworks més utilitzats i és de programari lliure.

En la seva versió 1.3 conté un conjunt de 41 classes i 17 classes addicionals d'utilitat.

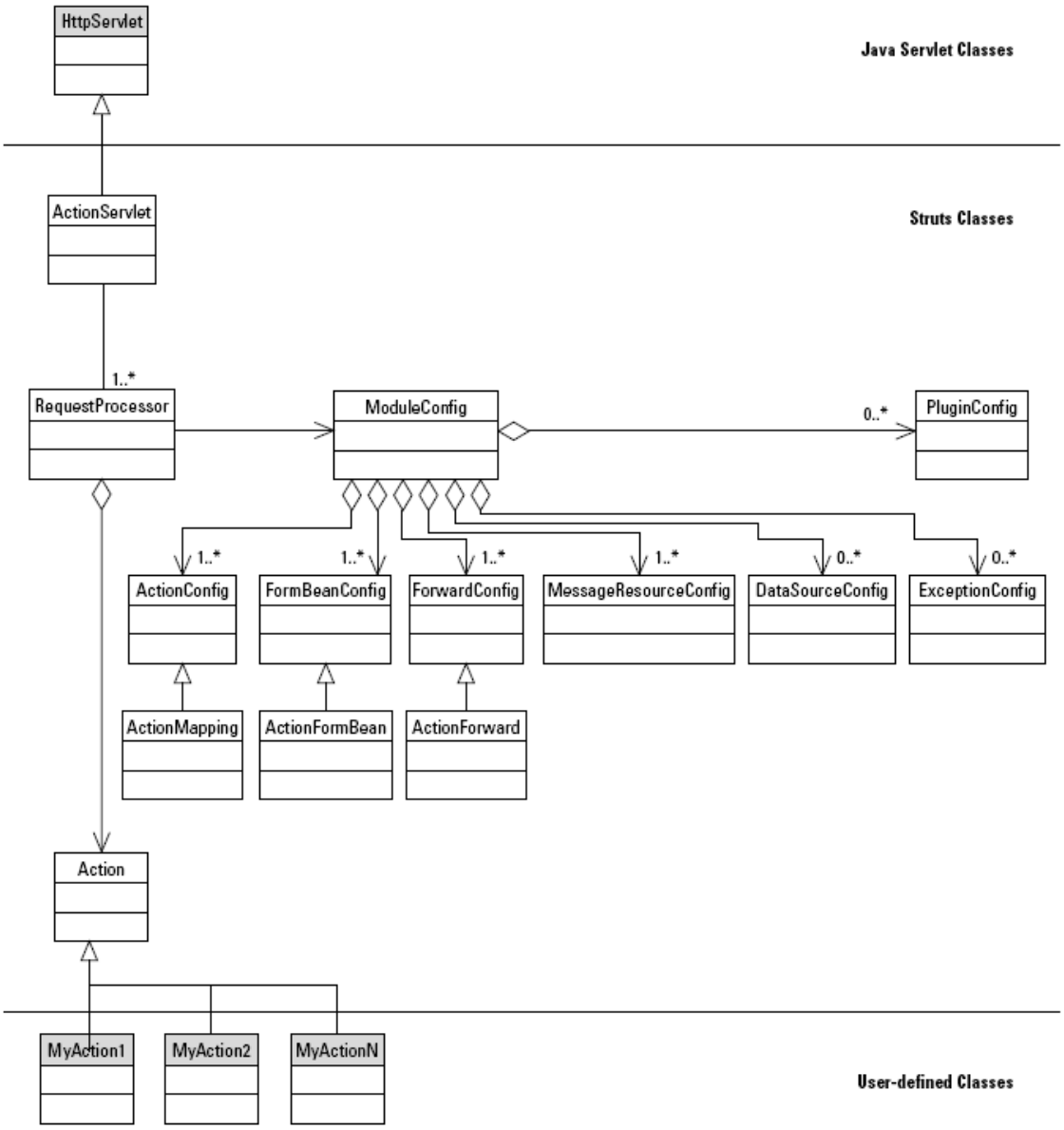


Figura 14: Diagrama de classes del framework Struts

Les característiques principals són:

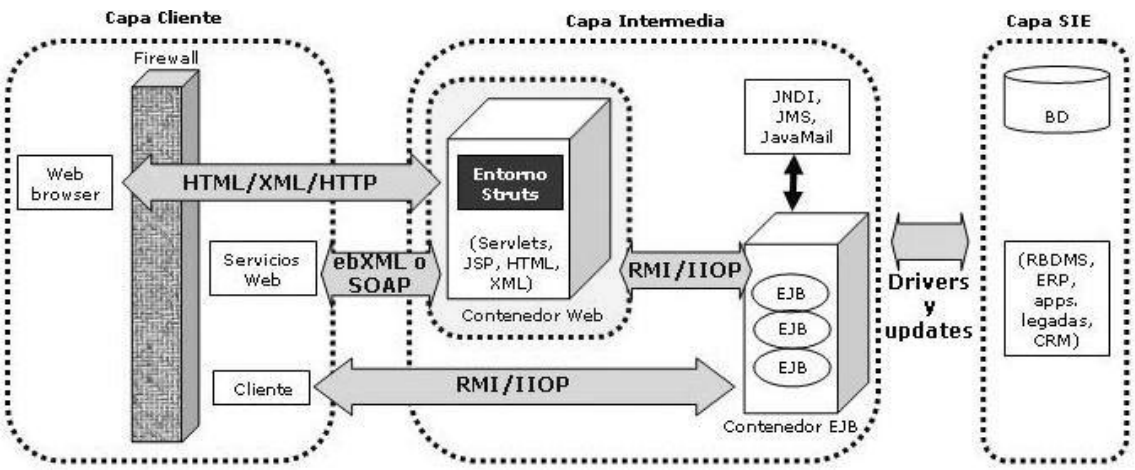
- És un framework que implementa el patró MVC.
- El controlador ens ve donat i es pot modificar.
- El workflow de l'aplicació es configura mitjançant un fitxer XML.
- Les accions s'executen sobre un model d'objectes de negoci que s'implementen basant-se en classes predefinides pel framework.

- Té una sèrie de Tags predefinits per Struts que eviten l'ús d'Scriplets.
- Separació del workflow de la lògica de negoci.
- Suport de múltiples interfícies d'usuari com html, shtml o wml, multi idiomes, etc.
- Gran número d'extensions estàndard disponibles.
- Corba d'aprenentatge reduïda.
- Ja que sempre es construeix igual, podem aplicar una metodologia a nivell de projecte.

Arquitectura

Les aplicacions Struts resideixen en un contenidor web (dins d'un servidor d'aplicacions) i pot fer ús dels serveis proporcionats dels serveis proporcionats pel contenidor, així com de les peticions HTTP i HTTPS. D'aquesta forma ens oblidem de la lògica de negoci. D'aquesta manera, utilitza diferents àrees per emmagatzemar i compartir recursos que són:

- Petició javax.servlet.http.HttpServletRequest
- Sessió javax.servlet.http.HttpSession
- Aplicació javax.servlet.ServletContext



Pàgina javax.servlet.jsp.PageContext

Figura 15: Estructura d'Struts

Com amb la resta d'aplicacions J2EE, podem dividir-la en nivells. Els nivells que podem utilitzar són:

- **Capa Client**: és l'explorador web. Mitjançant l'explorador web interactuem amb l'aplicació enviant i rebent peticions.
- **Capa Web**: Comunicació entre la capa web i la capa lògica. Tradueix i interpreta les peticions http/https i realitza les invocacions a la capa de negoci. Genera el flux de pantalles basant-se en l'estat de l'aplicació i l'usuari. Struts utilitza un servlet.
- **Capa d'accès de dades**: Proporciona els recursos de l'empresa, normalment la base de dades, encara que pot ser un mainframe, sistema de planificació de recursos, etc.

Classes

- **ActionServlet**, clase extesa de *javax.servlet.http.HttpServlet* que és la responsable de l'empaquetat i direccionalment del tràfic HTTP cap el controlador apropiat dins del marc de treball. No és abstracta i es pot utilitzar en qualsevol aplicació.
- **RequestProcessor**, classe que permet desacoblar el procés de petició (request process) del ActionServlet i així poder modificar com es processa la petició.
- **Action**, classe que independitza la petició del client del model de negoci. És una extensió del component de control i permet executar funcions com l'autorització, logging o la validació de la sessió, abans d'invocar la lògica de negoci. El seu mètode més important és: *public ActionForward execute(ActionMapping mapping, HttpServletRequest request, HttpServletResponse response) throws Exception;*
- **ActionMapping** classe que representa una acció de mapejat que es defineix en el fitxer de configuració d'Struts. Indica al controlador quina instància d'Action s'ha d'invocar en cada petició.
- **ActionForward**, classe que representa el destí al qual el controlador ha d'enviar el control un cop fet una acció.

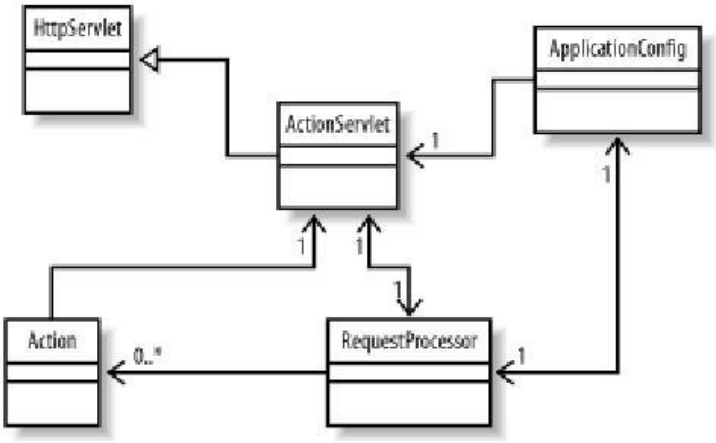


Figura 16: Capa de Control d'Struts

Components

Per a treballar amb Struts podem utilitzar (entre altres opcions):

- **ActionForms**, estàtics i dinàmics (DynaActionForm).
- **Tiles**, per crear i treballar amb plantilles (pàgines Web).
- **Accions** úniques (Action) i de grup (DispatchAction).
- **ActionMappings**, permeten mapejar accions en un fitxer XML.
- **ActionErrors**, per tornar missatges d'error a la validació de formularis (ActionForms).
- **Contenidors**, per desar tota la informació necessària de la sessió.
- **Struts Tags**, llibreria d' etiquetes que ens estalvien codi.
- **Patrons de disseny**: Business Delegate, Business Interface y DTOs (Data Transfer Objects) o Value Objects.
- **Serveis**, per connexió amb la BD mitjançant JDBC.

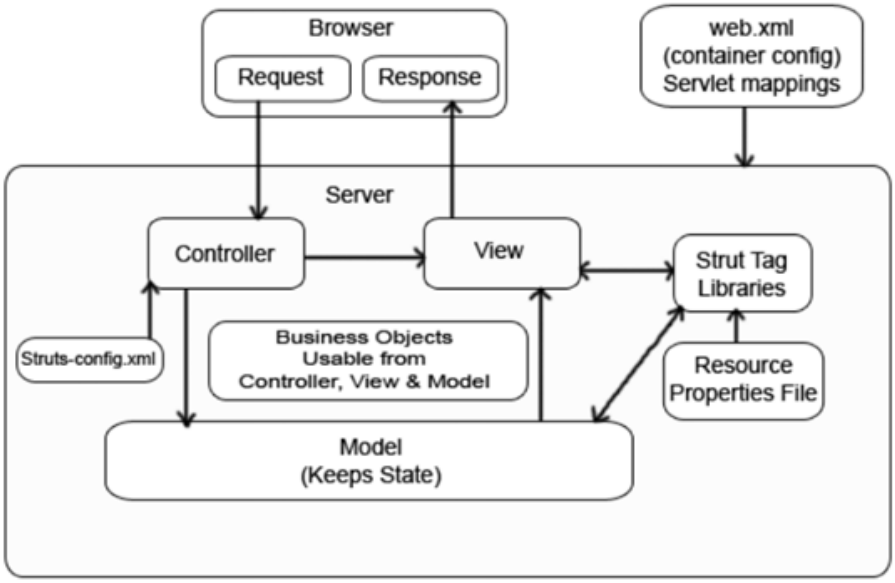


Figura 17: Diagrama de fluxe d'Struts

4.4.4 WebWork 2.2/ Struts 2.0

Introducció

WebWork és un framework opensource dissenyat per ser conceptualment senzill, interoperable i fàcil d'utilitzar, proporcionant un suport robust per construir interfícies d'usuari reutilitzables.

Fusió

Aquest framework intenta reduir al mínim la quantitat de codi necessari per treballar amb el framework, permeten centrar-se en la lògica de negoci i modelat. En el 2006, va aparèixer la versió 2.2 i es va fusionar amb Struts creant Struts Action Framework 2.0. Aquest framework manté la filosofia de WebWork i aprofita la gran comunitat d'usuaris d'Struts i el suport de la fundació Apache.

El 27 de novembre de 2005, Patrick Lightbody de WebWork va anunciar que WebWork estaria combinat amb Apache Struts. El proper llançament va ser el

WebWork 2.2.x que van anomenar WebWork i a partir d'allà, les revisions van ser 2.3.x de Webwork, que es van convertir en les mateixes de 2.0 de Apache Struts.

WebWork utiltza la llicència de programari OpenSymphony que és modificat per fer-ho compatible amb la llicencia de programari Apache.

Arquitectura

En el nucli de WebWork es troba l'API Xwork que és un framework basat en accions i patrons de disenys Front Controller i Comman, centrat a entorn a la interfície Action, que cal implementar en forma de POJO. Aquestes són les classes que implementen les accions que volem que responguin a peticions de la interfície d'usuari i tornin un resultat. El framework s'encarrega de controlar un cicle de les accions i de proporcionar un context en cada petició, a través del qual pot accedir a les propietats de l'aplicació.

Hi ha un fitxer que conté la relació vistes i classes acció. Els camps de les vistes són reomplerts des de i cap als camps de l'acció (definits com a JavaBeans) automàticament. La classe-acció té un mètode de validació que comprova les dades introduïdes i torna els missatges d'error en cas d'una dada introduïda incorrectament. Quan l'acció s'executa, torna un String indicant si l'acció ha tingut èxit i així saber quina vista cal mostrar. A més, les accions es poden encadenar una rere altra, de forma que la primera recull els valors d'entrada, la de la cadena de processament i l'última que decideix quina vista mostrar.

Components

Els components d'un WebWork són els següents:

- **DispatcherFilter**: és el punt d'entrada al framework. A partir d'aquest es llença l'execució del processament de cada request que involucra al

framework. Les seva funció principal és ocupar-se de l'execució dels actions, començar l'execució de la cadena d'interceptors associats al request, netejar el context sobre el qual s'executa una acció[6].

- **Interceptors**: Son classes que segueixen el patró interceptor, s'encarreguen d'interceptar les invocacions a un Action. A més, permeten realitzar operacions abans i després de la invocació d'un Action. També permet evitar que un Action s'executi. Ens ajuda a reutilitzar certes funcionalitats que volem aplicar a un conjunt d'Actions.
- **Actions**: són els encarregats d'executar la lògica necessària per controlar els request. Els Actions no estan obligats a implementar o heretar d'una interfície o classe ja definida. Struts 2 posseeix una interfície Action. Aquesta interfície permet definir el mètode per defecte que s'executarà sobre l'action per que no haguem de definir mitjançant altre medi.
- **Results**: Són els objectes que encapsulen el resultat d'un Action. Els Actions de l'aplicació simplement retornaran un String en els seus mètodes i un Action pot retornar diferents resultats després de la seva execució depenent de les dades.

Funcionament

El framework funciona utilitzant el patró MVC:

- **Model**: s'encarrega de les dades que controlen l'aplicació i les regles de negoci que operen sobre ells i que es tradueixen en Struts2 com a Actions.
- **Vista**: genera la interfície i en Struts2 equival als resultats.
- **Controlador**: comunicació entre vista i model responent als events generats per la vista i invocant els canvis en el model i tornant a la vista la informació del model necessària per a poder generar la resposta

[6] ActionContext

adequada per l'usuari. El controlador en Struts 2 funciona mitjançant el filtre filterdispatcher.

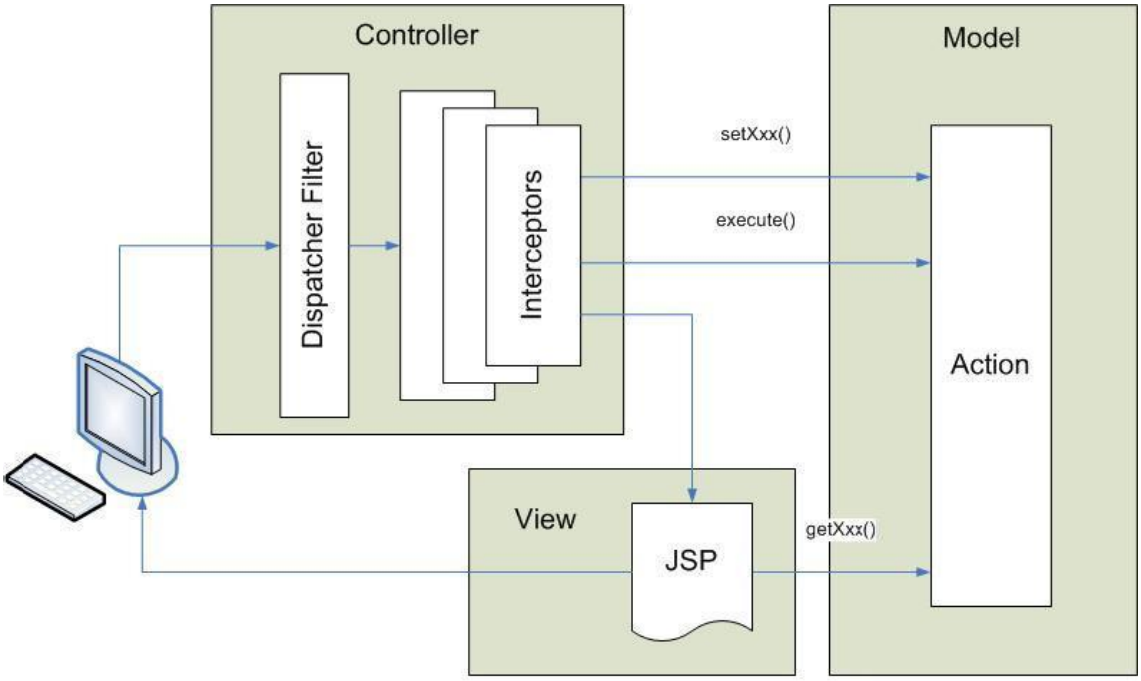


Figura 18: Esquema funcionament d'Struts2

4.5 Comparació

4.5.1 Struts 1 i Struts 2

Sobre les **classes Action** podem dir que tant Struts 1 com Struts 2 han de crear-se a partir d'una altra classe Action interfície. Això sí, Struts 1 sempre programa classes abstractes d'un mateix Action i Struts 2 permet programar classes que ofereixen altres serveis integrats. En Struts 2 també podem utilitzar un POJO que tingui mètode d'execució.

Sobre el **model de fils d'execució** podem dir que Struts 1 utilitza el patró singleton, és a dir, està dissenyat per restringir la creació d'objectes pertanyents a una classe en una única instància. Tots els recursos utilitzats pels actions han de ser thread-safe o sincronized. En Struts 2 els objectes Actions

són instanciats per cada petició, per tant no hi ha problemes de tipus de seguretat de fils. Tenint en compte que els contenidors de servlets generen moltes instàncies i tenir objectes de més, no suposa cap problema de rendiment ni per el garbage collection[7].

Sobre la **dependència de Servlets** podem dir que struts 1 depèn de les API de Servlets ja que HttpServletRequest i HttpServletResponse són passats al mètode execute quan l'Acció és invocada. En canvi Struts 2 no té les Actions acoblades al contenidor. El context dels Servlets a Struts 2 sol ser un Maps i es pot passar la request o el reponse si es necessita però no obligatòriament.

Sobre les **proves**, en Struts1 a les proves dels Actions cal que s'executin mètodes de Servlets o extensions que ho permetin. En canvi a Struts2 utilitzant la injecció de dependències permet fer proves més fàcilment.

Les **expressions** en Struts 1 és en JSTL ja que està integrat. Struts 2 permet, a més de JSTL, un llenguatge més poderós i flexible, el OGNL.

La **vinculació a la vista** en Struts 1 és JSP i en Struts 2 és en ValueStack. Aquest permet als taglibs accedir a valors sense estar acoblats a la vista on l'objecte es renderitza.

---

[7] El llenguatge de programació Java allibera el programador de la responsabilitat d'alliberar memòria. El llenguatge Java proporciona un procés en execució, en l'àmbit de sistema, que rastreja les operacions de reserva de memòria. Durant els cicles morts de la màquina virtual Java, el procés de *garbage collection* verifica quina memòria es pot alliberar i l'allibera.

4.5.2 Comparativa entre Struts i Spring MVC

Sobre **l'estructura** de Spring MVC, hem de dir que es més clara i separada que Struts. En Spring MVC no hem d'**estendre d'un Action**, sinó d'una interfície que pot ser configurada mitjançant un pluggin. En aquest sentit, Spring MVC s'assembla més a Struts 2 ja que les actions a Struts 2 tenen implementades funcions que MVC Spring ho fa mitjançant el pluggin.

Spring MVC permet utilitzar XLST, Velocity o qualsevol altre llenguatge, en canvi a Struts tenim que JSTL[8] per la Struts 1 i JSTL i OGNL per a Struts 2. En el tema de proves, Spring MVC té les mateixes avantatges que Struts 2.

Spring MVC **ofereix** a més de la capa de presentació, un ventall de solucions a partir del framework Spring, en Struts, nosaltres som responsables de decidir quina és l'opció que triem a la resta de capes.

[8] JSTL és un JSP amb etiquetes

Apartat
5

IMPLEMENTACIÓ D'UN FRAMEWORK

5.1 Descripció

El framework desenvolupat implementa el model MVC. Està escrit amb Java i funciona a partir d'un servlet. A més del model MVC, s'ha implementat un sistema de restricció d'accés a pàgines segons el rol de l'usuari. Pot ser desactivat i podem indicar quines pàgines són restringides, quines no i qui té accés. Atès que aquest framework l'he desenvolupat amb la prioritat de facilitar l'ús, he creat un mode de desenvolupament que es pot activar mentre estem treballant amb ell i desactivar-lo quan es faci el desplegament de l'aplicació.

Aquest mode facilita una pàgina amb l'error que s'ha comès i un suggeriment de com solucionar-ho. Amb això, el programador pot desenvolupar molt fàcilment, ja que sempre sabrà quin error s'ha produït i com pot solventar-ho. Evidentment, els errors de l'aplicació no han d'aparèixer un cop estigui finalitzada, ja que l'usuari final del programari no els ha de veure en cap cas.

5.2 Implementació MVC

5.2.1 Patró MVC

El patró està format per les tres parts que defineixen el nom del patró. Les parts en aquest framework estan implementades de la següent forma:

- **Controlador**: és el centre d'aquest framework. Està implementat a la classe NavegacioControlador i és un servlet. Aquest rep els paràmetres que li indiquem i segons els paràmetres que ha captat, triarà una acció o una altra. Al finalitzar aquesta acció, pren la sortida i triarà quina vista s'ha de carregar.

- **Vista**: la vista és el conjunt de les pàgines web en sí mateix. Són jsp i han de tenir la informació necessària per indicar al controlador quina acció hem de prendre.
- **Model**: El model conté els objectes de negoci del programari. En aquest framework hem d'utilitzar classes Java que deriven d'una interfície del framework i segons el que retornin definim quina pàgina hem de carregar.

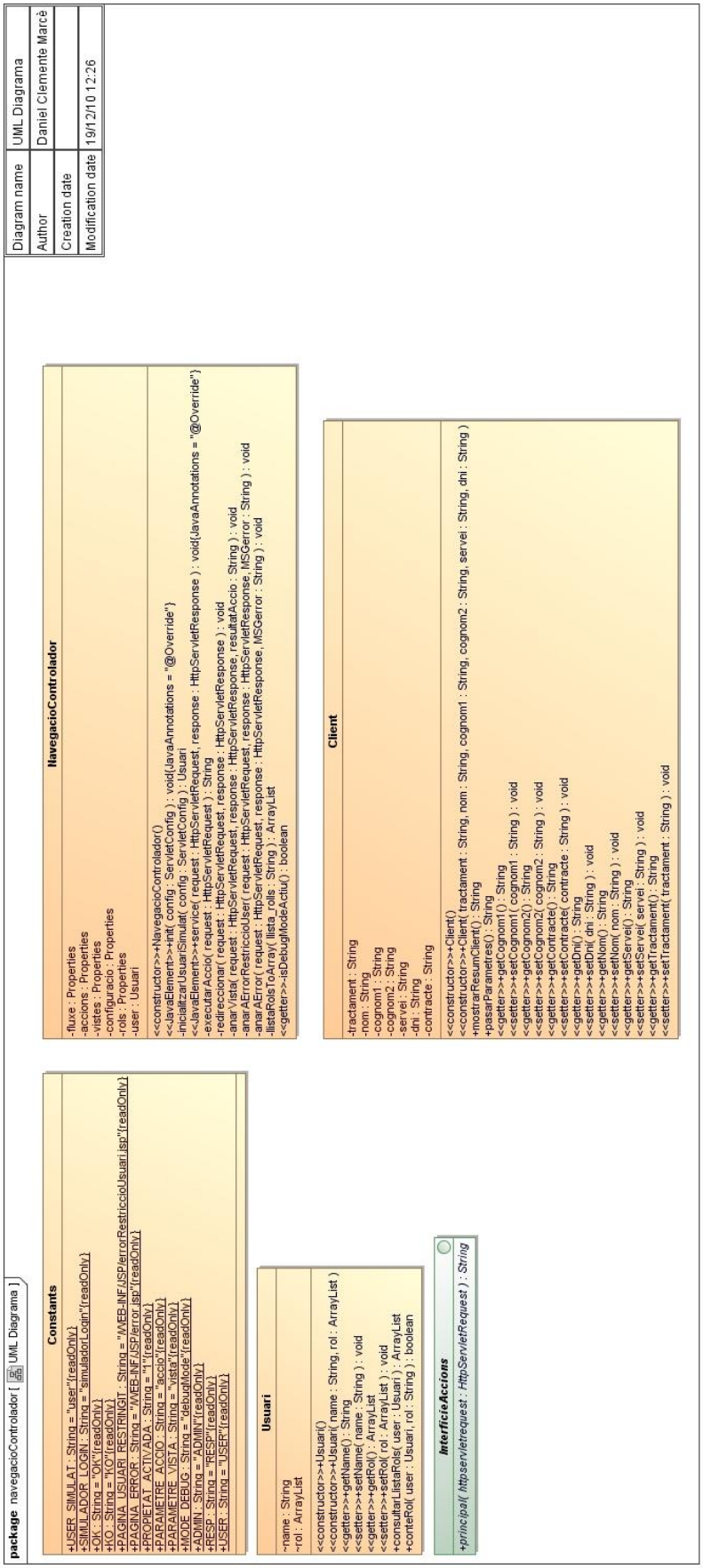


Figura 19: Diagrama UML del servlet

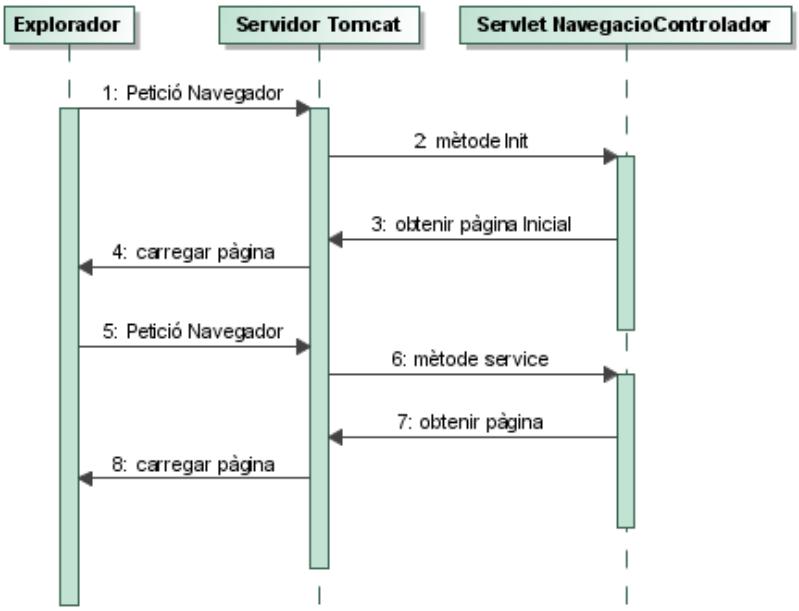


Figura 20: Interacció Navegador Servlet

5.2.2 Fitxers de configuració

Existeixen diferents fitxers de configuració en aquest framework. S'ha optat per utilitzar fitxer properties per a la realització de la configuració. Aquests són els següents:

Configuracio.properties

Aquest fitxer conté dos valors:

- **restriccioUsuari**: si està activat, podem definir quines pàgines volem restringir pel rol de l'usuari. Si no està activat, no restringirà cap pàgina.
- **debugMode**: si està activat, avisarà al desenvolupador dels errors que es vagin produint mentre ho està desenvolupant, i d'aquesta manera podrà aprendre a utilitzar aquesta eina molt ràpidament.

Addicionalment, té més atributs que es tractaran en el simulador de login (apartat 5.3.1) i restricció d'usuaris (apartat 5.3.2).

Accions.properties

Aquest fitxer conté la definició de les accions. Que hem d'entendre amb això? Doncs que hem de definir una acció amb un nom i donar-li com a valor quina classe Java és (amb el seu paquet).

Exemple:

DesarNouUsuari=uoc.pfc.danielclemente.pfcwebdemo.DesarNouUsuari

Definim l'acció DesarNouUsuari com la classe java DesarNouUsuari del paquet uoc.pfc.danielclemente.pfcwebdemo.

Vistes.properties

Aquest fitxer conté la definició de les vistes. De la mateixa forma que definíem les accions com a un nom i la seva classe corresponent, aquí li donem un nom a la vista i indiquem la seva ruta:

IndexPG=/WEB-INF/JSP/index.jsp

També li podem indicar si volem que la vista estigui restringida a una sèrie d'usuaris que tinguin un rol determinat:

TestPaginaRespPG=/WEB-INF/JSP/testResp.jsp,RESP,ADMIN

Aquesta opció es tracta amb més detall a la restricció d'usuaris (apartat 5.3.2).

fluxe.properties

Aquest fitxer conté a quina pàgina cal anar segons ha conclòs l'acció. La pàgina té una indicació per saber quina acció ha de prendre. Aquesta acció és una classe Java que s'ha d'executar i sempre ha de retornar un String. Aquest fitxer conté com a clau el nom de l'acció seguit per "." i el String que ha retornat aquesta acció. Com a contingut hem d'indicar el nom de la vista.

Login.OK=PrincipalPG

Si l'acció login retorna l'String "OK" hem de carregar la vista PrincipalPG.

Fitxer properties	Entrada	Valor	Descripció	Exemple
Configuracio	debugMode	1 està activat, i si no existeix l'entrada o pren altre valor es considera desactivat.	Ens mostra quin error ha aparegut i com podem resoldre'l.	debugMode=1
	restriccioUsuari	1 està activat, i si no existeix l'entrada o pren altre valor es considera desactivat.	Impedeix l'accés a vista per rols d'usuari.	restriccioUsuari=1
Accions	NomAccio	Paquet.classe	Indica a quina classe Java correspon una acció.	Login=uoc.pfc.danielclemente.pfcwebdemo.Login
Vistes	NomVista	ruta.fitxer.jsp	Indica on trobem el fitxer jsp corresponent a una vista.	PrincipalPG=/WEB-INF/JSP/principal.jsp
fluxe	NomAccio.ResultatAccio	NomVista	Indica a quina pàgina hem d'anar segons el resultat de l'acció.	Login.OK=PrincipalPG Login.KO=ErrorPG

Taula 2: Fitxers de configuració

5.2.3 Fluxe

En primer lloc, al fer una crida al navegador amb l'adreça que filtra el servlet (/NavegacioControlador definit al web.xml) s'executa el mètode init que inicialitza els fitxers de configuració i carrega la pàgina inicial (index.jsp també definit al web.xml).

A partir d'aquí, cada cop que el navegador fa una petició al servidor, s'executa el mètode service. Aquest mètode llegeix el paràmetre accio de la request, el

busca al fitxer de configuració accions.properties i recupera la classe java que ha d'executar. A continuació, executa la classe i recull el resultat obtingut. Quan executa aquesta classe podem comunicar-nos amb altres sistemes o accedir a B.DD. A partir del resultat i el nom de la classe busca quina pantalla ha d'accedir al fitxer fluxe.properties. Un cop sap a quina pantalla ha d'accedir busca la ruta de la vista a vistes.properties.

Altrament, podem trobar que no volem executar cap lògica de negoci i només volem redirigir-nos a una altra pàgina. Aquesta vegada, s'executa igualment el mètode service i enlloc de llegir el paràmetre accio (haurà de ser buit) llegirà el paràmetre vista. Cercarà el paràmetre vista al fitxer vistes.properties i carregarà la jsp que trobi a la ruta indicada.

Podem resumir-ho en els següents punts:

1. La pàgina fa una petició.
2. El controlador la recull (/NavegacioControlador).
3. Recupera el paràmetre accio.
4. Cerca aquest paràmetre a accio.properties i recupera el valor.
5. Executa la classe trobada com a valor del fitxer accio.properties.
6. El controlador recull el resultat de l'execució (és un String).
7. El controlador concatena el nom de la classe seguit de punt i seguit del resultat.
8. La cadena resultant de la concatenació la cerca a fluxe.properties.
9. El valor recuperat en el valor anterior és un àlies de la pàgina que volem situar-nos.
10. Cerquem l'àlies de la pàgina trobat a vistes.properties per trobar la ruta on es troba la jsp que volem carregar i la carreguem.

També podem tenir el cas que no vulguem executar cap lògica i decidim carregar altre pàgina. En aquesta situació:

1. La pàgina fa una petició.
2. El controlador la recull (/NavegacioControlador).

3. Intenta recuperar el paràmetre accio que no existeix. Al no trobar-lo recuperem el paràmetre vista.
4. Cerquem aquest paràmetre a vistes.properties i trobem la ruta on està la jsp que volem carregar, i la carreguem.

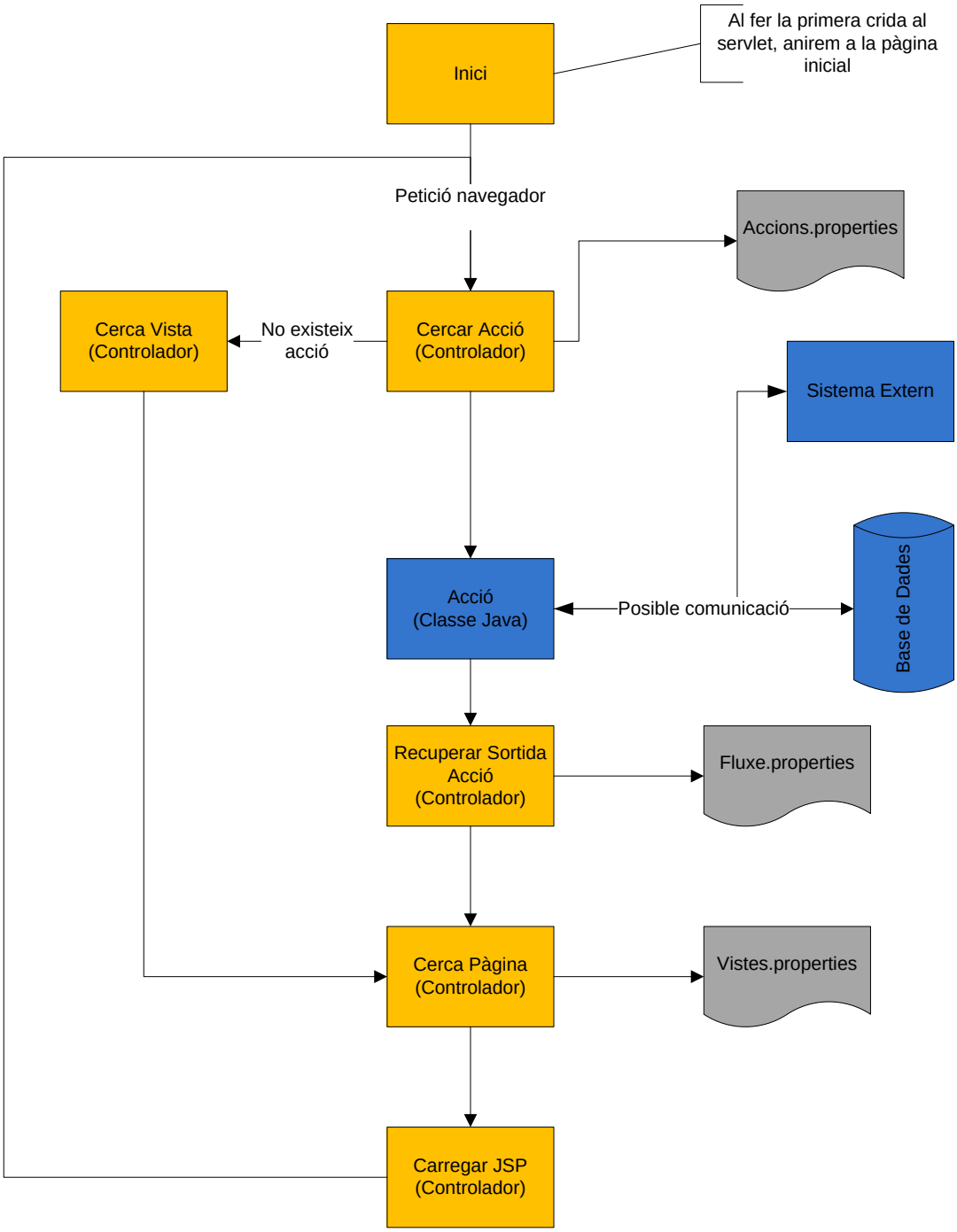


Figura 21: Flux del Framework desenvolupat

Al gràfic es mostra el flux del framework on:

- Taronja: són les execucions del controlador.
- Gris: són els fitxers de configuracions.
- Blau: són les classes Java externes al framework i que l'usuari haurà de crear o connectar.

5.2.4 Analogia amb Struts

Un desenvolupador familiaritzat amb Struts ho tindrà molt fàcil per treballar amb aquest framework ja que té un mètode de treball molt semblant.

Quan volem realitzar una classe a Struts, creem una classe que implementi a org.apache.struts.action.Action, de la mateixa forma que en aquest framework les classes estenen de uoc.pfc.danielclemente.navegacioControlador.InterficieAccions.

```
public class Login implements
uoc.pfc.danielclemente.navegacioControlador.InterficieAccions{ ...}

public class ExempleAction extends org.apache.struts.action.Action {...}
```

El mètode que hem de crear a Struts serà:

```
public ActionForward execute( ActionMapping mapping, ActionForm actionForm,
HttpServletRequest request, HttpServletResponse response) throws IOException,
ServletException {
       //codi
       ActionForward af = mapping.findForward("succeeded");
       return af;
}
```

En el nou framework és:

```
public String principal(HttpServletRequest request) {
       //codi
        String  resultat = "OK";
        return resultat;
 }
```

La classe que creem amb Struts ha de tenir un mètode execute amb 4 paràmetres i en el nou framework ha de tenir un mètode principal amb un

paràmetre. A Struts hem de retornar un ActionForward i en el nou hem de retornar un String.

La configuració a Struts es realitza mitjançant un fitxer xml anomentat struts-config, en el nou framework té quatre fitxers de configuració de tipus properties. Per definir el fluxe a Struts hem d'anar a action-mappings dins de struts-config.xml i indicar:

```
<action-mappings>
      <action path="/ruta" type= "paquet. ExempleAction ">
            <forward name="succeeded" path="/paginaA.jsp"/>
            <forward name="failed" path="/ paginaB.jsp"/>
      </action>
</action-mappings>
```

En el nou framework, definirem l'acció a accions.properties:

```
Login=uoc.pfc.danielclemente.pfcwebdemo.Login
```

En aquest framework, sempre estem a la mateixa ruta /NavegacioControlador i assignem un nom (àlies) a la classe. En canvi a Struts veiem com l'acció és la ruta path="/ruta" i el type és la classe a la qual ens referim.

El flux el definim a fluxe.properties:

```
Login.OK=PrincipalPG
```

El resultat de la classe, tal i com s'ha esmentat anteriorment, és un String i no un ActionForward. Addicionalment trobem que a Struts retornem la ruta directament i no un àlies. Al nou framework fem un pas més havent d'anar a cercar la ruta a vistes.properties. La justificació de crear un pas més és la comoditat de poder definir pàgines i la incorporació de la restricció d'usuaris, ja que també participa en aquesta funcionalitat (veure apartat 5.3.2):

```
TestPaginaRespPG=/WEB-INF/JSP/testResp.jsp,RESP,ADMIN
```

5.3 Funcionalitats

5.3.1 Simulador de login

Ja que la realització d'aquest projecte no inclou l'accés a Base de Dades, per poder crear la funcionalitat de "restricció d'Usuaris", he inclòs una forma de simular un login. Per utilizar-la, hem d'afegir al fitxer configuracio.properties l'entrada:

```
simuladorLogin=1
user=NomUsuari
```

Si simuladorLogin no es troba o pren altre valor, la funcionalitat estarà desactivada. Si activem aquesta funcionalitat, cal que li indiquem amb quin usuari volem estar logats, per això, hem d'indicar el nom de l'usuari que volem a l'entrada user.

Amb això aconseguim tenir una variable de tipus user accessible des del controlador, que conté el nom de l'usuari i els rols que pertany.

5.3.2 Restricció Usuaris

Una funcionalitat que s'ha incorporat ha estat la possibilitat de restringir l'accés a certes pàgines a grups d'usuaris.

Aquesta és una funcionalitat totalment opcional i activable en qualsevol moment. Per activar aquesta funcionalitat, només cal obrir el fitxer configuracio.properties i afegir el següent:

```
restriccioUsuari=1
```

Si té qualsevol altre valor o no existeix, aquesta funció estarà desactivada.

Al no tenir accés a Base de Dades (no està inclosa en aquest projecte) s'ha simulat al fitxer rols.properties quins rols té cada usuari. Així que, hem de crear una entrada per cada usuari i com a valor, una llista dels rols que cal que tingui cadascun dels usuaris separats per ",":

```
usuariC=USER,RESP,ADMIN
```

En aquest exemple l'usuariC (podem canviar d'usuari amb el simulador de login) té els rols USER, RESP i ADMIN.

Si volem que una pàgina només pugui accedir a un determinat rol, ho hem d'indicar al fitxer vistes.properties, introduint el nom de la vista com a clau i com a valor la ruta concatenant els rols als quals pot accedir separats per ",".

```
TestPaginaRespPG=/WEB-INF/JSP/testResp.jsp,RESP,ADMIN
```

Si només indiquem la ruta, aquesta vista no tindrà cap restricció.

Quan entrem a una pàgina a la qual no pot accedir l'usuari ja que no té el rol que requereix la pàgina, es mostrarà una pàgina indicant "Error; El teu usuari no té permisos per accedir a aquesta pàgina. Vostè té els rols: llistaDeRolsDeLUsuariLogat". A més, si el mode debug està activat, es mostrarà el missatge "Suggeriment: En el fitxer vistes.properties hi ha una entrada amb el valor NomPagina. Aquest valor de l'entrada ha d'estar seguit per , i el nom del rol que necessiti per accedir. En aquest cas la ruta/fitxer.jsp no te el teu rol: llistaDeRolsDeLUsuariLogat".

5.3.3 Mode debug

Intentant crear un framework el més senzill possible per l'ús del desenvolupador, s'ha implementat un mode debug que ajudarà al màxim alhora de realitzar el desenvolupament. Quan està activat, al produir-se un error, es carregarà un missatge amb l'error de forma entenedora i un suggeriment per a la seva correcció. Un cop finalitzat el seu desenvolupament, s'ha de desactivar per no mostrar aquest tipus de missatge a l'usuari final, en el cas que es produís algun error.

S'han estudiat tots els punts on poden aparèixer els errors i han aparegut 6 errors on s'indica l'error mitjançant el framework més un d'addicional degut a la

restricció d'usuaris que apareixerà sempre i que només mostrarà el missatge aclaridor si està activat aquest mode, altrament, només indicarà que no té permisos per accedir a aquesta pàgina.

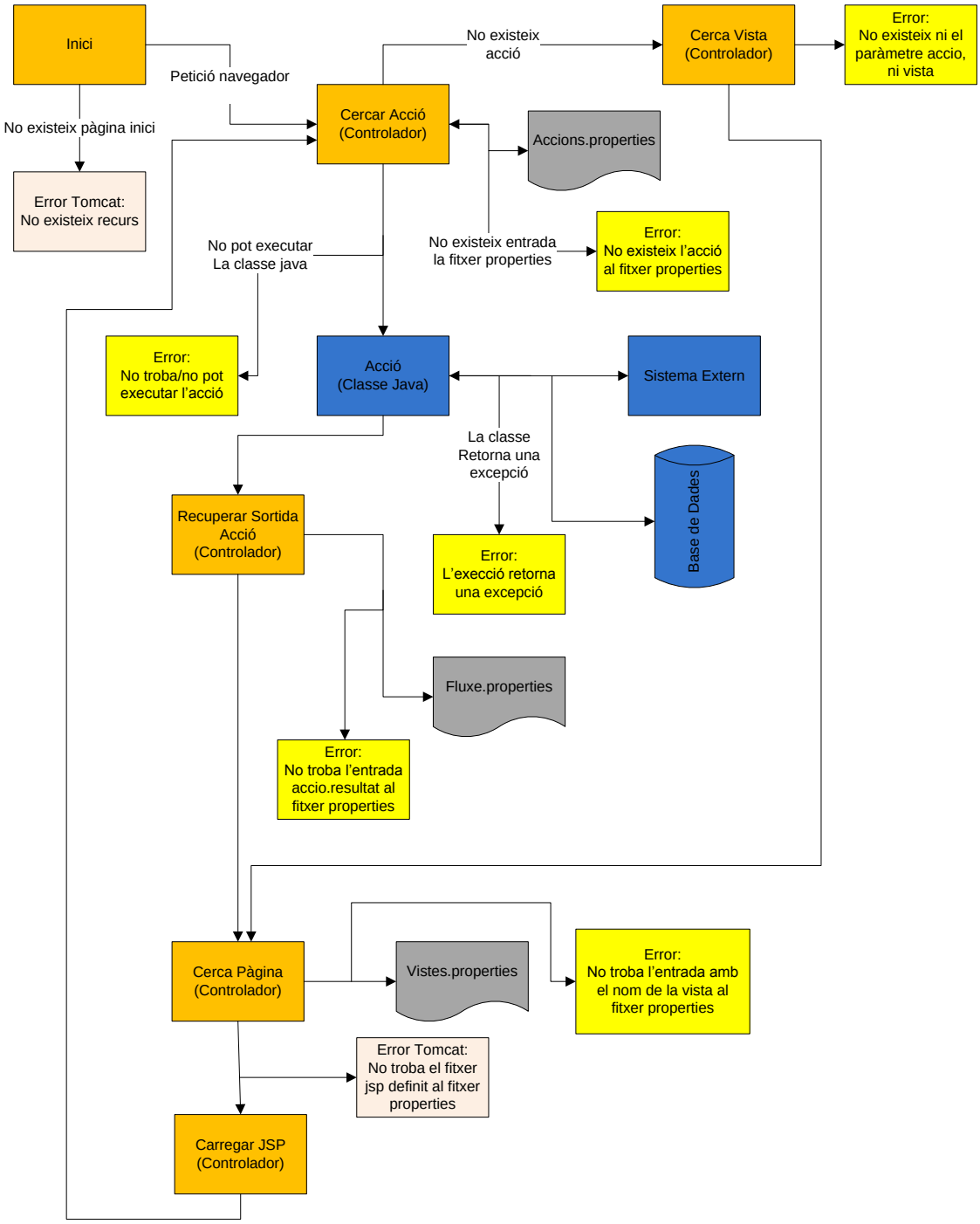


Figura 22: Flux amb aparició d'errors

Aquest gràfic mostra:

- Groc: són els errors identificats que el framework retornarà com error.
- Rosa: són els errors que el propi Tomcat retorna, en aquest cas, són molt aclaridors ja que en els dos possibles indica recurs no trobat i el nom del fitxer que no es troba.
- Taronja: són les execucions del controlador.
- Gris: són els fitxers de configuracions.
- Blau: són les classes Java externes al framework i que l'usuari haurà de crear o connectar.

Aquests errors poden ser:

- No existeix paràmetre a la jsp. Quan realitzem la petició al controlador, aquest cerca el paràmetre accio, i en cas de no existir o ser buit, el paràmetre vista. De no existir cap dels dos, es mostrarà aquest error amb el suggeriment: " introdueixi un paràmetre accio o vista a la jsp que l'ha redirigit fins aquí amb l'acció que necessita executar o la vista a dirigir-se. ".
- El paràmetre acció indicat a la jsp com a paràmetre no existeix al fitxer accions.properties. Volem executar una acció que no hem definit, el controlador no sap quina classe Java ha d'executar. Es mostrarà la suggeriment: " Introdueixi al fitxer accions.properties una entrada amb el valor parametreAccioRecuperat i posi quina classe correspon nompaquet.classe. ".
- No pot executar la classe definida. Al fitxer accions.properties hem indicat una classe que no pot executar. Això pot ser degut a que no existeix, no l'hem inclòs al projecte, no pot accedir o no és una classe que implementi InterficieAccions i/o contingui un mètode principal amb la capçalera de mètode public String principal(HttpServletRequest request). El suggeriment és: "En el fitxer accions.properties hi ha una entrada amb el valor TestErrorB i correspon a la classe paquet.incorrecte.TestErrorB que no existeix, no pot trobar o no pot executar".

- La classe que executem retorna una excepció. Capturem l'excepció i la mostrem indicant quina classe ha produït l'error. El suggeriment és: Revisi la classe :paquet.classe ja que ha retornat l'error: missatge retornat per l'excepció.
- La classe executada retorna un String que no està definit a fluxe.properties. Quan executem una classe, després cerquem a fluxe.properties la vista que ha de carregar. Per obtenir la clau concatenem l'àlies de l'acció seguit per "." seguit de l'String que retorna. Aquest error es produeix quan no existeix cap entrada amb aquest valor i el suggeriment és: "En el fitxer fluxe.properties no hi ha una entrada amb el valor AliesAccio.StringRetornat. Cal crear una entrada amb aquest valor ".
- No sabem quin fitxer jsp hem de carregar. Ja sigui pel resultat de fluxe.properties com recuperant el paràmetre vista a la jsp, hi ha un nom de vista que volem carregar. Aquesta vista té la seva ruta definida a vistes.properties, si no existeix una entrada amb el nom de la vista, no sap quin fitxer caldrà carregar. El suggeriment és: En el fitxer vistes.properties no hi ha una entrada amb el valor NomPagina . Cal crear una entrada amb aquest valor.

Altre error es produït per la restricció d'usuaris, aquest mostrarà que no té permís per accedir a aquesta pàgina i si està el mode debug activat indicarà què cal fer per poder accedir amb la suggeriment: "En el fitxer vistes.properties hi ha una entrada amb el valor Nom Pagina. Aquest valor de l'entrada ha d'estar seguit per , i el nom del rol que necessiti per accedir. En aquest cas la pàgina ruta/pagina.jsp no té el teu rol: llista de rols que té l'usuari". En el cas que l'aplicació estigui sent usada per l'usuari final, no indicarà com arreglar aquest problema, simplement indicarà que no té permís per accedir a aquesta pàgina (veure el detall a l'apartat 5.3.2).

Quan el servidor Tomcat no pot accedir a una pàgina, ens mostrarà clarament que no existeix aquesta pàgina indicant no es pot trobar el recurs

ruta/pagina.jsp. Això pot produir-se en dos punts, el primer si la pàgina d'inici no existeix i el segon si indiquem una ruta a vistes.properties que no existeix.

5.4 Resultats

L'avantatge principal que ofereix aquest framework és la seva facilitat d'ús. S'ha intentat prioritzar al màxim la corba d'aprenentatge d'aquest framework al desenvolupador. És molt senzill d'utilitzar i implementa el model MVC. Ens ofereix també l'avantatge de poder redireccionar sense haver de crear una classe intermitja, permetent-nos saltar aquest pas i incrementar la velocitat i reduir el codi.

Les funcionalitats addicionals són una restricció d'usuaris i un mode de debug per facilitar al màxim la tasca al desenvolupador seguint la filosofia del framework en prioritzar la facilitat d'ús.

Al funcionar amb Java i un servidor Tomcat, ens permetrà afegir altres funcionalitats amb les limitacions pròpies d'un sistema d'aquest tipus.

Apartat
6

CONCLUSIONS

En aquest Projecte Final de Carrera he estudiat els marcs de treball existents i creat un de nou. Amb la conclusió satisfactòria d'aquest projecte obtindré el títol d'Enginyer en Informàtica segons l'apartat 2 de l'article 6 del Reial Decret 1497/1987, de 27 de novembre degut a la finalització d'aquest estudis completant els 9 crèdits inclosos al pla docent corresponents a aquest Projecte.

En l'estudi, he analitzat prèviament els frameworks existents per tal de poder crear el propi. Aquest imprescindible estudi, és una part intrínseca pel desenvolupament, ja que el coneixement d'altres dóna les claus, a més de mostrar els requisits essencials pel posterior desenvolupament d'un marc de treball. Addicionalment, realitzant l'estudi he vist diferents alternatives; algunes no les coneixia i les he descobert, però sobretot m'ha donat l'oportunitat de saber els trets que els defineixen, el seu funcionament i les peces clau necessàries, per tal de crear un framework nou definint les seves característiques particulars. Per aquest nou desenvolupament, he optat per crear un framework, prioritzant la simplicitat i la corba d'aprenentatge. El meu bagatge professional com a professor en Formació Professional m'ha aportat la visió de les dificultats d'aprenentatge en eines tecnològiques, i per això he triat com a característica principal la facilitat d'aprenentatge, tant creant una eina que sigui molt senzilla d'utilitzar com aportant una eina que faciliti el seu ús.

Crec que en aquest treball he hagut de demostrar l'experiència, maduresa i criteri que he assolit al final d'aquest segon cicle d'enginyeria en Informàtica. Per poder evidenciar aquestes qualitats intrínseques en un professional he realitzat un producte que ho corrobori. La construcció d'un marc de treball vol dir deixar de banda la visió del desenvolupador d'aplicacions per posar-se al servei d'aquest, adoptant un altre rol, el d'arquitecte. Aquesta evolució significa

tenir una visió molt més àmplia sobre les tecnologies informàtiques actuals, fruit de l'estudi de l'enginyeria, de la feina de pedagog i de la meva experiència en consultoria de tecnologies de la informació i la comunicació.

En resum, la creació d'un framework ha estat una feina molt adequada i actual ja que és una peça clau en el paradigma de la programació actual.

Apartat
7

GLOSSARI

- **.NET:** Ens referim al framework Microsoft .NET que és un component software que pot ser afegit o estar inclòs al Sistema Operatiu Microsoft Windows. És l'entorn per la creació, distribució i execució de totes les aplicacions que suporten aquest entorn. El concepte de la tecnologia .NET de Microsoft, entre d'altres, ofereix la possibilitat que el programari modern pugui ser executat en un sistema de manera independent al maquinari (per exemple a una PDA o als aparells mòbils).

- **Apache Software Foundation** (Fundació de programari Apache) és la fundació sense ànim de lucre creada per donar suport als projectes Apache, incloent el popular servidor HTTP Apache. La ASF es va formar a partir de l'anomenat Grup Apache i es va registrar a Delaware (Estats Units), al juny de 1999. L'Apache Software Foundation és una comunitat descentralitzada de programadors que treballen els seus projectes de codi obert. Els projectes Apache es caracteritzen per ser programari lliure. Cada projecte es gestiona per un grup autoseleccionat d'experts tècnics que són participants del projecte.

- **API:** Una Interfície de Programació d'Aplicacions (Application Programming Interface, API), és un conjunt de declaracions que defineix el contracte d'un component informàtic amb qui farà ús dels seus serveis. Al moment de construir un sistema informàtic o llibreria de programació, per donar suport a les invocacions a serveis fetes per un altre programa, cal oferir una API, tant als programes externs (que podran usar els serveis oferts), com al programador (que disposa del

manual indispensable per poder treure el màxim del component que ha adquirit).

- **Framework:** és un terme adoptat de l'anglès i equival a entorn de treball o, també marc de treball. Aquest mot forma part de la terminologia tècnica utilitzada en múltiples àmbits de l'enginyeria i de la informàtica.

- **Hibernate:** és una solució implementada pel mapeig d'objectes relacionals (ORM) per aplicacions Java, sobre una base de dades relacional. Els seus propòsits bàsics són els d'alliberar el programador d'un seguit de tasques pròpies de la persistència de dades relacionals i dotar les aplicacions de portabilitat entre SGBDs diferents. Hibernate és lliure, de codi obert i està distribuït sota la GNU Lesser General Public License

- **HTTP:** El protocol de transferència d'hipertext (HyperText Transfer Protocol) estableix el protocol per al intercanvi de documents d'hipertext i multimèdia al web. Apareix al 1990 amb la versió referida com a HTTP/0.9 com un protocol pensat per a la transferència simple de dades a través d'Internet. No és fins a la versió referida com a HTTP/1.0 quan els missatges transferits són enviats en format MIME, el que implica l'enviament de metainformació conjuntament amb les dades transferides així com la capacitat de precisar el propòsit de les consultes (requests en anglès) entre el client i el servidor.

- **J2EE:** veure JEE.

- **Java:** és un llenguatge de programació dissenyat el 1990 a partir de C++. Des del seu naixement fou pensat com un llenguatge orientat a objectes. Entre el 13 de novembre de 2006 i el maig del 2007 Sun va alliberar parts de Java com a programari lliure de codi obert amb llicència GPL. És un dels llenguatges de programació més utilitzats, i s'utilitza tant per aplicacions web com per aplicacions d'escriptori.

- **JavaBeans:** són un model de components creat per Sun Microsystems per al desenvolupament d'aplicacions en Java. L'especificació de JavaBeans de Sun Microsystems els defineix com "components de programari reutilizables que es puguin manipular visualment en una eina de desenvolupament".

- **JEE**: Java Platform, Enterprise Edition o Java EE (va ser conegut com Java 2 Platform Enterprise Editino o J2EE fins la versió 1.4), és una plataforma de programació (una de les Plataformes Java) per desenvolupar i executar programari escrit amb el llenguatge Java amb una arquitectura distribuïda amb nivells, basada en components de programari, tot plegat executant-se en un servidor d'aplicacions.

- **JSF** (JavaServer Faces) és un marc de treball per aplicacions web basades en Java que simplifica el desenvolupament d'interfícies d'usuari per a aplicacions Java EE. JSF utilitza JavaServer Pages (JSP) com a tecnologia per fer el desplegament de les pàgines, però també pot utilitzar altres tecnologies, com per exemple XUL.

- **Framework** és un terme adoptat de l'anglès i equival a entorn de treball o, també, marc de treball.

- **Patró de disseny** és una solució general a un problema comú i recurrent en el disseny de programari. Un patró de disseny és una descripció o plantilla per resoldre un problema que es pot utilitzar en moltes situacions diferents.

- **Perl** és un llenguatge de programació dissenyat per Larry Wall. Fou alliberat al 18 de desembre de 1987 pel seu creador. Perl està inspirat en els llenguatges awk, C, sed, shell entre d'altres.

- **PHP** és un llenguatge de programació interpretat que s'utilitza per a generar pàgines web de forma dinàmica. S'executa a la banda del servidor, per aquest motiu al navegador web ja li arriba la pàgina en format HTML, no podent visualitzar-ne el codi php.

- **Python** és un llenguatge de programació d'alt nivell de propòsit general creat al 1991. Combina una potència remarcable amb una sintaxi clara i entenedora. Molt sovint és comparat amb altres llenguatges com Java, Tcl, Perl o Scheme. Utilitza sagnats com a delimitadors de blocs, fet poc freqüent als llenguatges de programació. Una altra de les característiques d'aquest llenguatge és el tipificat dinàmic i la capacitat per interpretar el codi en temps d'execució, en contradicció d'altres llenguatges com ara C, que ho fan en temps de compilació.

- **Ruby** va ser creat l'any 1993. És un llenguatge de guions totalment orientat a objectes. Està molt orientat al tractament de fitxers i per manteniment del sistema. És simple, extensible i portable.

- **Servlets** són miniaplicacions de servidor que consisteixen en objectes Java executats per un servidor d'aplicacions i que responen a invocacions HTTP, servint pàgines dinàmiques.

- **Struts** és una aplicació web de codi obert sota el patró de disseny MVC. Desenvolupada sobre la plataformaJ2EE (Java 2, Enterprise Edition).

- **Tag** o etiqueta és una paraula o conjunt de paraules que defineixen d'una forma clara i senzilla ja sigui l'article escrit, la imatge publicada o el document, dins d'un web.

- **Tomcat** (abans sota el projecte Apache Jakarta) és un contenidor de servlets desenvolupat a l'Apache Software Foundation. Tomcat implementa les especificacions de servlet i de Java Server Pages (JSP) de Sun Microsystems, proporcionant un entorn per al codi Java a executar en cooperació amb un servidor web.

- **Workflow** o Flux de treball (en anglès) és l'estudi dels aspectes operacionals d'una activitat de treball: com s'estructuren les tasques, com es fan, quin és el seu ordre correlatiu, com es sincronitzen , com flueix la informació que suporta les tasques i com se li fa seguiment al compliment de les tasques. Una aplicació de fluxos de treball (workflow)

automatitza la seqüència d'accions, activitats o tasques utilitzades per a l'execució del procés, incloent el seguiment de l'estat de cadascuna de les seves etapes i l'aportació de les eines necessàries per gestionar.

- **XML** de l'anglès eXtensible Markup Language («llenguatge de marques extensible»), és un metallenguatge extensible, d'etiquetes, desenvolupat pel World Wide Web Consortium (W3C). A l'actualitat té un paper molt important, ja que permet la compatibilitat entre sistemes, permetent de compartir informació d'una manera segura, fiable i fàcil.

Apartat
8

BIBLIOGRAFIA

8.1 Fonts utilitzades

Aspilcueta, A. M. (2007). *Struts: MVC en Java.* Consultat el 20 / Octubre / 2010, a http://www.slideshare.net/siis/struts-en-java

Buschman, F., & Sons, J. W. (1996). Pattern-oriented software architecture. A system of patterns. West Sussex: England.

Ceballos, F. J. (2000). *Java 2, Curso de programación.* Madrid : RA-MA.

Eckel, B. (2002). *Piensa en Java.* Madrid : Prentice Hall.

Foote, B. (1988). Designing to Facilitate Change with Object-Oriented Frameworks. Urbana-Champaign: University of Illinois.

Froute, A. (2002). Java 2, Manual de usuario y tutorial. Madrid : RA-MA.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Upper Saddle River, New Jersey: Addison-Wesley.

Gosling, J. (2001). *El lenguaje de programación Java.* Madrid: Addison-Wesley Iberoamericana.

Integración de JSF, Spring e Hibernate para crear una Aplicación Web del Mundo Real. (sense data). Consultat el 21 / Octubre / 2010, a http://www.programacion.com/articulo/integracion_de_jsf-_spring_e_hibernate_para_crear_una_aplicacion_web_del_mundo_real_307/3

Keogh, J. (2003). *J2EE. Manual de referencia.* Madrid: McGraw-Hill.

Lag, R. (Maig / 2007). *Java*. Consultat el 21 / Octubre / 2010, a http://www.proactiva-calidad.com/java/principal.html

Miquel, J. P., & Martos, J. A. (2005). Anàlisi i disseny amb patrons. A *Enginyeria del programari orientat a l'objecte.* Barcelona: Fundació per a la Universitat Oberta de Catalunya.

Miquel, J. P., & Martos, J. A. (2005). Catàleg de patrons. A *Enginyeria del programari orientat a l'objecte.* Barcelona: Fundació per a la Universitat Oberta de Catalunya.

Montenegro, E. M. (8 / Octubre / 2003). *CÓMO CREAR UNA APLICACIÓN CON STRUTS PASO A PASO*. Consultat el 20 / Octubre / 2010, a Adictos al trabajo: http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=strutsb

Roa, J., Gutiérrez, M., & Stegmayer, G. (2008). FAIA: Framework para la enseñanza de agentes en IA. *Revista Iberoamericana de Informática Educativa* , 43-56.

Apartat
9

ANNEXOS

9.1 Manual del desenvolupador

9.1.1 Preparació

Per realitzar aquest manual he utilitzat el Netbeans IDE 6.9.1.

Figura 23: Logotip NetBeans

Descomprimim el fitxer Framework-NavegacioControlador.rar i trobem una carpeta NavegacioControlador i un fitxer navegaciocontrolador.jar. Desem aquesta carpeta on vulguem crear l'aplicació i la renombrem amb el nom de l'aplicació que desitgem crear.

Obrim el NetBeans i anem a "Archivo", "Abrir Proyecto". A la finestra que s'obre cerquem la carpeta i premem "Abrir Proyecto". Al títol cliquem amb el botó dret, sel·leccionem "renombrar" i introduïm el nom de l'aplicació. Cerquem Web Pages→WEB-INF→web.xml des d'on podem canviar el servlet-mapping→url-pattern per /NomAplicacio per tal que l'aplicació final estigui més personalitzada.

Sobre el projecte cliquem amb el botó dret i seleccionem "Propiedades". A continuació, dins de la categoria "Libraries", anem a la pestanya "Compile", premem "Add Jar/Folder" i busquem el fitxer "navegaciocontrolador.jar". Opcionalment després podem editar per incloure el javadoc.

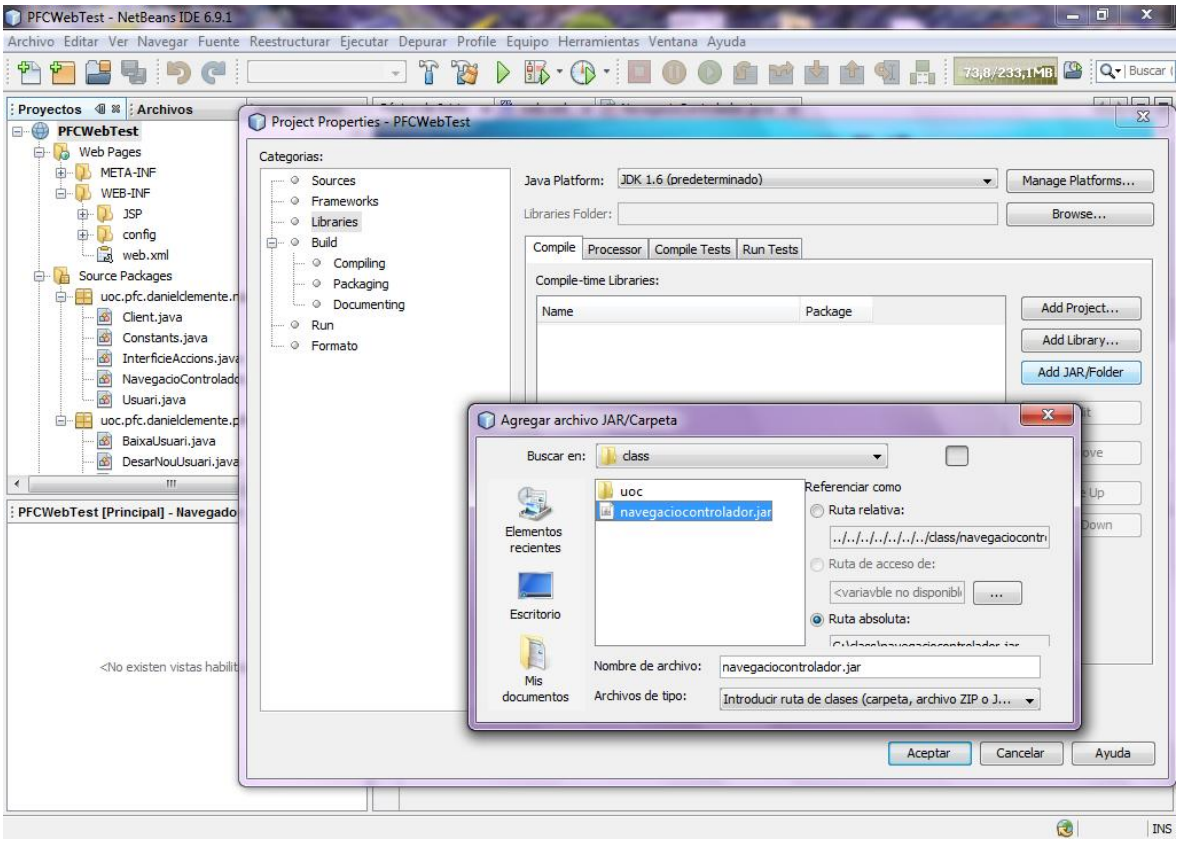


Figura 24: Captura de pantalla NetBeans IDE 6.9.1

9.1.2 Desenvolupament

Pàgina inicial

A web.xml tenim un element (welcome-file-list) on indiquem quina serà la pàgina d'inici.

```
<welcome-file-list>
<welcome-file>/WEB-INF/JSP/index.jsp</welcome-file>
</welcome-file-list>
```

Crear una jsp

Hem de crear un jsp. Creem una jsp i li definim un àlies al fitxer vistes.properties:

```
PaginaAPG=/WEB-INF/JSP/paginaA.jsp
```

Aquesta jsp haurà de tenir un paràmetre accio que indicarà l'acció que volem realitzar. Per fer això, podem fer-ho de dues formes, en funció de si ens interessa recollir les dades per pantalla o no.

En el cas que necessitem recollir les dades per pantalla, haurem de crear un formulari amb un paràmetre ocult anomenat accio i com a valor, l'acció a realitzar. També hem d'indicar que l'action del formulari ha de ser la URI del servlet.

```
<form name="formulari" action="/PFCWebTest/NavegacioControlador" method="post">
      <input type="hidden" name="accio"  value="DesarUsuari">
<!-- Codi formulari -->
<input type="submit" value="Desar"/>
 </form>
```

Si no necessitem recollir cap paràmetre, podem utilitzar un link indicant el paràmetre accio que volem introduir.

```
<a href="./NavegacioControlador?accio=LlistaClients">Veure clients</a>
```

En el cas que no vulguem executar cap acció i simplement necessitem redireccionar a una altra pàgina haurem de substituir el paràmetre accio per vista i com a valor l'àlies de la pàgina.

```
<li> <a href="./NavegacioControlador?vista=NouClientPG">Crear un nou
client&#42;</a></li>
```

Crear una classe java

Si hem indicat, que volem executar una classe Java, l'haurem de definir prèviament. Per fer això, anem al fitxer accions.properties i introduïm un àlies com a clau i com a valor el nom del paquet seguit del nom de la classe.

```
Login=uoc.pfc.danielclemente.pfcwebdemo.Login
```

Aquesta acció serà una classe de Java que s'executarà, permetrà recollir dades de la pàgina, accedir a la capa de dades i comunicar-se amb altres sistemes.

Creem la classe que vulguem, tenint en compte que ha d'implementar uoc.pfc.danielclemente.navegacioControlador.InterficieAccions i ha de contenir un mètode amb aquesta capçalera:

```
public String principal(HttpServletRequest request)
```

Tenint en compte que sempre ha de retornar un String.

```
public String principal(HttpServletRequest request) {
        //codi
        return "OK";
}
```

Crear el fluxe

Hem d'introduir al fitxer fluxe.properties la clau formada per l'àlies de la classe seguit de punt seguit del resultat de l'String i com a valor l'àlies de la pàgina que vulguem carregar.

```
Login.OK=PrincipalPG
```

Restricció usuaris

Si volem restringir l'accés a certs usuaris, tenim aquesta funcionalitat. Per activar-la hem d'introduir al fitxer configuracio.properties la clau restriccioUsuari amb valor 1.

```
restriccioUsuari=1
```

Per utilitzar-la només hem d'incloure al fitxer vistes.properties la llista de rols que volem limitar al costat de la ruta de la pàgina separat per comes.

```
TestPaginaRespPG=/WEB-INF/JSP/testResp.jsp,RESP,ADMIN
```

Si no indiquem cap rol, aquesta pàgina no tindrà cap limitació.

Simulador de login

Si no tenim cap sistema extern o encara no hem creat cap sistema propi de login, podem utilitzar el simulador. Per a fer-ho, podem activar-ho al fitxer configuracio.properties i incloure la clau simuladorLogin amb el valor 1, la clau user i com a valor el nom d'usuari que vulguem.

```
simuladorLogin=1
user=nomUsuari
```

Per indicar quins rols volem que tingui aquest usuari, hem d'introduir-ho al fitxer rols.properties indicant com a clau el nom d'usuari i com a valor la llista de rols separats per comes.

```
nomUsuari=USER,RESP
```

Mode debug

Aquest mode serveix per poder desenvolupar amb facilitat, ja que al cometre un error provocarem que el framework carregui una pàgina amb l'error i un suggeriment de com solucionar l'error. Per activar-ho, cal introduir-ho al fitxer

configuracio.properties indicant com a clau debugMode i com a 1. És important recordar treure-ho al posar l'aplicació finalitzada a mans dels usuaris.

```
debugMode=1
```